THE OPTIMAL CHOICE OF A GENERIC LASER SYSTEM FOR THE COMMERCIAL STERILISATION OF MICRO-ORGANISMS

Richard M. Farrar

A thesis submitted in partial fulfilment of the requirements of the University of the Wales, for the degree of Doctor of Philosophy.

This research programme was carried out at the University of Wales, Swansea under the supervision of Professor R. M. Clement, B.Sc., Ph.D., M.Inst.P., F.I.E.E., C.Eng. and in collaboration with ADAS Consulting Limited.

April 2003

DECLARATIONS

DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed  (candidate)

Date 28 April 2003

STATEMENT 1

This thesis is the result of my own investigations, except where otherwise stated.

Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed  (candidate)

Date 28 April 2003

STATEMENT 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loans, and for the title and summary to be made available to outside organisations.

Signed  (candidate)

Date 28 April 2003

ABSTRACT

University of Wales, Swansea

Department of Electronic and Electrical Engineering

Doctor of Philosophy

The Optimal Choice of a Generic Laser System for the Commercial Sterilisation of Micro-organisms

by

Richard M. Farrar

This thesis describes the theory and practical application of controlled laser radiation to denature micro-organisms on the surface of a given substrate in a commercial scale system. The described research system has been proven to achieve these aims tested on one specific bacterial example and one substrate.

The investigation evaluates the cell physiology of micro-organisms and the required physical and chemical parameters that are capable of denaturing them. A summary of the current state of sterilisation technologies with respect to their relative efficiencies and applications is presented, with specific attention being focused on the use of laser light sources and their associated photo-thermal effects. The optimal choice of laser wavelength is discussed together with its required spatial and temporal profiles to denature a broad range of micro-organisms.

The investigation centres on the use of continuous wave carbon dioxide lasers (with a wavelength of 10.6 μm) as the preferred choice to denature a variety of organisms, predominately S*almonella enteritidis*, residing on the surface of chicken eggs destined for hatching. The initial trials showed that a kill rate of 99.988 % (3.9 log) could be achieved.

The research system has been developed in conjunction with ADAS Consulting Limited (ADAS), which was formerly a government body affiliated to the Ministry of Agriculture, Fisheries and Food.

OBJECTIVES

The aim of this investigation was to ascertain the optimal parameters required of a generic laser system to sterilise a variety of surfaces that could contain a host of micro-organisms, without causing any damage or undesirable changes to the contaminated surface being treated.

The requirements of the study were to find the optimum laser source that could denature the greatest variety of organisms in the quickest and most efficient manner. Furthermore, such a system should also be capable of treating a plethora of substrates of numerous shapes and sizes, without any adverse effects to the substrates.

A further objective of the study was to investigate the means of delivering such laser energies to the required substrates while maintaining a uniform treatment of the substrate in order to achieve consistent sterilisation results.

ACKNOWLEDGEMENTS

I would like to acknowledge the following individuals for their help and support during this study, namely, Mr. Chris Williams and Prof. Marc Clement of ICN Photonics Limited (ICN) and Mr. Peter Redman and Miss Sue Tucker (who sadly died during the course of this study) of ADAS Consulting Limited (ADAS).

I would also like to thank the numerous individuals who kindly read and commented on this thesis and helped in the preparation of the research machine.

Finally, thanks must go to my wife Lynda for her encouragement and support throughout the period of my research, and my son James for all the hours I missed being with him during my studies.

"I wonder why I wonder why. I wonder why I wonder. I wonder why I wonder why I wonder why I wonder."

- Richard P. Feynman

To my son James.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

A/D	-	Analogue to Digital
ADP	-	Adenosine Diphosphate
ATM	-	Atmospheres
ATP	-	Adenosine Triphosphate
CAD	-	Computer Aided Design
CFUs	-	Colony-Forming Units
CO_2	-	Carbon Dioxide
COSHH	-	Control Of Substances Hazardous to Health
CW	-	Continuous Wave
D/A	-	Digital to Analogue
DNA	-	Deoxyribonucleic Acid
Er:YAG	-	Erbium Yttrium Aluminium Garnet
FIR	-	Far Infrared
HeNe	-	Helium Neon
LD	-	Lethal Dose
MRD	-	Maximum Recovery Dilutent
Nd:YAG	-	Neodimium Yttrium Aluminium Garnet
PCA	-	Plate Count Agar
PSI	-	Pounds per Square Inch
RF	-	Radio Frequency
RNA	-	Ribonucleic Acid
SAL	-	Sterility Assurance Level
SI	-	Système International d'Unités
TEA	-	Transverse Excited Atmospheric
TEM	-	Transverse Electromagnetic Mode
TMV	-	Tobacco Mosaic Virus
TVC	-	Total Viable Count
UV	-	Ultra-violet

1 : INTRODUCTION

1.1 Introduction

The principle of using high intensity light to denature living organisms has existed virtually since the discovery of the simple lens. Many a child has cruelly burnt ants with the aid of a magnifying glass, a clear day and a bright sun. Indeed, the power of the sun's rays harnessed through a simple lens was documented as far back as BC423 by a Greek writer Aristophanes (BC380 to BC448) who wrote a comedy, *Clouds*, in which a character used an object ("Burning glass") to reflect and concentrate the sun's rays, melting an "I owe you " recorded on a wax tablet.

Following the invention of the first practical laser in 1960 by T. H. Maiman [1], the ability of laser light to denature living cells was first demonstrated shortly afterwards by Saks and Roth in 1963 [2] using a pulsed ruby laser.

1.2 The Need

In today's society, there is an ever-increasing obsession for sterilisation and cleanliness. Things considered safe and clean only a few years ago are now considered dirty and therefore potentially dangerous. An example of this being the much-publicised controversy in the United Kingdom in 1988 over the S*almonella enteritidis* bacteria sometimes found on and in chicken eggs destined for human consumption.

The needs for sterilisation in modern society will continue to expand due to ever increasing media pressure. This will cross many boundaries from the sterilisation of surgical implements to that of foodstuffs, e.g. dairy produce and fruit juices. For most traditional applications of sterilisation, there exists the potential for an equivalent laser based sterilisation system.

1.3 Laser Advantages

Lasers have many advantages over many other more traditional sterilisation techniques, namely:

- Lasers are a non-contact process and therefore do not contribute to cross contamination of the items being treated.

- Being non-contact, lasers can treat physically delicate items that otherwise may be very difficult to process.

- Laser light can be readily transported from point to point with relative ease and a high degree of accuracy.

- Lasers can generate very high energy densities on substrates in very localised areas without effecting surrounding areas of the substrate.

- Lasers can generate very high energy densities within very short time periods without localised areas of substrates having sufficient time to heat up.

- Lasers have a benign public image in comparison to alternative technologies such as ionising radiation.

These features provide many benefits in laser materials processing, although some of the perceived benefits could become disadvantages in practical applications. For example, the ability of a laser system to produce very high energy densities in localised areas would be a distinct disadvantage if a large surface area needed to be processed with the same high energy density.

1.4 Existing Knowledge

While the application of laser technology for cell denaturation has been studied for many years by a variety of researchers [2] - [6], it has yet to be proved an efficient system suitable for commercial applications.

The effects of numerous laser systems [7] on a multitude of micro-organisms [6] have been investigated in recent years. The results conclusively prove that lasers can permanently denature living organisms. However, the choice of laser system is crucial to the sterilisation efficiency experienced. Moreover, the range of organisms that could be encountered makes the choice of a generic laser system, which will be commercially viable, somewhat more difficult.

1.5 Investigation Aims

The aim of this investigation was to review the current state of technology in this field and to expand on the knowledge to date, with experimental trials conducted on a large-scale laser based sterilisation system capable of denaturing a wide range of micro-organisms.

The experimental system was specifically constructed for the purpose of these trials. All trials were conducted on hatching eggs destined for breeding stock. A range of trials on artificially contaminated eggs was conducted. These trials were designed to assess the optimal laser parameters required from a generic laser system to achieve maximum, reproducible kill rates of selected bacteria.

To this aim, the investigation has achieved the original objective. The evaluation system constructed allowed a number of laser treatment parameters to be simultaneously varied on a semi-commercial scale to assess the viability of a generic laser device – namely a carbon dioxide (CO_2) laser.

The results of these trials show conclusively that such a system is a commercial viability, and with continuing refinement, could be applied to numerous applications beyond the experimental trials with hatching eggs for breeding stock.

2 : CELL PHYSIOLOGY AND STERILISATION METHODS

2.1 Introduction

In order to gain a deeper understanding of the current and proposed sterilisation methods for micro-organisms, a firm background in micro-organism cell physiology is required. This background is essential for the informed analysis of the primary sterilisation requirements for micro-organisms.

The first section of this chapter reviews the basic elements of micro-organism's cell physiology. In this section, the requirements for cell growth and reproduction are evaluated, together with the external factors effecting the cell's behaviour and life span. The primary focus of this section is the external factors that can be applied to micro-organisms in order to promote necrosis.

The second section of this chapter reviews current, commercially available sterilisation methods. The latter part of this section reviews new technologies for sterilisation that may not have been commercially implemented to date, or are still in their infancy. This section forms the existing knowledge base for sterilisation methods, upon which the proposed methods are founded.

Figure 2.1 shows typical bacterial and viral sizes in relation to other common particles [8]. This gives a useful comparison when trying to visualise the sizes of particular micro-organisms in relation to commonly occurring particles or objects, and provides some idea as to the scale of the micro-organisms that are subject to various sterilisation techniques.

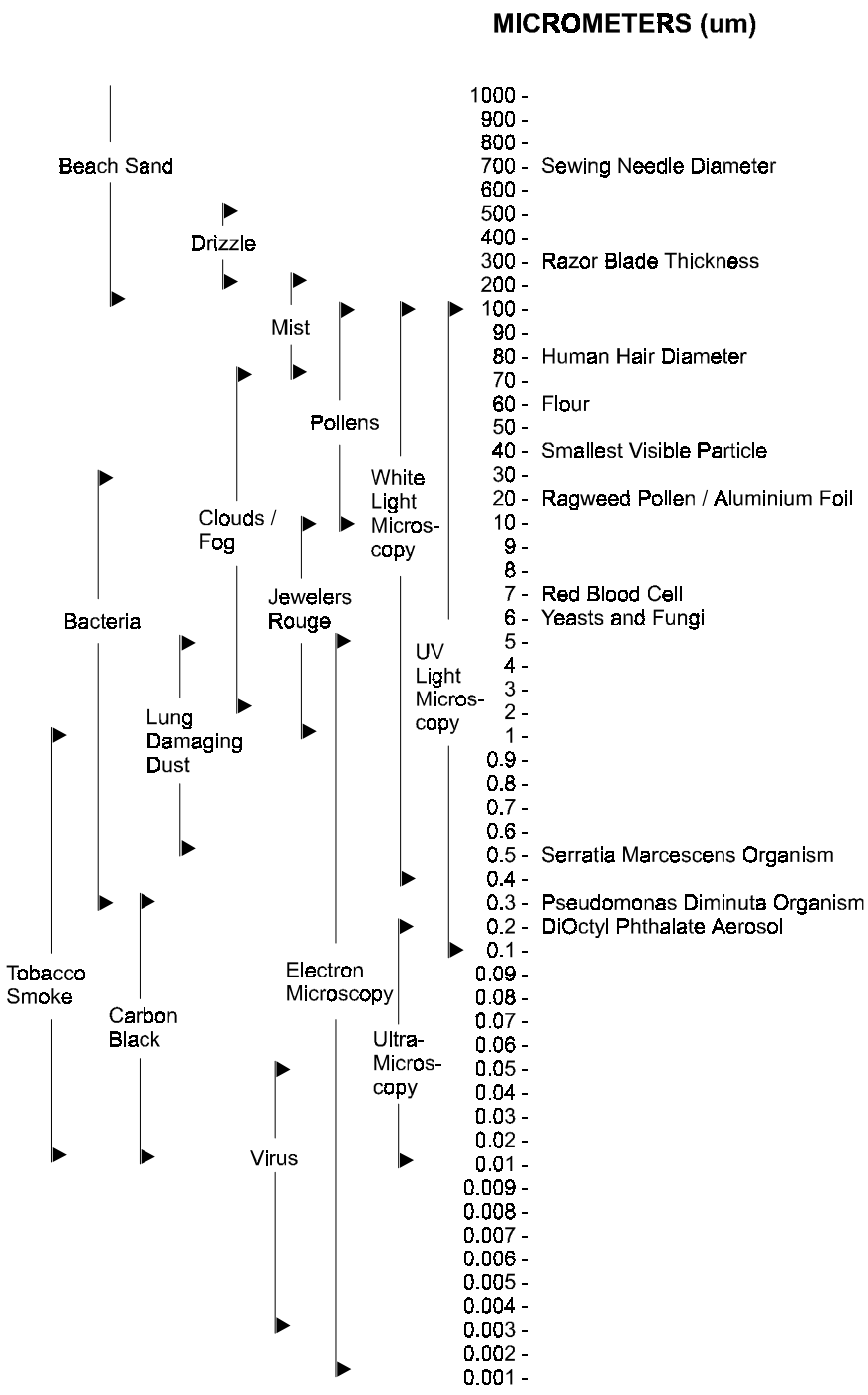

Fig. 2.1: Relative sizes of common particles and objects.

The following section will now discuss basic cell physiology, concentrating on bacteria with a more general overview of viruses and fungi.

2.2 Cell Physiology

Basic cell physiology is not a new science and has been documented since as far back as 1665 when Hooke first described the existence of cells as "empty structural units" [9]. There now exists a plethora of textbooks and papers on almost every conceivable area of cell physiology. It is beyond the scope of this programme of study however, to go into any great depth on cellular biology. Rather, the aim of this section is to gain an overview of the cell physiology in particular relation to pathogenic micro-organisms and the sterilisation of the aforesaid organisms.

The following three sections outline the physiology of bacteria, viruses and fungi respectively, with particular attention being paid to bacteria.

2.2.1 Bacteria

Bacteria were first discovered in 1674 by Anton van Leeuwenhoek [10]. It was not until the eighteenth century however, with the invention of the compound microscope, that serious study and classification of bacteria and other micro-organisms began.

Bacteria are members of the *protista* kingdom, which also includes algae, fungi, protozoa, slime moulds and blue green algae. This kingdom is further subdivided into *higher protists* and *lower protists*. A higher protista's cell nucleus is surrounded by a nuclear membrane for most of the cell's life. Lower protists however, do not have this membrane. Their nucleus is in direct contact with the cell's cytoplasm.

Figure 2.2 shows the three kingdoms of plants, animals and protists, with the closer relation of some protists to either the plant or animal kingdoms [10].

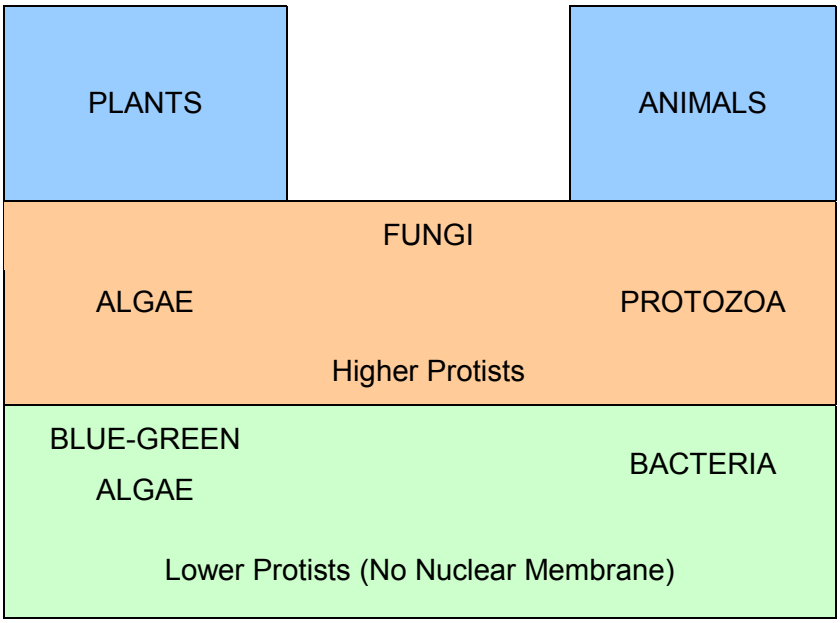

Fig. 2.2: Biological kingdoms of plants, animals and protests.

Algae, slime moulds and protozoa are beyond the scope of this thesis, with the concentration being primarily on bacteria, particularly pathogenic bacteria, with a brief discussion of fungi. Viruses will also be covered, although they are not strictly living organisms.

Bacteria are organisms that posses rigid cell walls and no nuclear membrane. Motile bacteria have *flagella*, which are the organelles of motion. The key purpose of the bacteria's life is to reproduce - this they achieve with incredible speed and efficiency.

Bacteria come in three primary morphologies; rods (*baccili*), helices (*vibrios* or *spirilla*) and spheres (*cocci*, literally translated from the Latin for "berry"). Rods are the most common variety. Spirilla have more than one turn of a helix while vibrios have less than one turn of a helix. An example of a spherical bacterium is *Pneumoccocus* (singular) or *Pneumococci* (plural), which is the bacterium responsible for causing pneumonia.

Louis Pasteur and Robert Koch, who both pioneered the early work in microbiology, showed that infectious diseases were caused by living organisms or *"germs"* (i.e. pathogenic bacteria) [11]. The harmful effects of many bacteria causing disease in animals are due to the direct or indirect production of toxins by the bacteria. These may be **endotoxins**, which are formed in the bacterial cells and only released when the bacteria die and disintegrate, or **exotoxins,** which are excreted by the bacterial cells during normal metabolism. These toxins all tend to be proteins [12].

While sizes of bacteria vary greatly through the different strains of bacteria, their typical size is of the order of 0.5 μm to 30 μm, with a typical volume of 2 μm^3 and weight of 20×10^{-7} g. Typically 80 % of this weight is water. Table 2.1 outlines the sizes and morphologies of a few common bacteria [12].

Table 2.1: Comparative sizes and morphologies of bacteria.

Bacteria	**Morphology**	**Size (μm)**
Staphylococcus albus	Spheroidal	1
Mycobacterium tuberculosis	Rod	2.5 to 3.5 × 0.3
Salmonella typhi	Rod	2 to 4 × 0.5
Escherichia coli	Rod	2 to 4 × 0.5
Clostridium botulinium	Rod	3 to 8 × 0.6 to 1
Aspergillus niger spore	Spheroidal	2.5 to 4
Botrytis cinerea spore	Ellipsoidal	8 to 12 × 6 to 10

Being very small bacteria have a very large surface area to volume ratio. For a given mass of cells, a larger surface area equates to a faster exchange of substances via the cell walls [10]. This is very efficient for the bacteria during normal metabolism, but can also be an advantage in the destruction of bacteria.

The following sections of this chapter give a brief outline of the physiology of a typical bacterial cell. Figure 2.3 shows a typical bacterial cell. This is a rod shaped cell, *Escherichia coli* bacterium, which has been the most commonly studied bacterium.

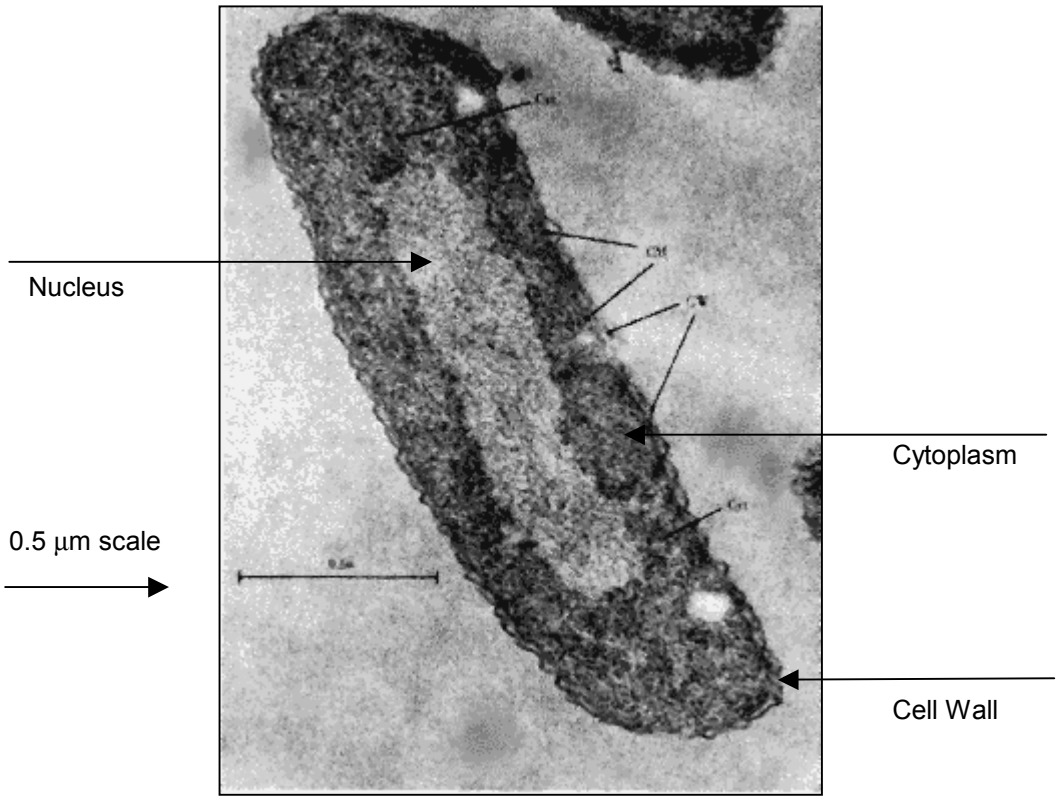

Fig. 2.3: *Escherichia coli* cell.

2.2.1.1 Bacterial Capsule

A capsule formed from polysaccharides (sugars) or polypeptides surrounds the cells of many bacteria. These capsules act as protective barriers to the cell. These capsules however can vary in size on the same strain of bacteria, with some bacteria losing the ability to form these capsules during their life. Cells with capsules are more resistant to attack from animal phageocytes (viruses) than those cells without capsules [10].

2.2.1.2 Gram Stain Reaction

The Gram reaction is a staining test performed on bacteria using the Gram stain, named after its inventor Christian Gram, who invented the process in 1884 [13]. This procedure involves staining bacteria with a *para-rosaniline* dye, such as crystal violet, at a slightly alkaline pH, to produce a blue stain. This will stain all bacteria blue. The next stage of the procedure involves treating the stained bacteria with iodine in a solution of potassium iodide (or alternatively picric acid), to fix the stain. The final procedure is to decolourise the stained bacteria with alcohol or acetone. The results provide two classifications of bacteria:

- *Gram-positive* - these bacteria remain coloured
- *Gram-negative* - these bacteria become decolourised

These classifications can be very useful when characterising bacteria. For example, all spore forming bacteria are Gram-positive and all polarly flagellated bacteria are Gram-negative. Gram-negative bacteria tend to be more resistant to antibiotics, chemical and physical attack and have differently structured cell walls than Gram-positive bacteria.

2.2.1.3 Flagella

The most common form of motility for bacteria is via flagella (from Latin meaning "whip"). Bacteria swim using these flagella and can obtain velocities of up to 20 $\mu m \cdot s^{-1}$. Different types of bacteria have different organisations of flagella. These flagella are typically 10 nm to 20 nm thick, 4 µm to 5 µm long and helical in shape. Their construction is entirely protein.

Bacteria with flagella can be sub-divided into two further classifications:

- *Polar flagella* - these flagella are restricted to one or both ends of a bacterium. This organisation of flagella is only found on the rod and helical bacteria morphologies.

- *Peritrichous flagella* - these flagella are found around the complete bacterium and are general for all bacteria.

2.2.1.4 Pili

Many Gram-negative bacteria have hair-like *pili* (from Latin for "hair") or *fimbriae* (from Latin for "fringe"). Each bacterium may have several hundred of these structures. These structures are of the order of 3 nm to 5 nm in diameter (compared to flagella which are 10 nm to 20 nm in diameter) and several microns long. Like the flagella, they are constructed entirely from protein. Their main purpose seems to be in aiding the bacteria in attachment, for example, they are often seen in bacteria that remain near the surface of a liquid, where there is an increased supply of oxygen.

Figure 2.4 shows an electron microscope photograph of a dividing cell that has two flagella and about 200 pili. The bacterium in this photograph is S*almonella anatum*.

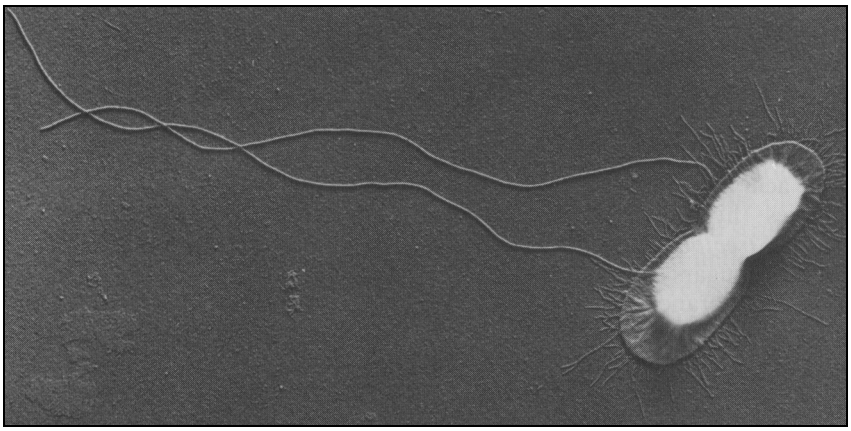

Fig. 2.4: Dividing bacterium with flagella and pili.

2.2.1.5 Endospores

Endospores are a unique feature of bacteria. The endospore is a resistant spore that has little free water and can withstand both chemical and physical extremes that normal vegetative cells could not. The endospore performs no metabolism. This is in effect a built in defence mechanism of survival, triggered when a bacterium finds itself in an increasingly hostile environment. For example, an endospore can still germinate after several hours in boiling water.

Having little free water, the endospores are significantly denser than the host bacterium. Moreover, the endospore is very highly refractile to light and is in what is termed a *"glassy state"*. Only one endospore is found per bacterium cell, and will typically form one quarter to one half of the cell's volume. The cell in which an endospore is found is a *sporangium*.

A spore requires a suitable medium and environmental conditions to germinate. In germination, the spore soaks water and swells. In doing so, it looses refractility and decreases in density. Finally, the spore coat ruptures and a new vegetative cell emerges and grows.

Some forms of bacteria, for example *Azotobacter,* have cysts formed from the entire cell. These perform a similar task to endospores, but are less resistant to environmental extremes. Endospores are mainly found in bacteria of the *Clostridium* and *Bacillus* genera. The *Clostridium* genera are strict anaerobes, while the *Bacillus* genera are aerobes.

2.2.1.6 Cell Walls

The bacterial cell membrane is surrounded by a rigid cell wall. This cell membrane (or plasma membrane) is semi-permeable. The cell wall is a purely mechanical structure, which has been demonstrated by removing the cell wall with the organism continuing to survive, but adopting a spherical shape.

Being semi-permeable, the cell membrane will allow the exchange of fluids by osmosis. The rigid cell wall is very strong, and has been shown to withstand up to 300 psi (pounds per square inch) or 20 atm (atmospheres) of osmotic pressure when placed in pure water [10].

Bacterial cell walls are approximately 5 nm to 10 nm thick and constructed from polysaccharides (sugars) such as glucose, galactose and mannose. Some of these however, can be the amino sugar varieties rather than simple sugars, for example glucose being the simple sugar, with glucosamine being the amino variety.

The cell walls of Gram-negative bacteria are more complex chemically and architecturally than Gram-positive bacteria. They are also stronger. For example, the *Escherichia coli* bacterium has at least two layers in the cell wall, providing a much stronger wall, with this particular bacterium being Gram-negative.

2.2.1.7 Nucleus

Bacteria have no nuclear membrane. The nucleus is the area of the cell that contains the localisation of genetic material, and contains a single deoxyribonucleic acid (DNA) molecule. Bacteria can have from one to three nuclei; all located more or less centrally to the cell.

2.2.1.8 Metabolism and Reproduction

Metabolism and reproduction of bacteria are key to this programme of study. If either of these is prevented from occurring, then sterilisation can be considered to have taken place. By definition, sterilisation is the power to *"deprive of power of reproduction"* [14]. Preventing a bacterium from performing metabolism will also prevent the bacterium from reproducing, as the cell will have no energy to perform the biosynthesis required for reproduction.

Cells are self-replicating, but in order to replicate the cell needs to obtain necessary material from its environment. Moreover, the cell will require energy to transform the material gained from the environment into viable cell material. This energy is also gained from material in the environment.

In the process of metabolism, a bacterial cell acquires material from the environment, which can generate *exergonic* or *catabolic* reactions - i.e. reactions that yield energy. This energy is trapped in the formation of ATP (adenosine triphosphate) from ADP (adenosine diphosphate) and phosphoric acid. This is an *endergonic* or *anabolic* reaction (i.e. a reaction that absorbs energy). This ATP can be later used by the cell for biosynthetic reactions requiring energy, which can be gained from the decomposition of the ATP molecule. In this manner ATP plays a pivotal roll in the metabolism and growth of the cell, and can be considered as an energy storage medium for the cell [15].

The oxidation of foodstuffs to generate energy is the key respiratory process in living organisms and is an exergonic reaction. This process may take place with or without the presence of oxygen, provided there are hydrogen receptors to remove hydrogen from the glucose molecule and hence oxidise it. Organisms that use oxygen as part of the respiratory process are classed *aerobes*, while those that do not are called *anaerobes*. Strict anaerobes are poisoned by air or oxygen and are unable to grow in its presence.

As the cell metabolises, the continuous process of biosynthesis will create new cellular material. Eventually the cell will reach a level at which binary fission occurs. One cell now becomes two, with the process now continuing in the two separate cells.

In order for cells to grow, there needs to be a sufficient supply of nutrients together with equitable physical conditions such as temperature. Cells typically have a three-phase development cycle when introduced into a new medium.

The first phase of a newly inoculated culture is the *lag phase*. During this time, no growth occurs while the cells take several hours to adapt to their new environment. During the second *log phase*, cells reproduce rapidly. During this phase, the logarithmic plot of cells with respect to time is a straight line, hence the name log phase. The final phase is the *stationary phase*. During this phase, the numbers of cells in the culture remain constant. In a closed system, this may be brought about by changes in pH, exhaustion of nutrients or accumulations of retarding agents.

Analysing the slope of the log phase, it is possible to determine the phase *generation time* of a culture of organisms. This generation time is the frequency with which the cells divide and is mainly determined by supply of nutrients and temperature. Table 2.2 shows typical generation times for organisms under optimum conditions. If a culture is held at a reduced temperature for a while, cell division ceases, but cell growth may continue at a reduced rate. When the temperature is reinstated, all cells will simultaneously divide synchronously. This is classed as a *synchronised culture.*

Table 2.2: Generation time of organisms under optimal conditions.

Bacteria	Generation Time
Escherichia coli	20 min
Mycobacterium tuberculosis	18 h
Saccharomyces cerevisiae	2 h
Schizosaccharomyces pombe	4 h

2.2.1.9 Temperature Effects on Growth

Varying temperatures can have marked effects on different organisms. Different organisms will all have different optimal temperatures for sustained growth. There are three key classifications of organisms with respect to temperature tolerances:

- *Psychrophils* are organisms with optimal temperatures lower than 20 °C. Some will continue to grow, albeit more slowly, at very low temperatures.
- *Mesophils* include most micro-organisms and have optimal temperatures in the range of 20 °C to 45 °C.
- *Thermophils* have optimal temperatures of 45 °C and above, some of which can survive at temperatures sufficiently high to kill most fungi and bacteria.

Table 2.3 shows comparative cardinal temperatures for a range of micro-organisms.

Table 2.3: Cardinal temperatures for vegetative growth of various organisms.

Micro-organism	Minimum Temp. (°C)	Optimum Temp. (°C)	Maximum Temp. (°C)
Clostridium thermocellum	50	60	68
Thermomyces lanuginosus	28 to 32	45 to 50	58 to 60
Rhizomucor sp.	25 to 30	45 to 50	60 to 61
Mucor pusillus	21 to 23	45 to 50	50 to 58
Chaetomium sp.	25	40 to 50	62
Escherichia coli	10	37	45
Mycobacterium tuberculosis	30	37	40
Saccharomyces cerevisiae	1 to 3	28	40
Fusarium caeruleum	5	20	30
Cladosporium herbarum	-6	Unknown	20

2.2.1.10 Aeration Effects on Growth

This effect does not always come into play, depending upon the organism's requirements for oxygen, its supply and the physical environment in which the organisms are resident. This is typically most important in liquid cultures.

If organisms cannot obtain sufficient oxygen for their requirements, then reproduction and cell growth will cease, with eventual death of the cells under prolonged conditions.

2.2.1.11 Light Effects on Growth

Light is generally not important for micro-organisms, other than the photosynthetic varieties. On most organisms, light has little or no effect on growth.

However, bacteria and many other organisms are susceptible to ultra-violet (UV) light and *x*-rays, which readily kill them. Exposure to sub-lethal doses of ultra-violet light may cause mutations of the organisms. Taking account of this effect, ultra-violet light is commonly used as a bactericidal agent (discussed later in this chapter).

2.2.1.12 Salmonella

As the main commercial trials planned for this study were to utilise chicken eggs from breeding stock contaminated either naturally or artificially with S*almonella enteritidis* bacteria, this section will discuss the *Salmonella* bacteria specifically in detail.

Salmonellae are rod shaped, motile bacteria with a few non-motile exceptions such as *Salmonella gallinarum* and S*almonella pullorum*. *Salmonellae* are non-spore forming, Gram-negative and are facultatively anaerobic. They are resilient micro-organisms that readily adapt to extreme environmental conditions. *Salmonellae* actively grow within a wide temperature range ≤ 54 ºC and also exhibit psychrotrophic properties, as reflected in the ability to grow in foods stored at 2 ºC to 4 ºC [16]. However, their growth is slow at temperatures below 10 ºC. There is a widespread occurrence of these bacteria in animals, especially in poultry and swine. Environmental sources of the organism include water, soil, insects, factory surfaces, kitchen surfaces, animal faeces, raw meats, raw poultry, and raw seafoods, to name only a few.

Salmonellae consist of a range of very closely related bacteria. This means that they all belong to the genus *Salmonella*, a division that groups similar, though not identical bacteria together. These bacteria are named after the scientist who discovered them, Dr. Daniel E. Salmon. The majority of the components of these bacteria are identical, and at the DNA level, they are between 95 % and 99 % identical (as a comparison *Escherichia coli* and *Salmonella*, which are closely related to each other, are about 60 % to 70 % identical at the DNA level) [17]. The S*almonella* family includes over 2 300 serotypes (Table 2.4) of bacteria, many of which cause disease in humans and animals [16].

Table 2.4: Species of the genus *Salmonella*.

Species	**No. of serovars**
Salmonella enterica	
subsp. e*nterica*	1 405
subsp. s*alamae*	471
subsp. a*rizonae*	94
subsp. d*iarizonae*	311
subsp. h*outenae*	65
subsp. i*ndica*	10
Salmonella bongori	19
	2 375

The majority of *Salmonella* species are generally different serovars of *Salmonella enterica*. As their name suggests *Salmonella enterica* are involved in causing diseases of the intestines ("enteric" means pertaining to the intestine). The three main serovars of *Salmonella enterica* are ***typhimurium***, ***enteritidis***, and ***typhi***. Each of these is discussed further below.

To complicate matters, serovars of *Salmonella enterica* can be subgrouped even further by "phage type". This technique uses the specificity of phage to differentiate between extremely closely related bacteria. Often these bacteria are indistinguishable by other means, and indeed, the reasons for the differences in phage specificity are often not known.

- ***Salmonella enterica*** serovar ***typhi*** (also called *Salmonella typhi* or abbreviated to *S. typhi*). This bacterium is the causative agent of typhoid fever. Although typhoid fever is not widespread in the western world, it is very common in under-developed countries, and causes a serious, often-fatal disease. The symptoms of typhoid fever include nausea, vomiting, fever and death. Unlike the other *Salmonellae* discussed below, *S. typhi* can only infect humans, and no other host has been identified. The main source of *S. typhi* infection is from swallowing infected water. Food may also be contaminated with *S. typhi*, if it is washed or irrigated with contaminated water.

- ***Salmonella enterica*** serovar ***typhimurium*** (also called *Salmonella typhimurium* or abbreviated to *S. typhimurium*). Until recently the most common cause of food poisoning by *Salmonella* species was due to *S. typhimurium*. As its name suggests, it causes a typhoid-like disease in mice. In humans, *S. typhimurium* does not cause as severe disease as *S. typhi*, and is not normally fatal. The disease is characterised by diarrhoea, abdominal cramps, vomiting and nausea, and generally lasts up to seven days.

- ***Salmonella enterica*** serovar ***enteritidis*** (also called *Salmonella enteritidis* or abbreviated to *S. enteritidis*). In the last 20 years or so, *S. enteritidis* has become the single most common cause of food poisoning in the United States (phage type 8) and Europe (phage type 4) [16]. *S. enteritidis* causes a disease almost identical to the very closely related *S. typhimurium*. *S. enteritidis* is particularly adept at infecting chicken flocks without causing visible disease, and spreading from hen to hen rapidly, hence the

interest in this particular serovar for this study. When tens or hundreds of thousands of chickens live together, and are slaughtered and processed together, a *Salmonella* infection can rapidly spread throughout the whole food chain. A compounding factor is that chickens from a single farm may be distributed over many cities and even countries, and hence *Salmonella* infections can be rapidly dispersed through millions of people. Thus, the possibility of treating hatching eggs at source to eradicate surface borne bacteria in a clean and safe way is of great interest. *S. enteritidis* has become the predominant cause of salmonellosis in recent years, overtaking *S. typhimurium* as the main causative agent [18].

Salmonella bacteria have been known to cause illness for over 100 years. Salmonellosis, or a *Salmonella* infection, is the illness that can occur if live *Salmonella* bacteria enter the body, usually through eating foods containing the bacteria. Salmonellosis is one of the most common bacterial food-borne illnesses. Two types, *Salmonella enteritidis* and *Salmonella typhimurium,* account for approximately half of all human infections. Strains that cause no symptoms in animals can make people sick, and vice versa. If present in food, it does not affect the taste, smell, or appearance of the food. The bacteria live in the intestinal tracts of infected animals and humans.

After *Salmonella* is ingested, it passes through the stomach to the intestine. Here, it binds to the wall of the intestine, and through special proteins that it generates in response to the particular conditions in the intestine it penetrates the intestinal wall. From here it can find its way to the liver or spleen. For most other bacteria, this journey would kill them, however *Salmonella* has evolved mechanisms to prevent the immune system from doing its job efficiently. In the liver, *Salmonella* can grow again, and be released back into the intestine.

Of course, not all of the *Salmonella* pass through the intestinal wall, and many of them are expelled from the intestine in the diarrhoea. In regions with poor sanitation, these bacteria can than survive in the soil or in rivers and infect the next person, cow, chicken or mouse that comes along.

A person infected with the *Salmonella enteritidis* bacterium usually has fever, abdominal cramps, diarrhoea, nausea, vomiting, and headache beginning 6 hours to 48 hours after consuming a contaminated food or beverage. The illness usually lasts 2 days to 6 days, depending on host factors, ingested dose, and strain characteristics. Most people recover without antibiotic treatment. However, the diarrhoea can be severe, and the person may be ill enough to require hospitalisation. Chronic consequences may include arthritic symptoms 3 weeks to 4 weeks after the onset of acute symptoms. The infective dose may be as few as 15 to 20 bacterial cells; dependant upon age and health of the host, and strain differences among the members of the genus.

The elderly, infants, and those with impaired immune systems may have a more severe illness. In these patients, the infection may spread from the intestines to the blood stream, and then to other body sites and can cause death unless the person is treated promptly with antibiotics.

2.2.2 Viruses

The origin of the word virus is derived from Latin, which literally translates to poison. A definition of a virus suggested by Lwoff is *"Viruses are infectious, potentially pathogenic nucleoprotein entities, with only one type of nucleic acid, which reproduce from their genetic material, are unable to grow and divide, and are devoid of enzymes"* [11].

While viruses have undoubtedly been around for thousands of years, the first virus to be studied was the *mosaic* virus affecting tobacco plants. This particular virus, known as *tobacco mosaic virus* (TMV) was studied carefully in the late 1800s when in 1892, Iwanowski came to the considered conclusion that the virus was non-corpuscular - i.e. that it was not cellular and was much smaller than a cellular organism [13]. The actual size of this virus is 300 nm by 15 nm by 22 nm [11]. This was also the first successfully isolated virus, achieved by W. M. Stanley in 1935 [11].

Table 2.5 outlines the comparative sizes and morphologies of some common viruses [11].

Table 2.5: Comparative sizes and morphologies of common viruses.

Virus	**Morphology**	**Size (nm)**
Foot and mouth disease virus	Spheroidal	8 to 12
Tobacco necrosis virus	Spheroidal	17
Poliomyelitis virus	Spheroidal	28
Tobacco mosaic virus	Rod	300 × 15
Influenza virus	Spheroidal	80
Tomato bushy stunt virus	Spheroidal	30
Turnip yellow mosaic virus	Spheroidal	22
Rabbit papilloma virus	Spheroidal	45
Cowpox virus (smallpox vaccine)	Brick shaped	28 × 22
Chicken pox and shingles virus	Brick shaped	290 × 230
Escherichia coli phage	Head Tail	95 × 65 100 × 25

Virology, the study of viruses, has progressed rapidly in recent years, particularly with the invention of the electron microscope and negative staining techniques. These have enabled virologists to study the sub-microscopic viruses from a new viewpoint, as the most common factor of all viruses is their small size (of the order of tens of nanometres), beyond the capabilities of optical microscopes, as can be seen from Table 2.5.

Viruses on their own cannot multiply, but rely entirely on the metabolic processes of a host cell. The host cell is also the means by which the virus can be introduced into other organisms. Viruses are *obligate intracellular parasites*, a fact, which explains why no virus has ever been cultivated on a cell-free medium.

The chemical constitution of viruses can contain just a nucleic acid core and a protein coat, or capsid. The nucleic acid may take the form of either RNA (ribonucleic acid) or DNA, and can form from 1 % to 40 % of the virus [11]. In most viruses, the nucleic acid is the infective agent with the protein forming a protective covering for the acid. Many viruses can behave as chemicals and be crystallised. Viruses themselves do not perform metabolism, they are replicated inside the host cell by synthesis of their component parts, followed by their assembly into the full virus particle.

The replication mode of viruses is similar for all viruses, with the virus particle, or *virion*, entering a cell and its nucleic acid taking over control of the cellular metabolic processes. This in turn codes for the separate synthesis of the viral nucleic acid and protein, which are later combined into a new virus particle. The nucleic acid is manufactured in the nucleus of the host cell and the proteins in the host cell's cytoplasm. The virus yield from a cell varies, but is typically from 10 to 100 particles [11].

Some virus particles have a very limited survival outside of the normal host cell, for example only a few hours. Others however, are extremely stable and can withstand heat, cold and drought that would kill many vegetative bacteria. Viruses are resistant to antibiotics and many disinfectants, but can be destroyed by oxidising agents such as hydrogen peroxide and hypochlorites.

To infect higher organisms a virus needs a *vector* to help transmit it from the host organism to the new organism, as they themselves are not motile. Such vectors include insects, as in mosquitos transmitting viral infected blood from one animal to another. Other viruses can be transmitted by physical vectors, such as spray droplets from a sneeze carrying the *influenza* virus.

2.2.3 Fungi

Fungi can vary greatly in complexity and size and include mushrooms, yeasts and moulds. However, this study is only interested in the smaller micro-organisms that are fungi.

Fungi are a member of the higher protists and as such have a nucleus that is surrounded by a membrane, compared to the bacteria that do not. A few very simple fungi are unicellular, such as yeasts. Most however, form long slender filaments called *hyphae*, which usually branch freely forming a flocculent mass named the *mycelium*. Most fungi are non-motile, but a few can produce motile cells.

Fungi occupy a wide variety of habitats. Some are aquatic, but the vast majorities occupy moist conditions on land. A few species however can withstand much dryer conditions, for example, *Aspergillus* and *Penecillium*. Fungi are either *saprophytic*, i.e. feeding on non-living organic matter, or *parasitic*, feeding on living organic matter.

Vegetative cells and spores of nearly all fungi are surrounded by a definite cell wall. In most fungi, these cell walls consist of 80 % to 90 % polysaccharides (sugars) together with proteins and other compounds [12].

The common propagative units of fungi are single celled spores. These may be produced sexually or asexually. These spores are much less resistant to attack than the equivalent bacterial spores. Some fungal spores may only be able to survive in normal conditions for only a few hours if they have not found a suitable host [12].

Some fungi such as *Oomycetes* can have flagella and are therefore motile. These flagella are larger and more complex than those of the bacteria. Each flagellum is made of 11 distinct parallel filaments, two of which are centrally placed, the remainder being peripheral. Fungi typically have two forms of flagella, *whiplash* and *tinsel*. Whiplash have 11 strands in which the central pair extend to form the end of the lash. The tinsel flagella are covered with a very fine fur of short slender threads [12].

2.3 Current Sterilisation Methods

There are many varied approaches to sterilisation dependent on the industry in which the application is being used and the degree of sterilisation that is required. For example, surgical instruments in hospitals have to be extremely clean, but a relatively low throughput of items is required. However, the commercial food packing industry requires an extremely high throughput, but the sterilisation requirements are not as stringent as those of medical facilities.

For the purposes of sterilisation of surgical instruments, the absolute level of sterilisation needs to be very high and reproducible. The desired sterilisation is defined as the Sterility Assurance Level (SAL), where the SAL is the probability of a viable micro-organism surviving on a given substrate. In general, a SAL of 10^{-6} is required for aseptic packaging materials and single use disposable medical devices. That is, there is a probability of not more than one viable micro-organism surviving in one million units [19].

Micro-organisms can only survive within certain environmental conditions such as pH, temperature and osmotic pressure. Outside of the optimal conditions the bacteria may survive, but may not be able to grow or reproduce. Increasing the limits further from the optimal will eventually result in death of the bacteria.

Before describing the various methods of sterilisation, some form of measurement of the effectiveness of these actions is required. Various measures of the effectiveness of lethal agents are used, with three common ones being:

- **LD_{50}** the dose needed to kill 50 % of the individual cells of a culture [13].

- The ***log*** method expresses the relative proportion of bacterial kill in a logarithmic fashion. For example, if 90 % of a bacterial culture is killed with 10 % remaining, this can also be expressed in relative terms as a kill rate of 0.9 of the bacterial culture with 0.1 remaining. Taking the base 10 logarithm of the remaining bacterial level (0.1) and ignoring the sign of the result will give the log kill. In this example, $\log_{10}(0.1)$ is -1 giving a 1 log kill. Similarly, a kill rate of 99.9 %, can be expressed in terms as a 3 log kill. In this method each increment in the log kill represents an order of magnitude (or decimal) reduction of the bacterial culture.

- Following on from the log method, the **D value** is the time taken to achieve a 90 % (1.0 log) reduction in the number of viable cells in a culture. This value gives an indication of the relative efficiency of a given sterilisation dose. The lower the D value, the more effective the sterilisation dose [16].

The following list details some current sterilisation methods. Each item will be subsequently dealt with in detail in the following sections of this chapter.

- Temperature
  - Dry Heat
  - Moist Heat (Autoclaving)
  - Cold
- Desiccation
- Cellular Disintegration
- Chemical Disinfectants
- Light
  - White Light
  - Ultra-violet Light
  - Laser Light
- Plasma
- Electric Fields
- Ionising Radiation

2.3.1 Heat

Heat is a widely used and very effective sterilising agent for micro-organisms. Heat will not leave any contamination and can be used wherever the heat will not have any adverse effect on the item being sterilised.

The effect of heat in sterilisation is directly related to the temperature and the duration of its application [13]. Figure 2.5 shows the effect of dry heat temperatures on *Escherichia coli* for different treatment times. A convenient term for this analysis is called the *"thermal death time"*. This is the time required at a given temperatures to kill a given culture, and will vary from bacteria to bacteria.

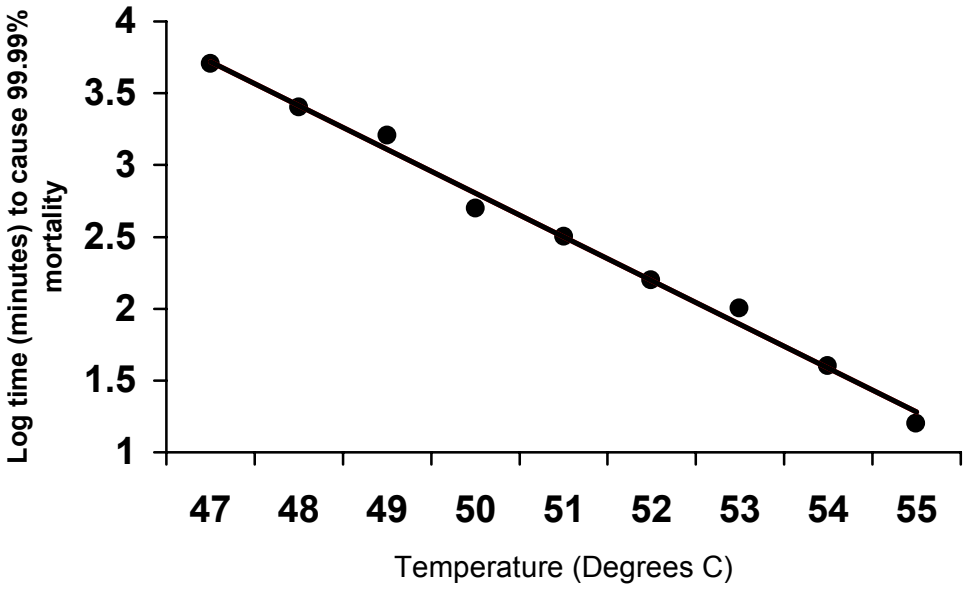

Fig. 2.5: Mortality effect of different temperatures on *E. coli* at pH 7.

2.3.2 Dry Heat

Dry heat is not the normal preferred method of sterilisation by heat, due to the increased length of exposure time compared to moist heat. It is however sometimes necessary when items to be sterilised cannot be subjected to moisture.

With dry heat sterilisation methodologies, the heat is transferred to the micro-organisms via convection, radiation and conduction. A continuous dry heat of 160 °C for 60 minutes is the considered requirement for efficacious sterilisation. This will kill all bacteria plus the more resistant bacterial spores. A lower temperature of 100 °C for 60 minutes will kill non-sporing bacteria but not the sporing varieties.

2.3.3 Moist Heat (Autoclaving)

Autoclaving is the common method for sterilisation of surgical instruments used in the medical industry.

Autoclaving uses moist heat to perform sterilisation. Moist heat is more efficient than dry heat, sterilising at lower temperatures and shorter durations. Table 2.6 shows a comparison of the sterilisation times and temperatures between moist heat and dry heat.

Moist heat is more efficient than dry heat since steam will provide rapid heat transfer to items being sterilised. On condensing on a cold object, the steam will give up its latent heat of vaporisation, but the partial vacuum created by this process will also contribute to sucking more steam to the object.

Table 2.6: Equivalent minimum sterilising temperatures in moist / dry heat.

Moist Heat			**Dry Heat**	
Pressure † (PSI)	Temperature (°C)	Sterilising time (minutes)	Temperature (°C)	Sterilising time (minutes)
0	100	1200	-	-
5	109	150	-	-
10	115	50	-	-
15	121	15	120	480
20	126	10	-	-
30	134	3	140	150
-	-	-	160	60
-	-	-	170	25
-	-	-	180	10

† Pure steam.

The relationship between the temperature of steam obtained and the system pressure is directly related to the vapour pressure of water. The boiling point of a liquid is obtained when the liquid's vapour pressure is equal to the local atmospheric pressure. Thus, when the local atmospheric pressure is increased, the water requires heating to a greater temperature to raise its vapour pressure until boiling point is reached and steam is produced [20].

A typical autoclave system will generate an internal temperature of 121 °C under a pressure of 15 psi. The heat generated is not a dry heat, but via heated steam generated from distilled water fed into the system prior to operation. These conditions are maintained for approximately 30 minutes, after which the items contained within the autoclave can be considered sterilised. The minimum quoted for these conditions are 15 minutes, but 30 minutes is considered the best practical time to include a reasonable safety margin.

2.3.4 Cold

Cooling bacterial cultures will kill a proportion of the cells, depending on the cooling procedure used. Some moulds and yeasts are more resistant to freezing than most bacteria, although bacterial spores are almost unaffected due to their virtual absence of free water.

The predominant process for denaturation by freezing is the formation of ice crystals within the cells causing mechanical damage, or the precipitation of coagulable cellular proteins. This latter effect is more prevalent at -2 °C than at lower temperatures. Hence, the death rate of cells is found to be greatest at temperatures just below freezing.

The rate at which cooling occurs is also important. For example, a cold shock caused by rapidly cooling *Escherichia coli* cultures from 37 °C to 4 °C produced a 95 % reduction in viable cells, whereas the same temperature drop produced over 30 minutes showed no loss of viability [13].

2.3.5 Desiccation

When drying, some loss of viability occurs in all micro-organisms, with bacterial spores being the least effected due to their lack of free water. A water content reduction to 30 % to 40 % is considered the most harmful to cells [13].

When dried and left, the period of survival of bacterial cells can range from a few hours for the most susceptible to months for the more resilient.

2.3.6 Cellular Disintegration

Micro-organisms can be disintegrated by ultrasonic or mechanical agitation.

Ultrasound, at frequencies of the order of 700 kHz, is only effective on micro-organisms in liquid suspension [13]. Apart from the minimal heating effects of the sound waves, the main bactericidal function of ultrasound is almost entirely attributed to the physical destruction of the cells. The effect seems more related to the intensity of the incident sound waves rather than their frequency.

Gram-negative bacteria are generally more resistant to this type of effect (due to their stronger cell walls), as are bacterial spores (due to their virtual absence of free water).

In addition to cellular disintegration via ultrasound, this can also be achieved by means of mechanical agitation of the bacteria in the presence of abrasives (e.g. carborundum), or alternatively, shearing forces brought about by forcing a liquid or frozen suspension through a narrow opening.

Mechanical methods of cellular disintegration do not have the same associated heating effects as with ultrasound methods.

2.3.7 Chemical Disinfectants

Chemical disinfectants can come in the form of both gases and liquids and are many and varied. Some of the major types of chemical disinfectants in use are:

- Halogens
- Alkylating Reagents
- Phenolic Compounds
- Aldehydes
- Alcohols
- Acids and Alkalis
- Heavy Metals and Their Salts

Each of the above topics will now be briefly discussed:

2.3.7.1 Halogens

Halogens or compounds that produce them are highly bactericidal via the action of oxidation of proteins and similar substances within the bacterial cells.

As a direct result of their method of action, prolonged action on bacterial cultures will dilute their effectiveness and hence limit their usefulness. Bleaches are examples of halogen disinfectants.

2.3.7.2 Alkylating Reagents

Such compounds as ethylene oxide are highly effective bactericidal agents through their ability to alkylate bacterial structures. The gas ethylene oxide will effectively kill bacterial spores and is in common use as a sterilising agent.

To avoid the risk of explosion, this gas is often mixed with an inert gas under pressure at a standard humidity and temperature. This method is widely used in industry for sterilisation because of the extremely penetrating properties of ethylene oxide.

2.3.7.3 Pheonolic Compounds

These reagents are extremely toxic by virtue of their action of protein denaturation. Proteins are precipitated by only 1 % to 2 % phenol. However, none of these chemicals are capable of killing bacterial spores. Most vegetative bacteria for example are killed by 1 % phenol in 5 minutes to 10 minutes at 20 °C, but anthrax spores survive 24 hours in 5 % phenol [13].

Halogenated phenols are much less toxic than phenols and suffer the same reduction of effectiveness as other halogens agents. This group of substances includes "Dettol", which is a chloroxylenol.

2.3.7.4 Aldehydes

Formaldehyde, often sold as a 40 % aqueous solution known as formalin, is able to kill bacterial spores even in concentrations as low as 1 %. Hence, this can achieve complete sterilisation, but will also remain toxic to other organisms for long periods.

In the form of a gas or solution, it is slow to penetrate into the bacterial cells, but is highly efficient.

Other aldehydes utilised for sterilisation include 2 % aqueous solution of glutaraldehyde, which is less irritant than formalin, but more rapidly bactericidal.

2.3.7.5 Alcohols

Alcohols kill vegetative bacteria very rapidly but have no action on bacterial spores. This is because their action requires the presence of water. Pure alcohol is less effective than a more dilute concentration, with 70 % being optimal. Isopropyl alcohol has been found to be slightly more effective than ethyl alcohol.

2.3.7.6 Acids and Alkalis

Mineral acids and alkalis produce their main action on bacteria via their hydrogen and hydroxyl ions respectively. Hydrogen ions have been found to be more effective than hydroxyl ions.

2.3.7.7 Heavy Metals and Their Salts

All of the heavy metals are bactericidal and fungicidal to some degree. Silver and copper exhibit these properties at minute concentrations, a property referred to as *oligodynamic activity*.

The salts and organic complexes of mercury, tin, silver and to a lesser degree copper are all bactericidal. When ionised in an aqueous solution the metal ions combine with and precipitate cell proteins.

2.3.8 White Light

Light will only have an effect on a bacterial cell if it is absorbed. If the light passes straight through the cell, then it will have no effect. The absorption of light can promote chemical changes within a bacterium and hence cause biological damage.

Light in the visible wavelength regions of 400 nm to 750 nm is absorbed by relatively few compounds found in bacteria, and will hence have little bactericidal effect. This is also true of ultra-violet light in the 300 nm to 400 nm region, below that however, effects are noted. These will be described more fully in the next section on ultra-violet light.

Visible light can be utilised to sterilise micro-organisms, but only with the addition of a photo-sensitising dye such as erythrosin [13]. Such dyes are said to posses photodynamic action. The addition of such dyes allows the visible light to promote the creation of cytotoxins, which subsequently kill the cell.

White light sterilisation systems have been proven to operate as efficient bactericidal systems, however, most of the bactericidal effect is found from the ultra-violet light within the broad spectrum of white light. The ultra-violet light can account for up to 25 % of the luminous energy emitted from an inert xenon gas flash lamp.

An embodiment of a flash lamp, white light sterilisation system has been demonstrated by PurePulse Technologies Incorporated, San Diego [21]. This system uses 200 μs to 300 μs pulses of white light from low-pressure xenon flash lamps at rates of 1 Hz to 20 Hz. The spectra of light emitted from these lamps ranges from the far ultra-violet (200 nm) to the infrared (1 100 nm), with the typical distributions being 25 % ultra-violet, 45 % visible and 30 % infrared. All of these radiations are non-ionising.

While the peak powers of the individual pulses are extremely high because of their short duration, the total energy in every pulse is relatively low. The energy density provided on the products being sterilised from these systems is in the range of 1.5 $J \cdot cm^{-2}$ to 4 $J \cdot cm^{-2}$ [22]. These energy densities are typically 20 000 times more intense than the natural radiation energy density of the sun on the earth [21]. Due to the short duration of these pulses, there is minimal heating effect on any items being sterilised.

The kill rate of the above system has been shown to achieve 9 log per colony-forming units (CFUs) per cm^2 on vegetative organisms and 6 log $CFUs \cdot cm^{-2}$ on bacterial spores (both on smooth surfaces) [21]. Reduced rates of about 2 log $CFUs \cdot cm^{-2}$ to 3 log $CFUs \cdot cm^{-2}$ are typical on porous surfaces, where certain areas of the article being sterilised may be in shadow from the incident light. The system has proved efficacious in the treatment of vegetative bacteria, yeasts, moulds, bacterial spores and viruses.

2.3.9 Ultra-violet Light

Ultra-violet radiation of wavelength less than 300 nm is strongly absorbed by proteins and nucleic acids. Relatively small doses of this form of radiation can bring about chemical changes in these compounds causing chromosome damage, genetic mutation or death. Higher dose levels are required to inactivate enzymes however [13].

In some micro-organisms, the harmful consequences of exposure to ultra-violet light can be partly averted by subsequently exposing them to visible light. This process is known as *photoreactivation*. An example of a bacterium containing a photoreactivating enzyme is *Escherichia coli*.

There exists a maximum absorption of ultra-violet radiation in DNA at 260 nm, thus resulting in maximum DNA damage when irradiated with this specific wavelength of ultra-violet light [23]. Additionally, polychromatic ultra-violet light can be more a more efficient bactericide than monochromatic light due to the photoreactivation mechanisms of certain bacteria.

Ultra-violet light has very poor penetration into cells, but is still an effective sterilising agent. Germicidal ultra-violet lamps can be used for low levels of sterilisation. One such example is a quartz-mercury vapour lamp. Typical examples of these germicidal lamps operate in the UV-C band (200 nm to 280 nm) at wavelengths of 253.7 nm (very close to the maximum absorption of ultra-violet radiation in DNA at 260 nm). The powers of these lamps are typically in the tens of Watts. Such a lamp can achieve up to 4.5 log reduction in CFUs in 10 s on *Aspergillus niger* spores [21].

Typical doses of these lamps range from 500 $W \cdot s^{-1} \cdot cm^{-2}$ to 150 000 $W \cdot s^{-1} \cdot cm^{-2}$ to achieve a 90 % (1 log) kill rate. An example for the *Escherichia coli* bacteria in air is 690 $W \cdot s^{-1} \cdot cm^{-2}$ and in water 5 400 $W \cdot s^{-1} \cdot cm^{-2}$. The high absorbency of the ultra-violet light in water can be clearly seen from these figures. This is the same reason for the poor penetration into cells, as the cells are typically 80 % water.

The following section on laser light also contains lasers that generate ultra-violet light at fixed frequencies (monochromatic). The dose levels found with laser system however, can be significantly higher that those generated by conventional lamps.

2.3.10 Laser Light

The use of laser light to obtain bactericidal effects will be explained in detail in the next chapter. This section will give a brief outline of the potential uses of laser light for this goal.

Both white light and ultra-violet light have been discussed with a view to promoting bactericidal effects. Laser systems offer a mechanism for providing exceptionally high energy densities of monochromatic wavelengths of light. Moreover, these high energy densities can be readily directed to various target sites.

Laser wavelengths cover the entire visible spectrum and extend into the ultra-violet and infrared portions of the electromagnetic spectrum. Previous methods discussed with ultra-violet light can also be accomplished with lasers. Lasers producing light in the visible spectra can also be used with additional chemicals to help produce cytotoxins. High-powered lasers in the infrared portion of the spectrum produce light that is highly absorbed in water. As cells are 80 % water, this portion of the spectra is extremely useful. Infrared effects will tend to be thermally based compared to the ultra-violet light effects that break the bonds of molecules directly.

Table 2.7 shows a list of lasers that have undergone trials for bactericidal effects.

Table 2.7: Laser types test for bactericidal effectiveness.

Laser	**Wavelength**
Far-Infrared (FIR)	118 μm
Carbon Dioxide	10.6 μm
Er:YAG	2.94 μm
Nd:YAG	1.06 μm
Ruby	694 nm
HeNe	632 nm
Frequency Doubled Nd:YAG	532 nm
Frequency Tripled Nd:YAG	355 nm
Laser Diode Array	810 nm
Argon Ion	488 nm
ArF Excimer	193 nm

Not all of the above lasers have proven efficacious in the sterilisation of bacteria. The Laser diode array, FIR, argon ion, ruby and helium neon (HeNe) lasers do not appear to be very promising from initial research trials [3], [7]. Although with the addition of photosensitising agents, the efficiency of some of these lasers can be increased [24], [25].

Lasers will be covered in detail in the subsequent sections of this thesis.

2.3.11 Plasma

Plasma technologies have been investigated for the past twenty years or more, but it was not until 1995 that a method of plasma sterilisation of medical devices was clearly demonstrated [26].

A plasma in general terms is a gas, in which charged ions exist, and in which some of the atoms, molecules or molecular fragments are electrically charged. Such examples of naturally occurring plasmas are lightning. These plasmas are typified by their characteristic high energy and high temperature discharges.

The use of low temperature plasmas for sterilisation purposes is akin to other more common low temperature plasmas embodied in fluorescent and neon lighting, facilitated by the use of vacuums. The use of low temperatures to create these plasmas will enable a range of products to be sterilised that would otherwise be adversely affected by high temperature sterilisation methods.

A typical plasma sterilisation procedure involves a two-stage process. The first stage requiring the exposure of the materials to be sterilised to a vapour created from a peroxygen compound, for example hydrogen peroxide. The second stage of the process involves exposing the material to be sterilised (and the peroxygen compound) to a low temperature glow discharge gas plasma. This process generates biocides from the peroxygen compound.

The use of peroxygen compounds is specified, as following the sterilisation process the compounds revert to non-toxic by-products, such as water and oxygen in the case of hydrogen peroxide.

The mechanism for bacterial kill is thought to be the oxidation of the cellular membranes.

2.3.12 Electric Fields

The damaging effects of high energy, 10 kV, electric fields on *Escherichia coli* have been recorded as long ago as 1967 [27]. More recent studies have concluded that the voltage potential threshold across a cell's membrane, in order to kill the cell is 1 V [27]. Based on this principle, a commercial system is now available and supplied by PurePulse Technologies Incorporated, called Coolpure™ [22].

This process kills high levels of vegetative micro-organisms through massive electroporation, or rupturing, of the bacterial cell membranes and can kill vegetative bacteria in pumpable liquids, held at temperatures between 25 °C to 60 °C, with virtually no chemical change in the product being sterilised. Being operated at low temperatures, foodstuffs that are normally processed using high temperature pasteurisation, can now be processed at lower temperatures, without significantly affecting the taste of the product compared to high temperature methods [22].

This system uses short high-energy pulses to rupture the bacterial cell walls. Such a system can provide 1 pulse to 20 pulses per second. Each pulse is between 1 μs to 10 μs in duration, with an electric field strength of 20 $kV \cdot cm^{-1}$ to 80 $kV \cdot cm^{-1}$. The treatment parameters can be varied, together with the duration of the liquid product in the system, by varying its flow.

Effective treatment of raw milk has been demonstrated [22] at 55 °C with bacterial kill levels in excess of conventional pasteurisation methods. In tests with *Listeria innocua* as a substitution for *Listeria monocytogenes*, greater than 6 log were killed with only a few seconds exposure at 55 °C.

Extensive chemical analyses by the manufactures of this process have shown that no changes in the chemical or physical properties of treated milk have been observed. This analysis included enzyme activity, fat integrity, starter growth, rennet clotting yield, cheese production, calcium distribution, casein structure and protein integrity.

2.3.13 Ionising Radiation

The effects of ionising radiation have been well documented on cells since Roentgen's discovery of *x*-rays in 1895 while performing experiments on electrical discharges on cathode ray tubes. Roentgen is said to have exposed his hand between the tube and phosphorescent plate and saw a shadow of his bones [28]. Shortly after Roentgen's discovery, Becquerel discovered similar radiations emitted by uranium ores, and Professor and Madame Curie succeeded in isolating the radioactive element radium.

Soon after the discovery of radiation emitted by uranium, Becquerel is said to have "burnt" himself while carrying some uranium in his pocket. Hence the two key properties of these new forms of radiation were soon discovered; the highly penetrating nature of the radiation on biological tissues, and their harmful effects to the aforementioned. It is interesting to note that nearly all of the pioneer radiologists later died from cancer, indicating the harmful, yet delayed effects that radiation can present.

Ionising radiation in the form of hard *x*-radiation, alpha (α) and beta (β) particles and gamma (γ) rays of radioactive elements have sufficient energy to ionise an atom and remove one of its electrons. The emitted electron will have a significant amount of energy, which when in collision with another atom or molecule can in turn ionise them. This is the secondary effect of ionising radiation.

The effect of the ionising radiation can be to change the DNA structure and prevent the cell from reproducing, or alternatively, produce mutations that are unable to survive in the existing habitat of the cell. A typical cell may have 4 000 genes, any of which may become mutated by ionising radiation. The effects of which may vary depending upon the gene affected. Mutated housekeeping genes can be catastrophic for a cell, while genes effecting cell wall protein production may be less significant [29]. Figure 2.6 shows the process by which ionising radiation can damage a cellular organism.

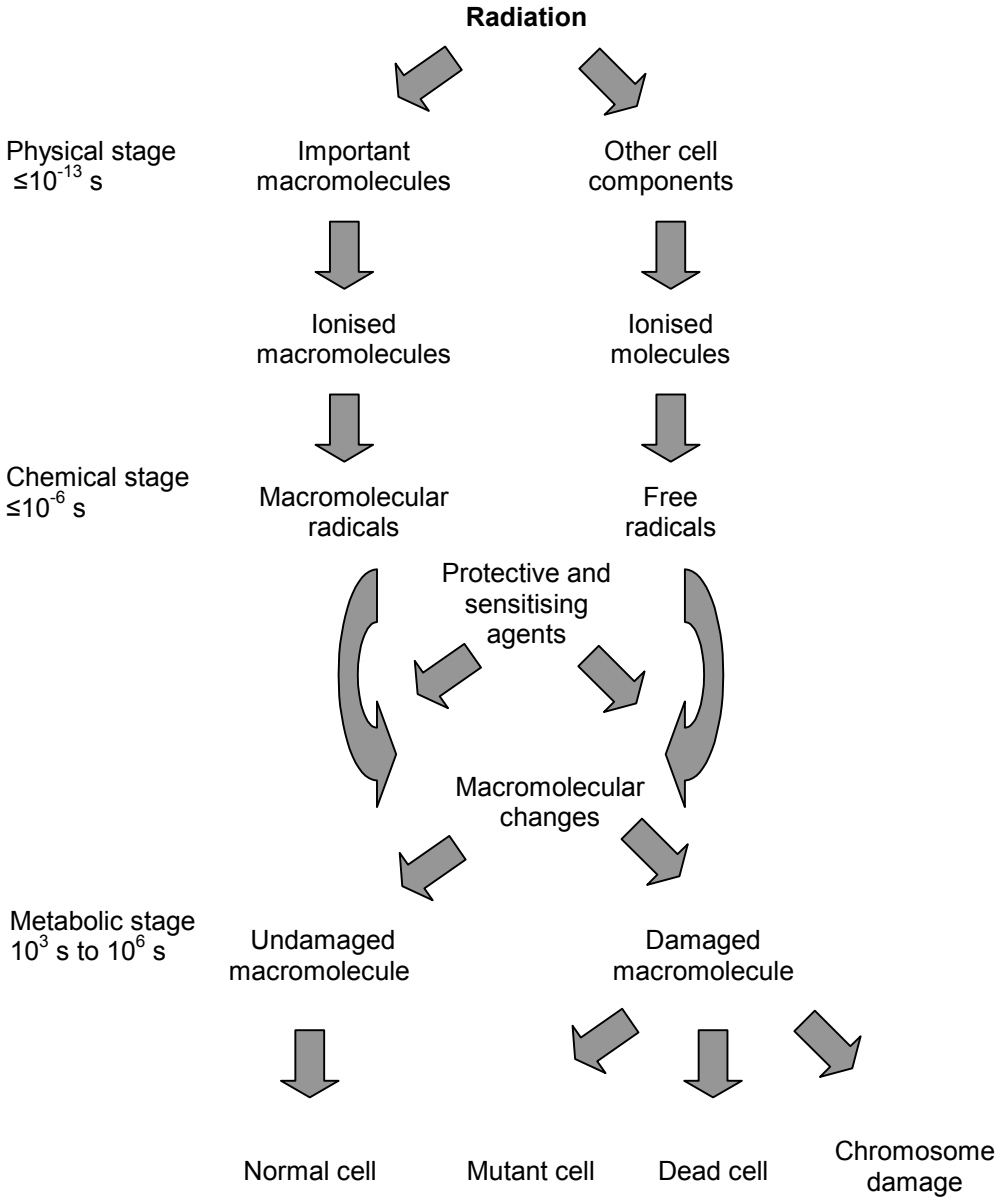

Fig. 2.6: Development of radiation damage in cells.

Ionising radiation can fall into two categories, *corpuscular* (particulate) or *electromagnetic*. Corpuscular radiations (alpha and beta) are streams of atomic or sub-atomic particles moving at high velocities and hence have high kinetic energies. Corpuscular radiations with energies of a few hundred electron volts are all capable of producing ionisations. Electromagnetic radiations in the *x*- and γ-ray wavelengths have sufficient energy to cause ionisations. More generally, electromagnetic radiations that cause ionisations are called *x*-rays if they are machine generated, or γ-rays if they are emitted from radioactive isotopes.

For commercial irradiating equipment, the cobalt 60 (^{60}Co) isotope is the most common source. This isotope has a half-life of 5.27 years. This isotope emits highly penetrating gamma radiation. Sterilisation via this method will leave no chemical residues on the items being sterilised. Doses of up to 25 kGray can be achieved with these types of systems [19].

Table 2.8 gives an outline of some radiation doses from a cobalt 60 source for typical sterilisation applications.

Table 2.8: Radiation doses for typical sterilisation procedures.

Typical Applications Dose	**kGray**
Sterilisation of medical devices	25
Sterilisation of packaging systems	5 to 25
Sterilisation of laboratory supplies	5 to 25
Microbial decontamination	10 to 25
Bioburden reduction of cosmetic's raw materials	2 to 10
Preservation of antiques	2 to 10
Plastics modification (cross-linking and scission)	10++
Bioburden reduction of food ingredients	1 to 10

Note that Table 2.8 includes the time duration by definition of the dose in the Gray unit. The Gray unit's dimension according to the SI system is L^2T^{-2} [30].

2.4 Summary

The above chapter has conducted a review of micro-organisms, with particular attention being focused on bacteria and more specifically *Salmonella*. Many bacteria cause disease in humans and animals. These pathogenic bacteria cause such diseases due to their direct or indirect production of toxins in the host organism. One such example being the bacteria *Salmonella enteritidis* commonly found on and in eggs causing Salmonellosis in humans.

Bacteria are single celled organisms, of the order of size of 1 μm and volume of 2 μm^3. By way of comparison, a human hair has a diameter of approximately 100 μm. Being this small, bacteria cannot be seen with the naked eye, but may be viewed using normal white light microscopy. Bacteria typically weigh about 20×10^{-7} g, 80 % of this weight being water. The small size of bacteria contributes to their high surface area to volume ratio en mass. This helps their metabolic process due to the rapid exchanges of substances across the cell walls, but by contrast also can be used against the bacteria for sterilisation.

Bacteria have an optimum temperature for growth, at which (under ideal conditions) they can reproduce at alarming rates. Above or below this temperature growth rates will reduce and eventually stop. At extremes of temperature, the bacteria will be killed. Gram-negative bacteria such as *Salmonella* are more resilient to physical and chemical extremes than Gram-positive bacteria and are hence harder to kill. Some bacteria can also form endospores, which have little free water and can withstand both chemical and physical extremes. This effect is triggered as a survival mechanism when environmental conditions become too harsh for normal vegetative cells.

Having looked at the physiology of micro-organisms and bacteria in particular there followed a discussion of a host of methods that can promote the sterilisation of these bacteria. These individual methods are summarised below.

Heat: This method is widely used, very effective, leaves no contamination and can be used wherever the heat will have little detrimental effect on the item being sterilised. The sterilisation efficiency is directly related to the temperature and duration of application.

Dry Heat: This requires an increased length of exposure time compared to moist heat. It is sometimes useful when items to be sterilised cannot be subjected to moisture. A continuous dry heat of 160 °C for 60 minutes is the considered effective for sterilisation.

Moist Heat: (Autoclaving) is the common method for sterilisation of surgical instruments and is more efficient than dry heat, sterilising at a lower temperature in shorter durations. Sterilisation is typically achieved in 30 minutes at a temperature of 121 °C under a pressure of 15 psi.

Cold: Cooling has been shown to kill vegetative bacteria, with a faster rate of cooling producing a greater degree of sterilisation. The predominant process for this method is the formation of ice crystals within the cells causing mechanical damage.

Desiccation: A water content reduction to 30 % to 40 % has been shown to be the most effective for reduction of viability in vegetative bacteria.

Cellular Disintegration: Ultrasound at 700 kHz can disintegrate bacteria in liquid suspension, with its efficiency being directly related to the ultrasound's amplitude. Mechanical agitation of bacteria in the presence of abrasives, or shearing forces generated by forcing a liquid through a narrow opening can also disintegrate bacterial cells.

Chemical Disinfectants: Chemical disinfectants used in both gaseous and liquid forms, are extremely effective and in common usage. Some popular examples being bleaches (halogen disinfectants), ethylene oxide gas, formaldehyde and isopropyl alcohol. All of these chemicals act in slightly different ways and will hence be used for different bactericidal purposes.

White Light: This can sterilise bacteria, however, most of the effect is due to ultra-violet light accounting for 25 % of its spectrum [21]. Such systems use xenon flash lamps pulsed at up to 20 Hz with pulse durations of 200 μs and energy densities of around 4 $J \cdot cm^{-2}$ [22]. The short durations of these pulses cause minimal heating of the items being sterilised. Kill rates of 9 log $CFUs \cdot cm^{-2}$ on vegetative organisms and 6 log $CFUs \cdot cm^{-2}$ on bacterial spores have been achieved.

Ultra-violet Light: Ultra-violet light of wavelength 260 nm is strongly absorbed in DNA [23], with relatively small doses causing sterilisation. quartz-mercury vapour lamps are used for low levels of sterilisation, operating in the UV-C band at wavelengths of 253.7 nm The powers of these lamps are typically in the tens of Watts with doses of 500 $W \cdot s^{-1} \cdot cm^{-2}$ to 150 000 $W \cdot s^{-1} \cdot cm^{-2}$ achieving a 1 log kill rate.

Laser Light: Lasers produce high energy densities of monochromatic light from the ultra-violet to infrared. Previous methods discussed with ultra-violet light can also be accomplished with lasers. High-powered infrared lasers produce light that is highly absorbed in water, where such effects are thermally originated.

Plasma: This is a two-stage hybrid process using a low temperature plasma and peroxygen compound (hydrogen peroxide), which following the sterilisation process reverts to non-toxic by-products, such as water and oxygen [26].

Electric Fields: These kill vegetative bacteria through the rupturing of the cell membranes, with virtually no chemical change in the product being sterilised. Such systems can operate at 20 Hz, with pulse durations of 10 μs and electric field strengths of 80 $kV \cdot cm^{-1}$. Treatment of *Listeria* at 55 °C has produced in excess of 6 log kill rates with only a few seconds exposure [22].

Ionising Radiation: This changes the cell's DNA structure and prevents reproduction, or causes death. Commercial systems commonly use the cobalt 60 isotope, which emits highly penetrating gamma radiation [19]. Sterilisation via this method leaves no chemical residue on the items being sterilised. Doses of up to 25 kGray can be achieved with these types of systems.

Of the forms of sterilisation discussed, the heat and chemical disinfectant methods are by far the most popular, with each method having its specific applications suiting its method of biocidal action and side effects. Other methods such as ultra-violet light and ionising radiation have their own niche applications but are less common than the aforementioned methods. Of the remaining methods, some are still in their commercial infancy (electric fields for example), while others have the potential, but have yet to achieve commercial realisation – laser light for example.

3 : LASER PARAMETER SELECTION

3.1 Introduction

The preceding chapter has discussed the basic cell physiology of bacteria, together with a brief discussion of viruses and fungi. This has provided an insight into the mechanisms for bacterial growth and reproduction, and the external influences that can be brought upon them with an aim to reduce their viability.

The second part of the preceding chapter outlined a cross section of current techniques for achieving these aims, using a broad range of methods and media. The use of light (in particular laser light) was also briefly mentioned, this is now expanded upon in this chapter.

This chapter investigates the use of lasers to promote the sterilisation of a host of micro-organisms with a view to implementation on a commercial scale. The first section of this chapter will look into the electromagnetic spectrum with the aim of choosing the optimum wavelength for the treatment of bacteria, viruses and fungi. Each wavelength will be analysed for its relative merits and practicality on a commercial scale with a typical laser system required to produce the desired wavelength.

In addition to a suitable wavelength for the sterilisation of micro-organisms, other laser parameters can come under direct control that will help influence the efficiency of the sterilisation method. These will be the optimisation of the laser's temporal and spatial profiles.

The temporal profiles are more important for pulsed laser systems, but are also of interest if continuous wave lasers are used with scanning technology to cover large surface areas requiring treatment.

Spatial profiles refer to the quality of the laser beam. That is, throughout a given laser spot, the energy may vary from point to point within the spot. These spatial profiles are commonly called laser modes.

3.2 Wavelength Selection

Lasers produce monochromatic light at selected wavelengths. The aim of this section is to select an optimal wavelength for the sterilisation of micro-organisms via laser, with the chosen laser being as practical as possible for commercial applications.

Before looking at individual lasers, it is first wise to look at the electromagnetic spectrum as a whole, with an analysis of different wavelength's properties with relation to the sterilisation of micro-organisms. An investigation into the generation of the specific wavelengths by different laser types will follow this analysis.

3.2.1 Electromagnetic Spectrum

All waves within the electromagnetic spectrum have certain features in common; one such is the relationship between their wavelength and frequency being a constant, which is the speed of light in a vacuum [31]. This can be seen in equation 3.1:

$$c_0 = \nu\lambda = 2.997\,924\,58 \times 10^8 \text{ m}\cdot\text{s}^{-1} \textit{ in vacuo} \qquad \textit{(Equation 3.1)}$$

Where: ν is the frequency of the wave
λ is the wavelength of the wave
c_0 is the speed of light *in vacuo* (constant)

Figure 3.1 shows the broad electromagnetic spectrum, covering the ranges from the ionising radiation of γ- and *x*-rays through the light waves (ultra-violet, visible and infrared) to microwaves and radiowaves [32].

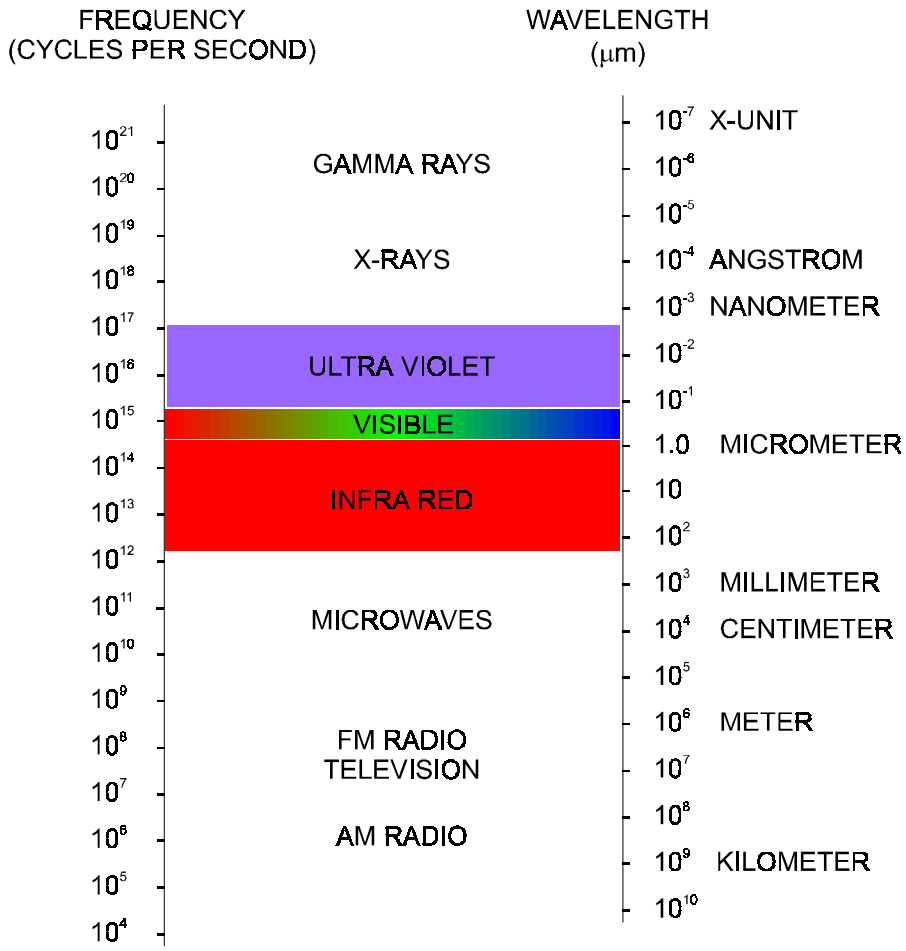

Fig. 3.1: Electromagnetic spectrum from radiowaves to *γ*-rays.

The previous chapter has touched on the use of *x*- and γ -rays (ionising radiation) as potential sources for sterilisation methods. Microwaves and radiowaves have not been discussed previously, and are beyond the scope of this thesis, although they can potentially be used for sterilisation, as in the use of microwaves to boil water and cook food. Hence, if microwaves were used on living organisms (which have a high percentage of water), sterilisation can be precipitated. The frequency of microwaves most commonly used in domestic microwave ovens 2 450 MHz [27].

Figure 3.2 expands upon the light portion of the electromagnetic spectrum from ultra-violet to infrared light, which forms the basis of the continuing discussion [32]. The bottom of figure 3.2 shows selected wavelengths for each section of the electromagnetic spectrum in the light region.

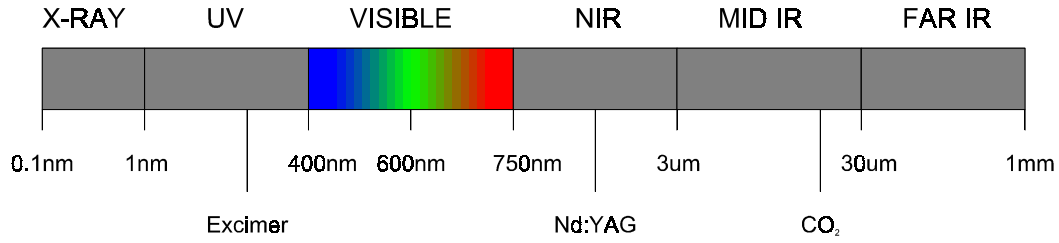

Fig. 3.2: Electromagnetic spectrum for light wavelengths.

As seen in the lower portion of figure 3.2, almost every portion of the light spectrum is covered by a commercially available laser system. Table 3.1 shows a brief list of some commercially available laser systems ranging from the infrared portion of the spectrum to the ultra-violet [32], [23]. Against each laser is shown its wavelength, the portion of the spectrum in which the light wavelength falls, the photon energy of the particular wavelength and the typical effect of the photon energy on biological organisms.

3.2.2 Photon Energies

The photon energies for a corresponding light wavelength can be calculated from the following equation:

$$E = h\nu \qquad \textit{(Equation 3.2)}$$

Where: ν is the frequency of the light wave

E is the photon energy of the light wave

h is plank's constant = $4.135\ 669\ 2 \times 10^{-15}$ eV·s

Table 3.1: Sample selection of commonly available laser types.

Laser	Wavelength	Colour	Photon energy	Interaction
CO_2	10.6 μm	Infrared	0.12 eV	Photo-thermal
Er:YAG	2.94 μm	Infrared	0.42 eV	Photo-thermal
Ho:YLF	2.06 μm	Infrared	0.60 eV	Photo-thermal
Nd:YAG	1.06 μm	Infrared	1.17 eV	Photo-thermal
Diode	810 nm	Infrared	1.53 eV	Photo-thermal
Ruby	694 nm	Deep red	1.79 eV	Photo-thermal
HeNe	632 nm	Red	1.96 eV	Photo-thermal
Dye	585 nm	Yellow	2.12 eV	Photo-thermal
Cu vapour	578 nm	Yellow	2.15 eV	Photo-thermal
Nd:YAG (2^{nd})	532 nm	Green	2.33 eV	Photo-thermal
Argon Ion	488 nm	Blue	2.54 eV	Photo-thermal
XeF Excimer	351 nm	Ultra-violet	3.53 eV	Photo-chemical
XeCl Excimer	308 nm	Ultra-violet	4.03 eV	Photo-chemical
KrF Excimer	248 nm	Ultra-violet	5.00 eV	Photo-chemical
KrCl Excimer	222 nm	Ultra-violet	5.58 eV	Photo-chemical
ArF Excimer	193 nm	Ultra-violet	6.42 eV	Photo-chemical
F_2 Excimer	152 nm	Ultra-violet	7.90 eV	Photo-chemical
H_2 Excimer	110 nm to 162 nm	Ultra-violet	11.27 eV to 7.65 eV	Photo-chemical

Equation 3.1 can be rearranged for ν (frequency) and substituted into equation 3.2 giving equation 3.3 below. This can then be solved, giving photon energies for known wavelengths of light, with the relative values substituted (equation 3.4):

$$E = \frac{hc_0}{\lambda} \qquad \textit{(Equation 3.3)}$$

$$E = \frac{1.239\,842\,435 \times 10^{-6}}{\lambda} \qquad \textit{(Equation 3.4)}$$

Where: λ is the wavelength of light in metres

E is the photon energy in eV

Table 3.1 showed corresponding photon energies for each specific laser wavelength and the expected biological effect. These effects can be calculated from the relevant molecular bond energies found between certain atoms in biological molecules. Table 3.2 lists a range of typical bonds found in biological organisms in proteins, and their bond energies [20].

Table 3.2: Common biological molecular bond energies.

Molecule	Bond Energy (eV)
C-N	3.0
C-C	3.6
N-H	4.0
C-H	4.3
H-H	4.5
O-H	4.8
C=C	6.4

From Table 3.2, it can be seen that certain energies are required to break specific chemical bonds. The lowest bond energy in Table 3.2 is 3.0 eV for the carbon-nitrogen bond. Referring back to Table 3.1, it can be seen that the first laser to generate a short wavelength sufficient to break this bond is the XeF excimer laser with a wavelength of 351 nm and a photon energy of 3.53 eV. This laser is in the ultra-violet region of the spectrum.

Photon energies between 0.01 eV to 1 eV (generated from the infrared region of the electromagnetic spectrum) cause increased rotational and vibrational activity in molecules, which is manifested by the increase in heat in the compound being irradiated [33]. Absorbed energies between 1 eV and 3 eV (in the visible and near ultra-violet regions) will result in changes to the energies of the valence electrons in molecules.

Energies between about 3 eV and 6 eV cause electronic excitation and molecular dissociation in atoms and molecules (as in Table 3.2). Energies above about 6 eV are those associated with ionising radiations such as *x*-rays. These energies are typically above the normal bond energies found in common biological molecules.

3.2.3 Principal Radiation Effects

From the above, it can be seen that there are two predominant methods for the precipitation of sterilisation via lasers; either photo-thermal or photo-chemical.

- **Photo-thermal** effects lie within lasers whose photon energies are not sufficient to break the chemical bonds in typically occurring biological compounds, but are sufficient to heat the compounds when absorbed in such. These lasers form the visible and infrared part of the electromagnetic spectrum. Their photon energies are less that 3.0 eV. There is a wide range of laser types available in this portion of the spectrum to produce these wavelengths and various power levels.

- **Photo-chemical** effects lie within lasers whose photon energies are sufficient to break the chemical bonds in biological compounds. These lasers all lie in the ultra-violet portion of the electromagnetic spectrum from approximately 400 nm and beyond. Their photon energies are greater then 3.0 eV and are principally produced by excimer (excited dimer) lasers [23].

3.2.4 Light Absorption

Only light absorbed by a molecule will have any effect on that molecule. Any light that is reflected from the molecule's surface or that passes straight through will have no effect on the molecule and can be considered as wasted energy.

When light is absorbed by a molecule, the incident radiant energy is converted into rotational and vibrational energy or an increase in the electronic state of the molecule. The chemical structure of a molecule determines the specific wavelength of non-ionising radiation that will be absorbed.

Results of much spectrographical research [33] have concluded that fully saturated compounds do not absorb visible and near ultra-violet light, while compounds containing unsaturated groups (multiply bonded atoms) absorb the longer wavelengths of ultra-violet light. As the number of conjugated unsaturated groups in a compound increase, there is a corresponding increase in the absorption of longer wavelengths of light. The portion of a compound responsible for light absorption is called a *chromophore*.

The effect of light absorption on a cell depends on the specific chemical composition within the cell, that is, on the presence of absorbing molecules or chromophores. The radiation must be absorbed within the cell for it to have any effect on the cell. Molecules in excited electronic states have different chemical and physical properties than those in ground states. These different properties can

have profound effects on the viability of a cell and can be used to promote biocidal action.

Nucleic acids and most cell proteins are essentially transparent to, and completely transmit, visible light, but absorb certain wavelengths in the ultra-violet region (between 250 nm and 295 nm) and can be damaged by this form of radiation [33]. Other macro-molecules in cells which appear coloured will absorb in the visible spectrum of light and hence can be damaged by high intensity irradiation of these wavelengths of light. For example, the absorption of ruby laser light (red) in *spirogyra* to provide a bactericidal effect has demonstrated as early as 1963 [2].

Since the absorption of non-ionising wavelengths is determined by the chemical composition of the organisms being irradiated, the more radiation that a molecule absorbs, the greater the effect of the radiation.

3.2.5 Light Absorption in Water

As discussed in chapter 2, bacteria contain approximately 80 % water. Hence, the absorption of specific wavelengths of light in water are of particular interest to this study.

Water will only be heated by laser light that is absorbed by it. In order to raise the temperature of 1 mm^3 of water from typical body temperature to boiling point, with subsequent vapourisation, 2.52 J of energy are required. This can be calculated from the following:

> *By definition, the thermo-chemical calorie value is the amount of heat required to raise the temperature of 1 g of water from 14.5 °C to 15.5 °C.*

This can also be defined by its Joule equivalent where:

$$4.184\ 0\ \text{J} = 1\ \text{cal}_{\text{th}} \qquad \textit{(Equation 3.5)}$$

Thus to raise 1 g of water from a body temperature of 37 °C to 100 °C will require:

$$4.184\ 0\ \text{J} \times 63 = 263.592\ \text{J}$$

1 g of pure water is 1 cm^3, which equates to 1 000 mm^3. Therefore, to raise the temperature of 1 mm^3 water from body temperature to 100 °C will require:

$$\frac{263.592}{1\,000} = 0.26\ \text{J}$$

The molar heat of vapourisation, is the heat required to vapourise 1 mole of substance. The molar heat of vapourisation of water is 40.7 $kJ \cdot mol^{-1}$. The molar mass of water is 18 g, That is, 1 mole of water weighs 18 g. Hence, 1 cm^3 of water weighs 1 g as the density of water is 1, therefore, 1 mm^3 of water will weigh 1 mg. Hence, the amount of energy required to vapourise 1 mm^3 of water at 100 °C will be:

$$\frac{40.7 \times 10^3}{18 \times 1\,000} = 2.26\ \text{J}$$

In addition, the temperature of the water had to be raised to 100 °C from 37 °C, which required 0.26 J, hence the total amount of heat required to vapourise 1mm^3 of water is **2.52 J**.

From the above discussion it can be seen that the majority of the energy is required to vapourise the mass of water rather than to raise its temperature to boiling point.

Clearly, only the light energy that is absorbed in the water will contribute to its increase in temperature. Hence, the light wavelength that is most readily absorbed in water is of especial interest, to make the most efficient use of available laser power. Under normal conditions a percentage of light will not be absorbed in irradiated water, hence the 2.52 J figure quoted above can be considered the absolute minimum energy level from a laser system to vapourise 1 mm^3 of water.

Figure 3.3 shows the light absorption graph for water [34]. In a homogeneous medium the absorption coefficient is equal in magnitude to the inverse of the skin depth δ. The skin depth being the distance over which an electromagnetic wave (light in this case) will decay to $^1/_e$ (approximately 37%) of its incident value in a conductive medium, and is described by equation 3.6.

$$\delta_s = (2/\sigma\mu\omega)^{1/2} \qquad \textit{(Equation 3.6)}$$

Where:
- δ_s is the skin depth in metres
- σ is the conductivity in mohs/m
- μ is the permeability in henries/m
- ω is the angular frequency in radians/s

From equation 3.6 it can be seen that conductors (with a high value of σ) have a shorter skin depth and consequently higher absorption than do insulators. Pure water is a very good insulator, but as other substances are added to the water (such as the key constituents found in a typical cell) the conductivity of the "water" will be expected to increase, with a corresponding increase in absorption.

Figure 3.3 shows a complex absorption curve for water due to the composite interactions of the vibrations and rotations of the water molecule. As impurities are added to the water a general increase in absorption would be expected, but with little chance of predicting the actual peaks and troughs of the resultant absorption curve.

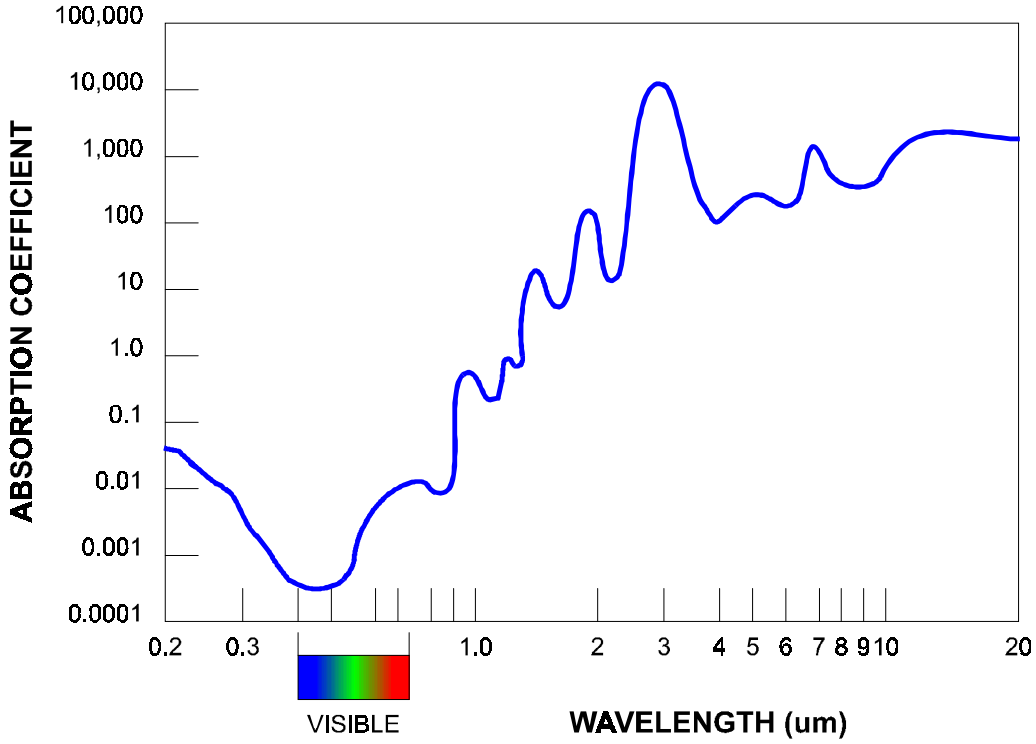

Fig. 3.3: Light absorption in water plotted against wavelength.

Figure 3.3 shows that the lowest absorption of light in water is found in the visible range of wavelengths from 400 nm to 700 nm, as would be expected because water can be seen through. Below 400 nm, ultra-violet light is increasingly absorbed. Above the visible range, in the infrared range, light is increasingly absorbed, with various peaks. The highest absorption levels in water are all found in the longer wavelengths, i.e. the infrared regions, which by nature have a photo-thermal effect as previously discussed.

From figure 3.3, it can be seen that there is an absorption peak at a wavelength of approximately 3 μm. The closest laser to this is the Erbium YAG (Er:YAG) with a wavelength of 2.94 μm. The absorption minimum of the graph is at approximately 500 nm, which has a relative absorbency of approximately 10 million times less than the above peak at 3 μm.

The absorption of the ubiquitous carbon dioxide (CO_2) laser, which has a wavelength of 10.6 μm, is approximately 10 times less than that of the Er:YAG laser. CO_2 lasers are however significantly more efficient than Er:YAG lasers for the following reasons. Er:YAG lasers generate their laser energy by imparting high energy light into an Er:YAG crystal. This is typically done via broadband white light generated by xenon flash-lamps. The typical efficiency of an Er:YAG system is around 0.2 %, due to the nature of a large amount of waste light from the flash lamps not being absorbed in the Er:YAG crystal. The CO_2 laser however, is typically 10 % efficient, as the means of excitation of the lasing medium is more direct. The lasing medium in this situation is stimulated directly by electrical means.

Ninety percent of the incident light of a given wavelength is absorbed in a certain characteristic length, known as the *extinction length*. Essentially all of the energy in the laser beam is deposited in one extinction length. A closely related and more commonly used measure of absorption is the *absorption length*, which is the length over which 63% of the light is absorbed. There are approximately 2.3 absorption lengths per extinction length. The absorption length of the CO_2 wavelength is approximately 10 μm, while that of the Er:YAG is 1 μm. Hence, their extinction lengths are 23 μm and 2.3 μm respectively. Laser light that is absorbed by water heats the absorbing volume instantly (in a period of pico or femto seconds) [34].

While water will typically absorb Er:YAG 10 times more efficiently than the CO_2 laser, the CO_2 lasers are typically 100 times more efficient than the Er:YAG systems. Hence, the CO_2 will be at worst equal in efficient use of power with regards to heating water, and could theoretically be up to 10 times more efficient.

A comparative disadvantage in using CO_2, is that 10 times more energy will need to be imparted into a substrate to heat the water contained in micro-organisms, hence the substrate is likely to receive 10 times more energy. Different substrates however may well absorb CO_2 much less than the Er:YAG light, by a factor greater than 10. In this case, the CO_2 laser would be the preferable option.

A further practical consideration is the relative cost per Watt, and the maximum energy obtainable from the individual laser types. In this respect, CO_2 lasers will prevail, as they are the most common form of laser used for industrial operations and are typically much cheaper per Watt than other laser variants, and can offer significantly higher powers than other technologies. For example, powers of 1 kW are readily available with CO_2 lasers.

Continuous wave (CW) lasers of high power are much easier to manufacture in CO_2 systems than flash-lamp pumped systems. This could also be a significant benefit for commercial systems, as the product to be sterilised may pass through a laser beam with a given relative velocity. If a pulsed laser source were used, areas of the substrate may well be missed, by passing through the laser beam while the beam is not on. This may only be a problem if the pulse repetition frequency is not sufficiently high for a given product velocity.

From the previous chapter, the volume of a typical bacterium is approximately 2 μm^3. 1 mm^3 equates to 1×10^9 μm^3, hence there could be up to 5×10^8 bacteria in a 1 mm^3 volume. Now if 2.5 J of energy were required to vapourise a 1 mm^3 volume of water, $2.5 \div 5 \times 10^8$ would be required to vapourise a single bacterium, i.e. 5.0 nJ.

If all the above bacteria were arranged on a planar substrate in a single plane, with the typical height of bacterium being approximately 0.5 µm, the area covered by 5×10^8 bacteria would be 2×10^9 μm^2, or 20 cm^2. Thus, 2.5 J are required for an area of 20 cm^2, or in other words an equivalent energy density of 0.13 $J \cdot cm^{-2}$.

The standard unit for energy density (or fluence) used in laser terminology is $J \cdot cm^{-2}$, hence the above departure from standard SI units. The above calculation makes some fundamental assumptions about the bacteria, such as that they are uniform height and distribution and are 100 % water. However, this is a useful approximation, giving the minimum amount of energy required to raise the temperature of the bacteria to 100 °C and cause vapourisation.

Other factors not taken into consideration are; the thermal conductivity and reflectivity of the substrate, the different chemical composition of bacteria, spores, viruses and fungi, the ability for micro-organisms to form multiple layers on a substrate and the relative absorption of laser light. All of these effects will contribute to a general increase in the energy density required to kill a bacterium.

Conversely, the previous theoretical calculations assumed that all the bacteria would be completely ablated (vapourised). However, only a single water vapour bubble may need to be generated within a bacterium to compromise the cell's membrane and thus lead to the death of the cell. Furthermore, if simply raising the cell's temperature to 100 ºC is sufficient to kill the bacterium, then this would lead to a great reduction in the energy required to promote sterilisation, as can be seen from the previous calculations in this chapter. These differing kill mechanisms may in turn contribute to a general decrease in the energy density required for efficient sterilisation.

3.2.6 Wavelength Analysis

3.2.6.1 Visible light

From section 3.2.4 (Light Absorption), it is evident that visible light radiation is not an optimal choice for sterilisation due to its poor absorption in some of a cell's key constituent chemical components. Furthermore, section 3.2.5 (Light Absorption in Water) shows that visible light is the least absorbed in water, which typically forms 80 % of a micro-organism's mass (spores excepted).

Past work with ruby lasers, producing visible deep red light at 694.3 nm, on Gram-negative bacteria (*Proteus vulgaris* and *Ps. aeruginosa*) and Gram-positive bacteria (*Staphylococcus* and *Bacillus subtilis*) has concluded that this particular wavelength is not suitable as a bactericidal agent. Ruby energy densities of over 600 $kJ \cdot cm^{-2}$ showed no effects, nor did 2 hours of continuous exposure to a 0.5 mW HeNe laser focussed to a 1.5 mm spot (equating to 200 $J \cdot cm^{-2}$) [3].

Watson *et al.* [7] compared the bactericidal effects of seven different laser wavelengths on *Escherichia coli*. The lasers in the visible spectrum used; argon ion (488 nm) and frequency doubled Nd:YAG (532 nm) produced no bactericidal effects.

3.2.6.2 Visible Light with Photosensitisation

However, visible light has been proved bactericidal, although to increase its efficiency, photosensitising agents have to be added. On a commercial scale, it is desirable to avoid chemical additives, as this is one of the primary reasons for considering laser based sterilisation systems. Photosensitising agents could also prove deleterious to potential substrates to be sterilised.

The addition of 0.01 % *methylene blue chloride* to stain the cell walls of *spirogyra* algae, has permitted the ruby laser to puncture the cell walls with energy levels that previously had no effect on the cell. Ruby laser light is deep red in colour, hence the blue dye acts as the chromophore to absorb this light and cause localised heating of the cell wall with subsequent vapourisation. Prior to the addition of the photosensitising dye, energy densities of 300 $J \cdot cm^{-2}$ were required to effect localised damage areas of 25 μm. After the addition of this dye, the energy levels required to affect the same damage required only one tenth (30 $J \cdot cm^{-2}$) of the energy prior to its addition. These were performed using a pulsed ruby laser with a pulse length of 500 μs [2].

The primary reason for the lower energy levels required to effect damage on a *spirogyra* cell compared to the previous example in section 3.2.6.1 (Visible light) [3] is probably due to the presence of a natural chromophore in the algae, namely the chloroplasts, giving the algae the characteristic green hue.

Experiments performed with a 7.3 mW HeNe laser (632.8 nm, red) on the bacterium *Streptococcus sanguis* highlighted three effective photosensitising agents; *methylene blue*, *azure B chloride* and *toluidine blue O* at concentrations of 0.005 % (wt/vol). For each of these dyes, further experiments were performed on oral bacteria *Porphyromonas gingivalis*, *Actinobacillus actinomycetemcomitans* and *Fusobacterium nucleatum*. For each case, biocidal action was observed to exposure of laser light for 30 seconds. The effective energy doses for these effects ranged from 2.75 $J \cdot cm^{-2}$ to 33 $J \cdot cm^{-2}$. Tests performed without the sensitising agents showed no bactericidal activity [25]. It is interesting to note that all of the above dyes stained the micro-organisms blue. The dyes also had peak absorption close to the HeNe wavelength, thus acting as the chromophore for the transfer of energy into the micro-organisms.

Other trials also using a 30 mW HeNe laser operating at 632.8 nm with a 1 mm diameter for 90 minutes showed that the addition of the photosensitiser *toluidine blue* allowed laser sterilisation to be achieved on 11 different strains of micro-organisms in aqueous suspension. None of these micro-organisms were sterilised without the addition of the photosensitising dye [24].

From these discussions, light wavelengths in the visible portion of the electromagnetic spectrum will now be disregarded as not being feasible for application in large-scale systems. While visible light can produce biocidal effects, the requirements to add photosensitising agents to promote efficacious treatment remove the advantages of the laser systems over other methods. The ultra-violet and infrared radiations will now be considered in detail as being the more promising candidates.

3.2.6.3 Ultra-violet Light

Previous discussions have shown that high-energy photons emitted from ultra-violet sources can prove lethal to micro-organisms. Ultra-violet lasers provide intense sources of this region of radiation. The most common form of ultra-violet laser is the excimer laser, first developed in 1975. Such lasers typically use a noble gas and a halogen, which when energised form an excited dimer (hence the name excimer) [23].

With varying combinations of the noble and halogen gases, ultra-violet wavelengths from 351 nm to 157 nm (Table 3.1) can be achieved. These lasers are readily available for commercial applications, and can produce parameters of 200 Watts, 1 kHz repetition rates, 4 J pulses and pulse durations of 10 ns to 250 ns. Excimer lasers typically have efficiencies of 1 % to 4 % comparing the output laser power to the input electrical power. The maintenance free lifetime of these lasers is typically of the order of 5×10^9 pulses [23]. One major downside to the use of excimer lasers however, is the use of halogen gases, which are not particularly environmentally friendly.

Ultra-violet light is known to have cell mutagenic action, as has been previously discussed, by a photo-chemical process. Peak DNA damage is caused by single photon absorption at 260 nm wavelength, i.e. 4.77 eV [34]. The excimer laser closest to this wavelength is that of the KrF (248 nm, 5.13 eV). Germicidal lamps tend to also centre on this wavelength, and are typically 253.7 nm. It is also known that irradiation of these frequencies can be carcenogenic in human tissue, such as in promoting malignant melanomas (skin cancer).

While ultra-violet light is much less absorbed in water than infrared wavelengths, the effect of the ultra-violet light on cells is not photo-thermal, but photo-chemical, thus the absorption by water is of little interest from the ultra-violet view point. The ultra-violet light causes its damage to cells via direct interaction with key cell constituents. Hence, the lower the absorption of ultra-violet light by water, the better, as the incident light will have a higher remaining energy with which to inflict damage on the cell's molecules.

Research conducted by Karoutis *et al.* [35] has shown that the ArF excimer laser (193 nm, 6.4 eV) is strongly biocidal. This laser wavelength is preferential to that of the longer wavelengths produced by the likes of the KrF (248 nm) laser as the photon energies are strongly absorbed in the cell proteins, before reaching the cell's nucleus. Thus, the micro-organisms are more likely killed by DNA disruptive photoproducts instead of DNA mutagenic action. These effects are likely to be more immediate and predictable than DNA mutagenicity.

Tests have shown that using an ArF excimer laser for sterilisation, with increases in laser energies, pulse numbers and repetition rates all contribute to increased kill levels [35]. Typically, 10 $J \cdot cm^{-2}$ to 15 $J \cdot cm^{-2}$ are needed to kill bacteria, achieving up to an 8 log kill rate.

Watson *et al.* [7], in their comparison of seven different wavelengths of laser on bactericidal activities, utilised a frequency tripled Nd:YAG system producing ultra-violet light of 355 nm. This wavelength is close to that emitted by a XeF excimer laser (351 nm). While the effects of these tests were biocidal, they were low compared to other lasers tested (for example the CO_2 laser). The typical energies densities used by the above laser in these tests were of the order of 10 $J \cdot cm^{-2}$. The low levels of bactericidal activity may be attributed to the longer wavelength of the ultra-violet light compared to that of the ArF laser as discussed above.

United states patent 5 439 642 cites an example of the use of ultra-violet radiation to sterilise contact lenses, with wavelengths of 100 nm to 350 nm and energy densities of 0.1 $J \cdot cm^{-2}$ to 10 $J \cdot cm^{-2}$ suggested [36].

Frucht-Pery *et al.* [37] demonstrated the use of an ArF excimer laser to sterilise the fungus *Candida albicans* with approximately 800 pulses at an energy density per pulse of 0.3 $J \cdot cm^{-2}$ and pulse rate of 10 Hz. This equates to an accumulative energy density of 240 $J \cdot cm^{-2}$.

This body of evidence shows that the use of ultra-violet laser technology to sterilise items in industrial applications is being taken very seriously. The use of the shorter wavelengths has been selected as they will not provide significant damage to the item being processed at the given energy densities compared to longer wavelengths such as CO_2 lasers.

3.2.6.4 Infrared Light

Moving to the longer wavelengths of the electromagnetic spectrum, the infrared lasers become the next focus of attention. The most common of these being the Nd:YAG laser (1.06 μm wavelength) and the CO_2 laser (10.6 μm wavelength), the two laser systems upon which much recent work has centred pertaining to bactericidal activities.

However, as has been discussed previously, the infrared Er:YAG laser has the highest absorption peak in water of any laser. Unfortunately, there has been little work conducted into the bactericidal effects of this laser, with the notable exception of Hibst *et al.* [38] who studied the effects of this laser on extracted human teeth with carious lesions containing caries bacteria. An energy density of 1.7 $J \cdot cm^{-2}$ per pulse with a total of 50 pulses giving an accumulative energy density of 85 $J \cdot cm^{-2}$ proved efficacious in sterilisation. This demonstrates a real scenario in which the bacteria being targeted reside in the caries of human teeth rather than a homogenous agar medium in a petri dish.

3.2.6.5 Infrared Light - Nd:YAG

Nd:YAG lasers are as common in industry as the CO_2 laser, and with this in mind their availability, state of development, cost and power outputs are likely to have been optimised over the years to produce highly efficient and cost effective lasers. From the water absorption graph (figure 3.3) it can be seen that the Nd:YAG laser

with its 1.06 μm wavelength is almost 1 000 times less absorbed in water than the CO_2 laser. Moreover, being a flash-lamp pumped system, the Nd:YAG laser will be 10 times less efficient at least than the CO_2 laser. This leads to a relative predicted efficiency between the two systems of 10 000 times in favour of the CO_2 laser, considering the thermal heating effects of water only.

Nd:YAG lasers have proved effective sterilisation agents, but require relatively high energy density levels to achieve sterilisation, for example an energy density of 144 $J \cdot cm^{-2}$ is required for moist *Bacillus stearothermophilus* spores (dry spores needed higher energy levels again) [39].

Experiments with a Nd:YAG laser on *Pseudomonas aeruginosa*, *Staphylococcus aureus* and *Escherichia coli* have shown that 1 667 $J \cdot cm^{-2}$ were required to provide between 2 log to 8 log of kill. This study also investigated the addition of photosensitising dyes to promote biocidal action of the Nd:YAG laser. Methylene Blue did prove to aid the bactericidal effect on *Pseudomonas aeruginosa,* however this bacterium also has the ability to produce pigments, one such being a bluish-green pyocyanin, which would give it a predisposition to absorbing red light. A further investigation of this study hinted that the temperature increase in the substrate or carrying medium of the micro-organisms due to the irradiation by laser can also contribute to the biocidal activity of the laser [5].

A recent study conducted on several bacteria and yeasts on agar with a Nd:YAG showed that levels between 1 768 $J \cdot cm^{-2}$ to 4 489 $J \cdot cm^{-2}$ were required to completely inactivate the bacteria over half the surface area of the laser spot size. The most resistant strain was *Bacillus stearothermophilus*, which belongs to the group of bacteria resistant to high temperatures. The strains of bacteria tested included various morphologies and Gram strains, with no particular strain showing any great advantage over the others. One organism tested, *Deinococcus radiodurans* is know to be highly resistant to ionising radiation, but this did not prove resistant to the laser irradiation [40].

Experiments with *Enterococcus faecalis*, a bacteria which is relatively heat resistant, Gram-positive, non-spore forming and a facultative anerobe, have been conducted by Rooney *et al.* [41] which again confirm the biocidal effect of the Nd:YAG laser. These trials have shown a 4 log kill for a total imparted energy of 54 J. Unfortunately, due to the nature of their experiment, an energy density level was not available.

3.2.6.6 Infrared Light - CO_2

Tests have shown that CO_2 lasers are equally efficient at providing bactericidal effects on both Gram-negative (*Escherichia coli*) and Gram-positive (*Staphylococcus aureus*) bacteria [42]. The same tests achieved between 4 log to 7 log of kill depending on the use of a focused or unfocused beam, with the focused beam being the least efficient. The reason for the inefficiency of the focused beam is that the treated area may not have been completely covered due to inaccurate scanning mechanisms. The focused beam of 0.2 mm diameter provided an energy density of 1.2 $MJ \cdot cm^{-2}$, while the unfocused beam of diameter 3 mm gave 3 540 $J \cdot cm^{-2}$.

Clinical tests with CO_2 lasers *in vivo* have been proven to reduce post-operative infection in amputation cases. Additional tests were performed on rabbits *in vivo* with the addition of *Pseudomonas aeruginosa* to open wounds. After treatment with a 15 W CO_2 laser less than 10 % of the wounds caused infection post laser treatment compared to 40 % being treated with iodine solutions. There is not sufficient data to extrapolate the energy densities used, but it is clear that the CO_2 laser has a bactericidal action [43].

A test performed on metal scalpel blades with a 10 W CO_2 laser for durations of 1.5 minutes to 2.0 minutes, that had been previously inoculated with *Bacillus subtilis* and *Clostridium sporogenes* spores, showed 100 % sterilisation on every scalpel blade [4].

In a comparative test of lasers operating at seven different wavelengths (including Nd:YAG) on *Escherichia coli* cultures, the CO_2 laser achieved bactericidal effects at between 1.3 $J \cdot cm^{-2}$ to 8 $J \cdot cm^{-2}$, whereas the Nd:YAG laser required 1 210 $J \cdot cm^{-2}$ to 1 940 $J \cdot cm^{-2}$, approximately 1 000 times greater [7]. This concurs with the previous prediction from the water absorption graph (figure 3.3) that the CO_2 laser would be 1 000 times more absorbed in water and therefore 1 000 times more efficient in the sterilisation of micro-organisms. Moreover, with the CO_2 laser being inherently more efficient, the real power efficiency is more likely to be 10 000 times that of Nd:YAG.

United States patent 3 941 670 defines a method of altering biological and chemical activity of molecular species. In particular, it quotes the use of an unfocused 23 W CO_2 laser to inactivate **dry** *Bacillus subtilis* spores in 100 ms [44]. No mention of the beam diameter is made, so the relevant energy density cannot be calculated. This effect is of particular interest as spores contain little water, particularly when they are not contained in solution. It is anticipated that most organisms requiring sterilisation are likely to be vegetative, but the effect of the CO_2 lasers on spores is especially encouraging as a potential source for a generic laser system.

Hooks *et al.* [45] showed that endodontic stainless steel reamers seeded with *Bacillus subtilis var. niger* spores and *Bacillus stearothermophilus* spores and treated with a 10 W CO_2 laser for 3 seconds per surface showed no subsequent growth of the micro-organisms. This trial further demonstrates the efficiency of the CO_2 laser in sterilising bacterial spores. Unfortunately, there is insufficient information from these trials to extrapolated the system's energy density.

3.3 Selected Laser Wavelength

Following the above discussions it was decided that a carbon dioxide laser system would be the preferential choice for a generic large-scale laser based sterilisation system.

While the excimer lasers have proven to have high bactericidal effects, they are still relatively costly and inefficient in power terms compared to the CO_2 lasers. Additionally, the use of halogen gases with the excimer lasers has a negative environmental impact, where the proposed use of lasers for sterilisation is to promote a clean alternative to other traditional methods. Excimer systems, being pulsed systems, also give more processing problems when being implemented in a continuous process to achieve uniform kill levels.

While the Er:YAG is theoretically the best choice for the sterilisation of bacteria due the high absorbency of its wavelength in water (the primary chromophore), its poor system efficiency, high cost and pulsed operation compared to the CO_2 laser make it a less appealing choice for a commercial application.

The Nd:YAG system, again proved to provide bactericidal action but is 1 000 times less efficient at the process than a CO_2 laser.

As part of this study a CO_2 laser is to be designed into a commercial scale sterilisation system for the sterilisation of eggs from *Salmonella enteritidis*. This will enable the quantification of such a system under real operating conditions instead of laboratory conditions.

The following outlines the key parameters that influenced the decision to utilise a CO_2 laser source:

- Highly efficient laser system, typically 10 %
- Readily available and comparatively cheap
- Flexible to interface into control systems
- Theoretically the best commercial choice
- Have demonstrated high sterilisation rates in laboratory trials
- 1 000 times more efficient than Nd:YAG for sterilisation
- Available in continuous wave
- Environmentally friendly

With the laser system and wavelength now settled upon, there are further considerations of the laser's spatial and temporal profiles for the processing of the aforementioned eggs. These two areas will now be discussed in detail.

3.4 Spatial and Temporal Profile Analysis

The efficiency of a given wavelength of laser for the sterilisation of micro-organisms can be significantly altered by the temporal and spatial profile of the laser's output and the surface texture of the substrate being treated.

Lasers come in two forms, continuous wave (CW) and pulsed. The pulse shape, repetition rate, mark-space ratio and pulse length are all parameters that can be varied within the temporal profile of a pulsed laser. There are less parameters that can be modified in the temporal domain for the CW lasers. The key parameter being the duration of exposure of the CW beam to irradiate a given surface.

The spatial profile of a laser beam output refers to the distribution profile of the laser's energy within a given area. These spatial profiles of raw laser beams are termed modes and are effected by various physical parameters acting on or within the lasing medium and laser cavity.

Curved and irregular surfaces present significant problems to the laser sterilisation of these surfaces, altering the spatial profile of the incident laser beam, as seen by the surface. Reduced fluences will be found when the incident laser beam does not hit a surface perpendicularly. These reduced fluences will reduce the effectiveness of the laser's sterilisation abilities. A further complication for irregular (rough) surfaces, will be certain areas of the surface in micro-cavities may be in shadow from the incident laser light. Additionally, bacteria also have an affinity to grow in micro-cavities, thus compounding the problem [35].

3.4.1 Spatial Analysis

For the application in question, a uniform energy density profile is required from the emergent laser beam, and particularly the spatial profile on the surface being treated.

The spatial profile of a laser beam is commonly referred to as its mode, or more accurately its **transverse electromagnetic mode** (TEM). These modes can vary from the simplest TEM_{00} (often called the uniphase mode) to much more complex, higher order modes, which can look like mountain ranges (with numerous peaks and troughs of laser energy distributed over the laser's spot). These modes are designated TEM_{qr}, where q and r are integers referring to the numbers of minima (or phase reversals) as the laser beam is scanned horizontally and vertically respectively. The TEM_{01}^{*} mode is a combination of TEM_{01} and TEM_{10} modes and is sometimes referred to as the doughnut mode. Figure 3.4 shows a selection of low order TEM modes [31].

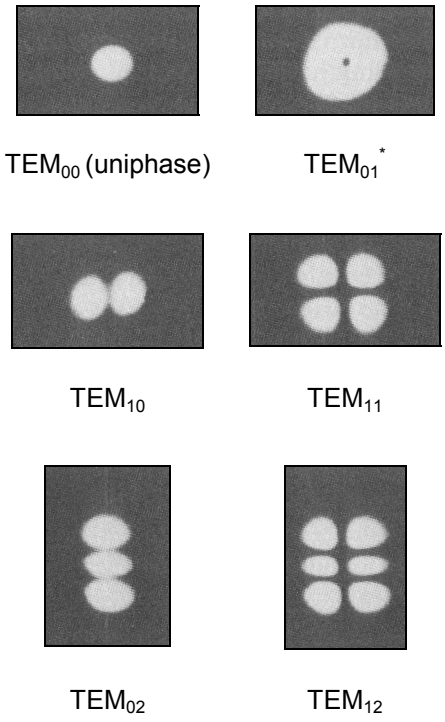

Fig. 3.4: Low order laser transverse electromagnetic mode patterns.

To give the best starting point for uniform coverage of substrates and uniform kill rates, it is desirable to choose the most homogenous TEM mode. This is the TEM_{00} mode, with its Gaussian (or bell shaped) beam profile. The TEM_{00} mode also has the greatest spectral purity, degree of coherence and lowest divergence for the emergent laser beam of all the TEM modes. Analysing the energy at a cross section of such a laser spot will give a graph as shown in figure 3.5. Conveniently, the majority of commercial CO_2 laser systems on sale today are offered with the TEM_{00} mode (or near TEM_{00} mode) as standard, as this is the most uniform and efficient mode for most applications.

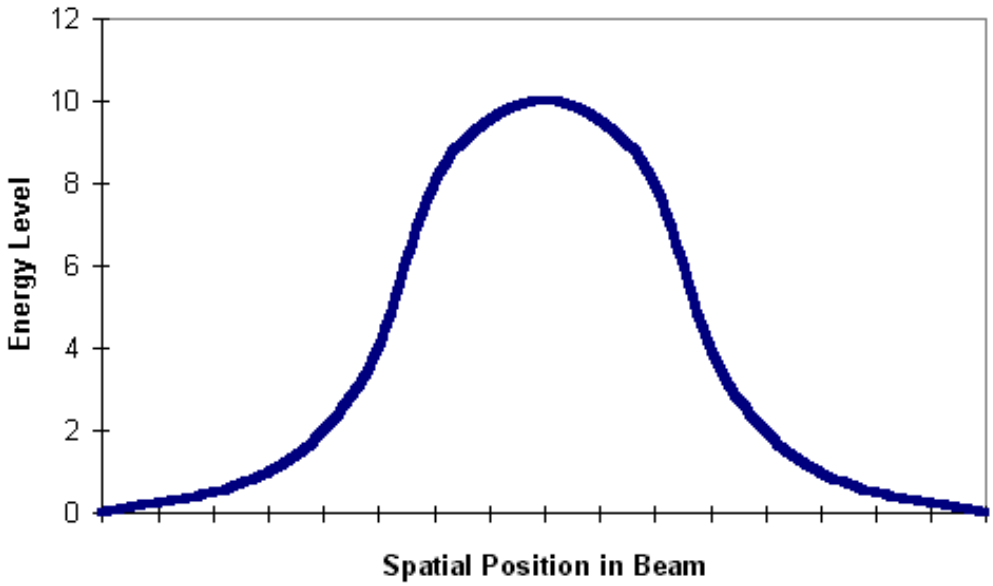

Fig. 3.5: Energy profile of TEM_{00} mode laser beam.

In addition to the effects of the laser mode on the uniformity of substrate coverage, the topographical nature of the item(s) to be sterilised will have a significant impact upon the efficiency of the sterilisation procedure. The ideal environment is in which the incident laser beam will impinge upon the substrate perpendicularly.

If the angle of the surface presented for irradiation varies from 90°, then the energy contained within in the laser's beam will become diluted over a larger surface area. This reduces the energy density imparted to the treated substrate. Following from the previous discussions, energy density (fluence) is the critical factor determining effectiveness of bactericidal activity. Hence, for a given area a sufficient amount of energy must be imparted to kill all bacteria. If this area now increases, but the given energy remains constant, then the effective energy density has decreased and the possibility of certain bacteria surviving increases.

This effect is of major importance to the continuing study as all items to be treated in the real world will have a multitude of shapes and sizes. The choice of eggs for this study was brought about by recent scares of eggs being contaminated with *Salmonella enteritidis*, this however has presented serious processing problems with the egg shape, as the egg curves in every one of the three classical dimensions, x, y and z. To achieve a near constant energy density, while covering the whole egg, is a difficult task, but has been achieved. Chapter 4 describes in detail the design and construction of a prototype system for sterilising eggs using a CO_2 laser and numerous processing techniques to try and maintain a constant energy density, while covering the whole surface area of the egg.

3.4.2 Temporal analysis

The chosen laser is a continuous wave CO_2 laser, and as such there are no pulse shape, duration or repetition frequency parameters that can be directly varied. The CW beam from the laser can be gated and its power varied. That is, the beam can be switched on and off by a control system and the beam's power can be continuously controlled by the same control system. This would give the ability to mimic laser pulse trains if so desired. These parameters give some control of the laser's temporal profile.

Furthermore, the above laser beam could impinge on a controlled scanning mirror. Such a mirror can sweep the laser beam over a given surface at a controlled velocity. This scanning mirror gives the laser beam another temporal effect with regards to the surface being sterilised. The faster the angular scan of the mirror, the faster the laser spot will move across a substrate's surface, thus the dwell time for the laser spot at a given point will be reduced. This in turn will reduce the amount of light energy imparted to the substrate's surface and reduce the energy density seen by any particular point on the substrate's surface.

Under many industrial or commercial applications, products to be sterilised would have to be scanned by a laser, or continuously moving products would pass by a static laser beam. If a train of laser pulses were used, and the relative velocity between the laser beam and the substrate were too high, then there may be untreated gaps between consecutive laser pulses, where the product had not been irradiated.

Pulsed systems can often have advantages over CW systems, as the peak power of the individual pulses can be many factors greater than the average power of the CW laser. These are very good for ablating small areas with high peak powers, raising localised areas to very high temperatures, without surrounding areas heating up due to the thermal inertia of the substrate and the short length of the pulse.

A CW system will prove advantageous for general sterilisation applications, with the laser beam being continuously on there should be less chance of any area being missed, depending on the type of laser scanning technology adopted. Furthermore, the laser may also impart heat to the upper layer of the substrate, which could well help to kill bacteria hidden by shadowing by a secondary thermal heating effect as has been hinted at in previous laboratory trials [5]. Obviously, this system will have to be carefully balanced for particular substrates, taking care not to impart too much heat to cause any deleterious effects to the substrate's surface or beyond.

3.5 Summary

The beginning of this chapter looked at the broad electromagnetic spectrum with a particular focus on the middle portion of the spectrum – light. This portion of the electromagnetic spectrum is covered by a wide range of commercially available lasers from the infrared CO_2 laser, through the visible HeNe laser to the ultra-violet excimer lasers.

Each laser wavelength above has an associated photon energy ranging from 0.12 eV for the CO_2 laser to 11.27 eV for the H_2 excimer laser. These photon energies have different effects on biological organisms. The lower photon energies have *photo-thermal* effects on micro-organisms, i.e. the laser energy absorbed by a bacteria produces a raise in heat of the organism. Higher photon energies have a predominantly *photo-chemical* effect, with the photon energies being sufficient to break apart atomic bonds. For example, ultra-violet light of 3.0 eV has sufficient energy to disassociate the C-N bond.

Only light actually absorbed by a micro-organisms will have any biocidal effect on that organism either through photo-chemical or photo-thermal effects. The part of a substance that absorbs light is called a chromophore. As bacteria are composed of 80% water, this makes the best choice of chromophore within bacteria.

Water absorbs infrared light greater than other portions of the light spectrum leading to photo-thermal biocidal effects in micro-organisms from these wavelengths. The absorption peak of water is at 3 μm, close to the wavelength of the Er:YAG laser. While the CO_2 laser is absorbed 10 times less in water than the Er:YAG laser, it is 100 times more efficient making it the preferable choice for a commercial system. While the Nd:YAG laser is a common industrial laser, it is absorbed 1 000 less in water than the CO_2 laser and is typically 10 less efficient. This difference has been observed in recent research comparing the biocidal activity of the CO_2 and Nd:YAG lasers, concluding that the CO_2 laser is the most efficient for the sterilisation of bacteria [7].

With the choice of laser wavelength defined for optimum sterilisation efficiency and commercial implementation the remaining laser parameters of spatial and temporal profile had to be defined. The spatial profile of TEM_{00} was chosen for its homogenous quality and low divergence. While CO_2 lasers are most commonly CW, pulsed versions are available, i.e. the Transverse Excited Atmospheric (TEA) laser. However, for this application a pulsed laser was deemed inappropriate as the gaps between consecutive laser pulses may lead to areas of the substrate being treated receiving no irradiation.

4 : EXPERIMENTAL SYSTEM DESIGN

4.1 Introduction

The overriding conclusion from the preceding chapter was that a CW CO_2 laser was the preferred choice for a commercial scale sterilisation system based on system cost and bactericidal efficiency.

All previous discussions and research papers have only considered static targets with static laser beams. In a commercial environment, high volumes of product would have to be treated, more than likely moving through a continuous process. The processing demands for such systems would either require product to be held stationary while laser processed, or require the laser system to sterilise moving targets. Moreover, common items being sterilised have numerous surfaces, all of which will require sterilisation. Such products would require either multiple laser beams impinging on the product from a variety of directions, or fewer laser beams scanning the surface of a product.

With the introduction of complex surfaces and moving targets, the requirements of meeting uniform dose levels per unit area of product become increasingly difficult. Indeed the complexity of such scanning systems in the requirements mentioned above have already been predicted in recent publications as being potentially troublesome [35].

The following sections of this chapter outline the approach adopted and the reasons behind the decisions made, to irradiate a moving three-dimensional target with uniform dose levels.

The targets for this application were chicken eggs destined for breeding stock - not eating eggs. Part of the reason for the initial trials being conducted on hatching eggs compared to eating eggs is their relative value. The value of a hatching egg from quality breeding stock can be up to £10 each. The high cost of these eggs was sufficient to justify the development costs for the proposed research system, over £250 000.

It is advantageous to ensure that the surface of hatching eggs are free from contaminating micro-organisms, particularly *Salmonella enteritidis*, which can be passed on to the emerging chicks via the shell when a chick hatches, or can enter the egg via pores in the egg's shell [18]. By a reduction of the surface micro-organisms found on the shells, it is hoped to increase the number of chickens bred from a given number of eggs and hence increase the overall profitability of the complete chicken / egg production cycle. With particular recent media publicity regarding *Salmonella* contamination of eating eggs, the treatment of eating and hatching eggs are both of interest. If *Salmonella* can be eradicated from the hatching eggs via surface sterilisation, this can have a profound effect of the complete production cycle of eating eggs.

The design of the experimental system has incorporated as many variable process parameters as possible, allowing the full analysis of the ideal sterilisation requirements for eggs via laser on a commercial scale. The specific design criteria will be individually examined in this chapter, but first the structure of the egg and its particular requirements and problems associated with handling and susceptibility to damage will be investigated.

4.2 Egg Structure

The proposed system must maintain laser power levels below the egg's surface and internal damage thresholds. In addition, the mechanical handling of the eggs for processing must also be achieved in such a fashion that is not deleterious to the egg.

If the laser power becomes too high or mechanical shocks are imparted to the egg, one risks damaging the egg and reducing the chick's viability. With this in mind, it is advisable to first analyse the detailed structure of the egg, with particular attention being paid to the external parts that will be exposed to the laser beam and mechanical handling systems.

Eggs consist of three main parts, the shell, albumen and yolk. Figure 4.1 shows the inner structure of a typical egg:

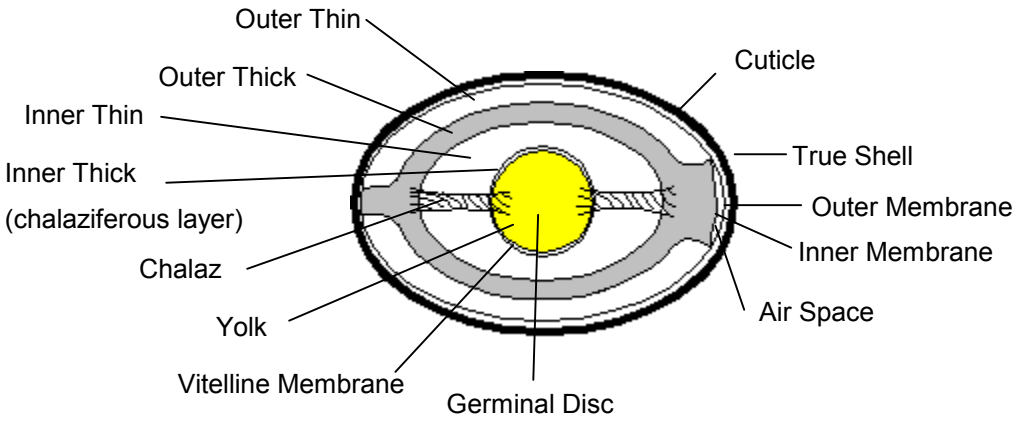

Fig. 4.1: Internal structure of an egg.

The three key areas of the egg will now be discussed individually with reference to the above diagram.

4.2.1 Egg Shell

4.2.1.1 True Shell

The true shell is approximately 0.30 mm to 0.37 mm in thickness (membranes excluded) with a density of 60 mg·cm^{-2} to 100 mg·cm^{-2} and weight of 5.0 g to 7.5 g [46]. The weight of the shell accounts for 9 % to 12 % of the overall weight of the egg (depending on size) [47]. The shell is rigid but brittle, with its mechanical strength being obtained from several contributory factors, not least of which is its shape (others being the shell thickness and its physical composition). Due to the curvilinear nature of the egg, shell strength varies according to position on the shell (as well as varying shell thickness and shell density). Being brittle, the shell

is easily broken by the emerging chick, while still being an excellent container to protect the incubating embryo.

There is evidence that the strength of the egg shell under compression or impact depends upon the rate at which the energy is transferred to it - the higher the rate, the greater the strength. It is clear that thinner shells are more likely to suffer cracks. Anderson & Carter (1975) showed that when otherwise well-formed eggs impact upon a heavy, stiff body a significant number will fracture if the drop exceeds 3.3 mm, which corresponds to a velocity of 250 $mm \cdot s^{-1}$ [48].

The shell is formed from approximately 94 % calcium carbonate ($CaCO_3$) with small amounts of magnesium carbonate, calcium phosphate and other organic matter [47]. The shell is deposited in such a way that an inner *medullary* layer of inverted cones is formed (figure 4.2). The *mammillary* layer represents approximately one third of the shell's thickness. The outer "spongy" *palisade* layer forms the remainder of the true shell.

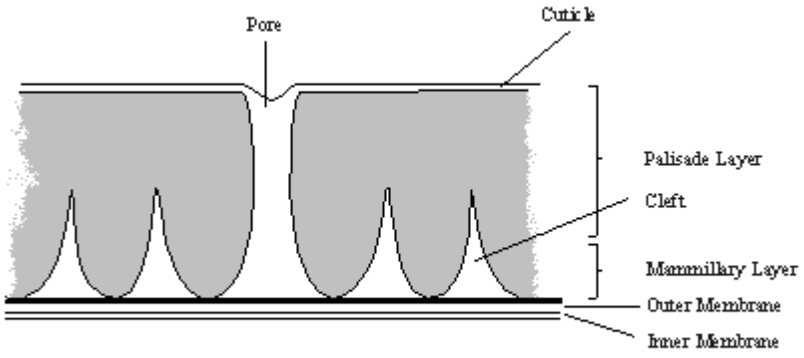

Fig. 4.2: Transverse section of an egg's shell.

4.2.1.2 Pores

There are between 7 000 and 17 000 pores distributed over the shell's surface allowing gaseous exchanges to occur between the contents of the egg and the surrounding environment [47]. These pores are approximately 10 μm to 70 μm in diameter.

However, the pores are not uniformly distributed over the entire shell surface area, but have a significant predominance at the broad end of the shell than the narrow. In fresh eggs, the outer cuticle may also seal the pores.

As well as allowing the exchange of gaseous products, the pores also present an opportunity for the entrance of micro-organisms into the egg. However, the egg has a host of defence mechanisms against invading micro-organisms, one of which is the cuticle, with others being the true shell, membranes and albumen. Damage to the shell and cuticle can assist in the invasion of micro-organisms, and hence must be avoided with the laser processing and handling of the eggs.

4.2.1.3 Cuticle

The characteristic bloom of a fresh egg is due to the presence of a thin, transparent organic cuticle on the outer surface of the true shell. Once the cuticle is dry, it is very resistant to damage, although during storage the cuticle can begin to disintegrate. If this cuticle is compromised in any way, there exists a potential passageway for pathogenic organisms into the egg and the embryo.

During laser processing of the egg, it is vital that damage to the cuticle is avoided, as this would leave the egg vulnerable to attack. The choice of laser power density and processing time needs to be carefully controlled to provide the required levels of sterilisation of the egg's surface without damage to the cuticle. Any mechanical handling of the eggs must also be sufficiently gentle to prevent damage to the cuticle for the same reasons.

4.2.1.4 Shell pigmentation

Shell colour is caused by the presence of a red-brown pigment (*ooporphyrin*) being deposited on the outer surface of the shell and in the cuticle. A blue-green pigment (*oocyan*), which can be found in some shells, is deposited throughout the shell however.

Different pigment colours will have different absorption characteristics for given wavelengths of light and so could have a potential impact upon the laser power levels utilised for different types / colours of eggs.

4.2.1.5 Membranes

The inner surface of the shell has two membranes. These are firmly attached to each other throughout the egg, except at the broad end of the egg where an air gap is formed.

The outer membrane is approximately 50 μm thick, while the inner membrane is approximately 20 μm thick. The outer membrane forms the foundation of the shell and consequently any deformation in this membrane can lead to a deformed shell.

A fine network of *keratin* based fibres within the membranes exists, particularly in the denser inner membrane, which forms an excellent barrier against potential pathogenic micro-organisms.

4.2.1.6 Air Space

The typical body temperature of a hen is 41.5 °C. As soon as an egg is laid, the contents of the egg begin cooling and hence contract. Air is drawn into the egg via the pores and the semi-permeable membranes during this process, forming a small air space (a little less than 2 mm in height) at the broad end of the shell. This air space probably occurs at the broad end of the shell due to the higher concentration of pores in this region and therefore has a higher rate of gaseous exchange.

The size of the air space varies depending on the size and shape of the egg, the permeability of the shell, the surrounding temperature and humidity and the age of the egg. As eggs age, water is lost from the egg by evaporation, hence increasing the size of the air space. This can be used as a useful gauge for determining the approximate age of an egg.

4.2.2 Egg Albumen (Egg White)

The clear jelly-like albumen accounts for approximately 67 % of the egg's weight [49] and consists of four layers. The yolk is surrounded by a narrow layer of **inner thick** albumen (*chalaziferous* layer), which is extended at two points to form the *chalazae*. These fibrous *chalazae* straddle the **inner thin** albumen and are attached in the capsule of **outer thick** albumen. The outer thick albumen is loosely attached to the shell at both ends and is in turn surrounded by a layer of **outer thin** albumen. Hence, the yolk is held near the centre of the egg and is prevented from making contact with the shell.

This damping effect of the albumen is critical in protecting the developing embryo from physical damage due to mechanical shocks and vibration of the egg. The effect will greatly help in the mechanical handling of the egg in the proposed experimental process.

The albumen also forms a biological barrier and contains substances, which can inhibit the activity of any potentially pathogenic micro-organisms breaching the outer defences of the shell.

The albumen will also act as a thermal heat sink, conducting the high temperatures reached at the surface of the egg's shell, generated by the laser system, away from the shell's surface. Due to the relatively large volume of the albumen compared to the short duration of the laser beam and the surface area treated, the relative internal temperature rise of the albumen is expected to be insignificant.

4.2.3 Egg Yolk

The egg yolk is a nutritious material enclosed within a thin transparent *vitelline* membrane. The yolk surface is a uniform yellow viscous liquid, with the exception of the germ cell (*blastodisc*), which is the region of cellular division in a fertile egg.

4.3 Specific Egg Handling Requirements

The objective of producing a commercial scale research system required certain criteria being met to enable the research system to integrate with standard hatchery operating conditions, with regards to egg processing and handling.

Standard plastic trays are used within hatcheries to hold and transport eggs. These trays contain 12 rows of eggs, with each row containing 11 eggs, a total of 132 eggs. The spacing between consecutive eggs in a row is 80.5 mm (between centres). The spacing between eggs in these trays was adopted for use within the system design to facilitate handling, and provide fixed distances between eggs.

This meant that standard egg trays could be used to load and unload a commercial scale working research system in a normal operating environment.

Being of a fragile nature, the eggs were picked up by vacuum rubber suction cups while being transported through the research system machine. This is a well-established approach to egg handling, with very few breakages occurring.

The average diameter of a chicken egg is 45 mm at its widest point, with its typical height being 60 mm.

4.4 Concept Design

Before proceeding to a commercial scale system for full evaluation, certain key variables and concepts had first to be evaluated. These would give an indication of the laser power levels required and potential line speeds that could be accommodated for given laser powers to achieve adequate kill levels. The key ideas providing uniform dose levels could be evaluated before committing to a final design.

A prototype system was configured to process a row of seven eggs, spaced at 80.5 mm centres (as per standard egg tray) at a linear speed of approximately 150 $mm \cdot s^{-1}$. A linear conveyor belt was used for these trials with a variable speed motor drive that could be adjusted to test the system's effectiveness at different line speeds. Attached to the conveyor belt was a shaft encoder, enabling the test system's computer control unit to monitor the belt's speed, making any alterations necessary to compensate for fluctuations in velocity.

This system incorporated some of the key features proposed in the final system design, which would help to achieve a uniform laser dose over the complete surface area of an egg. The concepts behind these features will be discussed below, with the technical implementations of these processes being discussed in detail in the following sections of this chapter.

The system was under microprocessor control enabling parameters to be adjusted, and in essence constituted the core of the project with all major subsections of the final system being present. Numerous trials conducted on this set-up helped to determine the ideal mounting locations of the relevant optical systems and laser powers / line speeds for the final system.

4.4.1 Resonant Scanner Concept

The natural beam emitted from a CO_2 laser is approximately circular and of 4 mm diameter for the particular laser chosen. Under most processing conditions, this beam will need to be manipulated by a series of optics to provide a more useful beam profile. In this application, a line of laser light would be more appropriate than a single spot.

The initial intention of the system was to pass an egg past a stationary curtain of laser light covering the complete height of an egg, as can be seen in figure 4.3. In this way the complete egg would be covered with laser light, providing the egg was illuminated from both sides. Figure 4.3 shows an egg being illuminated from a single side only.

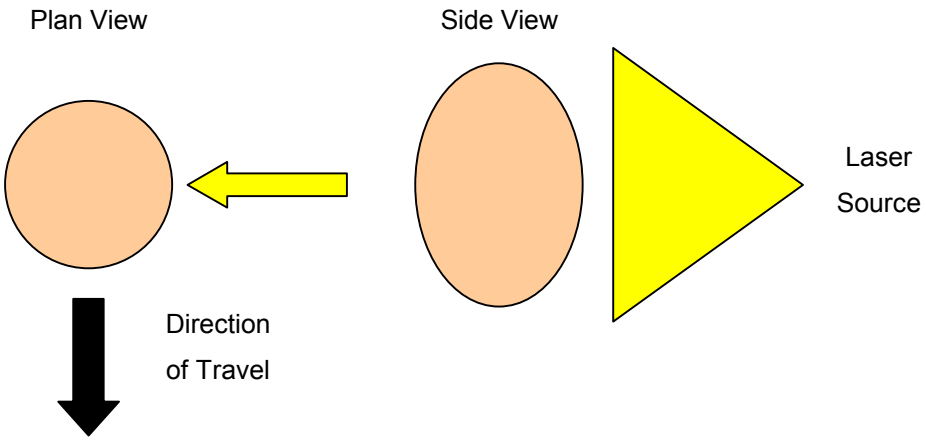

Fig. 4.3: Egg passing stationary curtain of laser light.

The generation of a line of laser light can be achieved in one of two ways, either using a fixed optical system or by mechanical manipulation of the laser beam.

The fixed optical arrangement was not desirable, partly due to the prohibitive cost (£6 000 per laser beam processed) and partly due to the nature of the light beam emitted. The power density profile from such an optical system can be seen in figure 4.4. From this it can be seen that a high power density is achieved at the centre of the line, fading to almost zero at the extremities of the line.

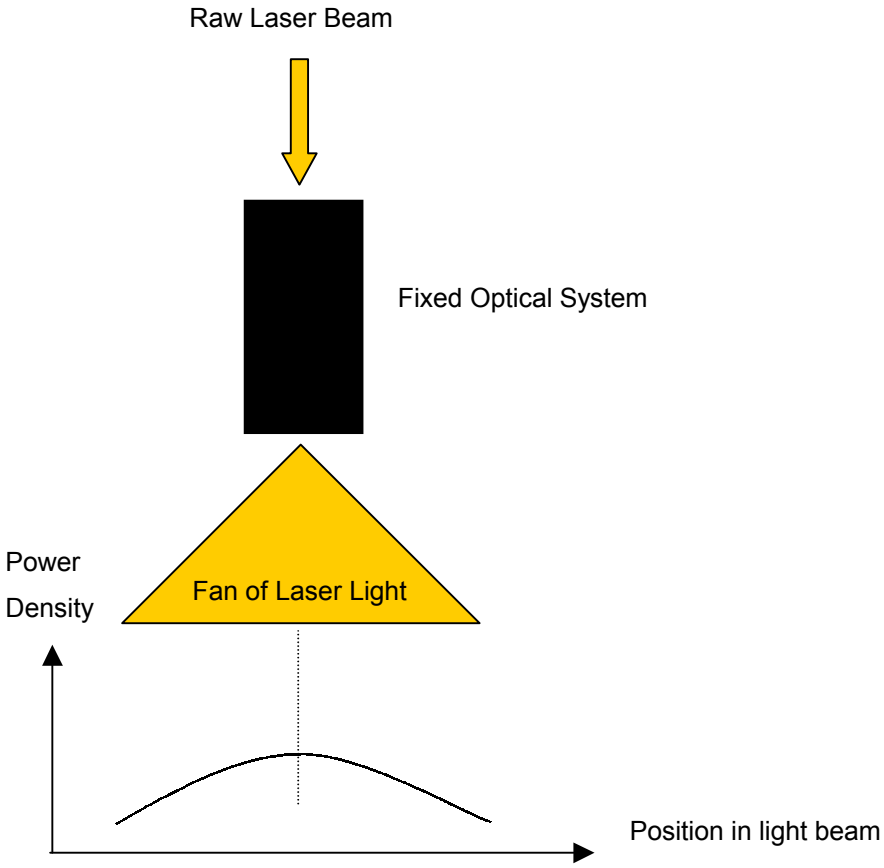

Fig. 4.4: Linear beam profile from fixed optical arrangement.

Unfortunately, the egg's surface curves away from the laser source at its extremities. Hence, to obtain a uniform energy density at all points of the egg's surface, additional laser power needs to be applied to the egg at its upper and lower points. The fixed optical method works against the natural curvature of the egg at this point, compounding the problem, and was hence discounted.

The alternative method utilised mechanical manipulation of the beam. This essentially involves firing a laser beam onto a movable mirror than can be rotated about a single axis so describing a line of laser light in a single plane.

The mirror's movement could be achieved using a galvanometer, where a computer system could accurately position the laser beam at any point but would involve significant processing power and increased cost, or a free running resonant scanner.

A resonant scanner is effectively a mirror mounted on a tuned spring mechanism with two driving solenoids and a feedback coil monitoring the position of the mirror. The two solenoids are driven via a small, low cost control board, which also monitors the position of the mirror for closed loop control.

In operation, the mirrors resonate in simple harmonic motion about a single plane at approximately 300 Hz. The frequency being dependent upon the natural resonant frequency of the mechanical system, being influenced by the properties of the spring and the physical mass of the mirror attached to the spring. For this particular system, mirrors of 18 mm diameter and 3 mm thickness were used, coated especially for the CO_2 laser wavelength.

With a laser beam incident on the resonant scanner, the scanner would describe a single line in simple harmonic motion, slowing down at the extremities of the lines, stopping, changing direction, and accelerating.

The resonant scanner system had the advantage of being low cost and requiring little computer intervention to perform their task. The natural profile of the resonant scanners also helped compensate for the profile of the egg. As the resonant scanner approaches the end of its scan, it decelerates, hence allowing the laser source to dwell on any particular point of the egg for longer, so imparting more energy. This occurs at approximately the same location where the curvature of the egg's increase.

A complete system costing approximately £300 could be realised for providing a line of laser light with a resonant scanner. This option was considerably cheaper than that of the computer controlled galvanometer.

4.4.2 Tracking Galvanometer Concept

In the simple case outlined above, a fixed line of laser light is aimed at a passing egg to irradiate half of the egg. Due to the standard spacing of the eggs in a tray (80.5 mm) and the typical egg diameter of approximately 45 mm, there exists a *mark to space* ratio of nearly 1:1. Hence, approximately 50 % of the available laser power is not being efficiently utilised.

Moreover, at the centre position of the egg, the laser beam is incident at right angles to the egg. As the incident beam impinges on either side of the egg, the incident angles become progressively acute. The effect of these grazing incidences at the extremities of the egg serve to reduce the energy density of the incident beam. This can be seen in figure 4.5 with the fixed laser beam positions of 1, 2 and 3. Beam 2 has a normal incidence, but beams 1 and 3 have severe grazing incidences.

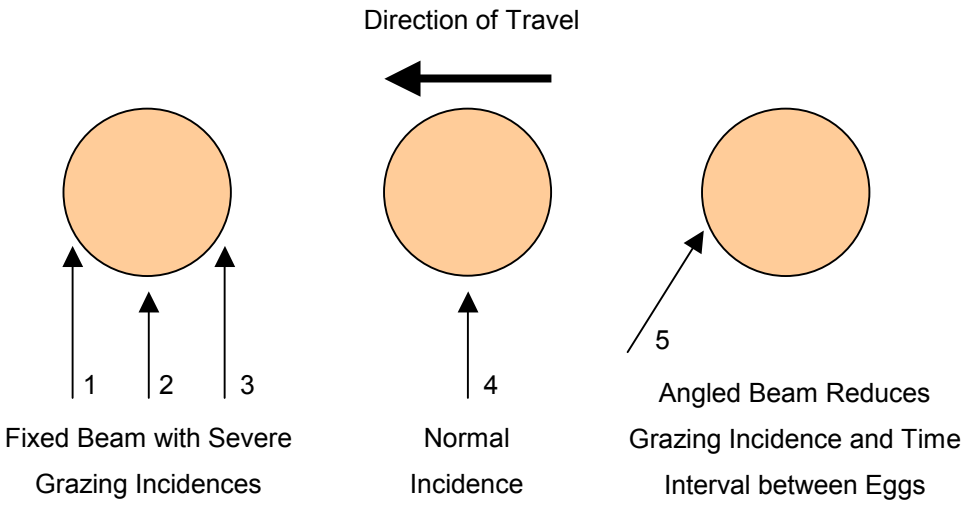

Fig. 4.5: Compensation of grazing incidences and mark to space ratio.

This effect can be compensated for in two ways. Firstly, by moving the angle of the incident beam to reduce the severity of the effect. Secondly, to allow the incident laser beam to dwell on these areas for a longer period to allow a similar total energy transfer per unit area. This can be visualised by the angle of the beam 5 in figure 4.5.

This approach is best adopted using a computer-controlled galvanometer. A galvanometer being a mirror fixed onto an electrical device similar to an ammeter. When an electrical current is applied to the galvanometer, the mirror will change position according to the magnitude of the current applied.

In the initial prototype system, the ***tracking galvanometer*** tracked at a fixed angular velocity, set via a 10 kΩ linear potentiometer, hence not allowing any automatic compensation for variations in line speed, or for modification of the galvanometer's scan profile, (to compensate for the natural shape of the egg), or varying egg widths. These features however are included in the final research machine. The galvanometers were triggered by a 5 V pulse from the CPU (Central Processing Unit), scanning at the pre-set rate.

4.5 Specific Design Parameters

To achieve the desired aims of the concept design outlined above, a number of key design considerations had to be addressed in detail. There are four critical system parameters that lead to the overall systems efficiency and control effectiveness.

- Resonant Scanner with Laser Power Modulation
- General Beam Manipulation
- Return Stroke Optimisation
- Tracking Galvanometer Optimisation

Before each of these are discussed in detail, the overall system design and its aims will first be discussed.

4.5.1 Outline System Design

The basic concept of the design was to take trays of contaminated eggs in on one side of the machine, process the eggs via laser and provide clean trays of treated eggs at the output. To achieve this, the machine had to be segregated into two halves, clean and contaminated. Once cleaned, the eggs could not be replaced in their original tray, but had to be stored in a fresh, uncontaminated tray. The original tray being ejected from the machine, ready to be repopulated with contaminated eggs for processing.

Being of a fragile nature, the eggs were picked up by vacuum rubber suction cups while being transported from one side of the machine to the other. This is a common approach to egg handling, with very few breakages occurring.

Once picked up from a tray of contaminated eggs, the eggs are transported through a central processing point, which laser processes the sides and undersides of the eggs. During this process, the eggs are held from above via the rubber suction cups. Figure 4.6 shows the central processing point of trial machine with the rubber suction cups.

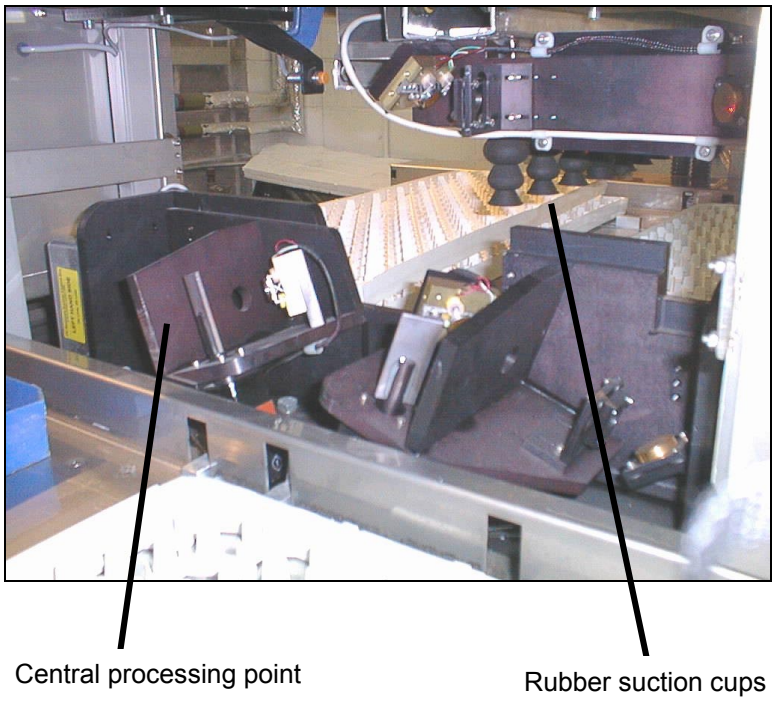

Fig. 4.6: Central processing point of trial machine.

The treated eggs are then deposited in the clean egg trays. However, the tops of the eggs have been obscured by the suction cups, and are hence processed on the return stroke of the machine, while the moving carriage transverses back to its home position to collect a fresh row of eggs for processing.

4.5.2 Resonant Scanner with Laser Power Modulation

To prevent damage to the egg's cuticle at the extremities of the resonant scanner's stroke (where the resonant scanner slows down, stops, and then reverses direction), the laser power must be reduced prior to this point, and subsequently restored to its previous level.

Unfortunately, all three of the resonant scanners used in the design were "*free running*", and hence not in synchronisation with each other. This meant that while one resonant scanner was at the top of its stroke, the others might be only half way through a stroke for example. Three resonant scanners were used in total, one for each side of the egg, and one for the top of the egg.

To compensate for this, a separate control system for each laser tube and for each of the resonant scanners would be required. This would also remove the possibility of using a single higher powered laser for the project, as two separately controlled laser beam sources are a requirement of this approach, using the resonant scanners, with modified laser power during the stroke. If however, the laser power could be left at a fixed level, then a single laser source would be an option, or if the resonant scanners could be synchronised.

One option considered for this approach was to have the main computer system monitoring the position of the scanners via separate A/D (Analogue to Digital) converters, and two individual D/A (Digital to Analogue) converters to control the laser's power. This could have been done, but was very complex and would have potentially slowed the computer response time, which might have been needed at a later date, or necessitate a more powerful (costly) computer control system, with associated longer development times.

The adopted approach was to use discrete hardware controllers that continuously monitored the scanner's positions. When a scanner's position passed a pre-set threshold, the laser power was reduced to a lower, pre-defined value. Hence, the laser systems had to have three power settings, one for full power, and two at reduced power levels for the upper and lower strokes of the resonant scanners

corresponding to the top and bottom of the eggs. Figure 4.7 shows the resonant scanner waveform with the upper and lower thresholds highlighted.

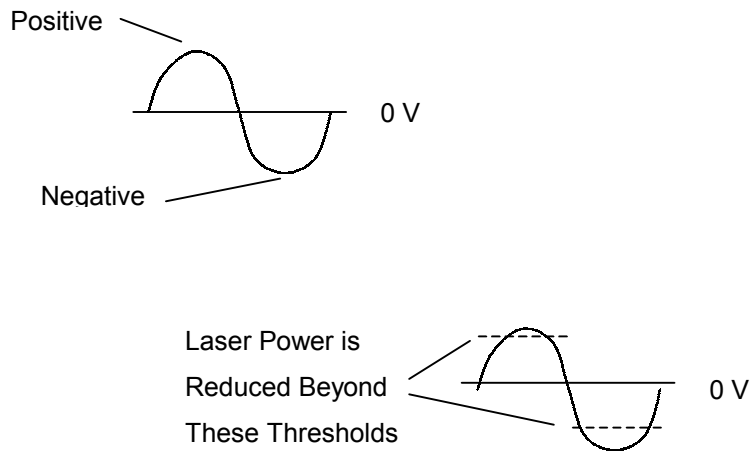

Fig. 4.7: Laser power modulation for resonant scanner waveform.

The upper stroke needed to have a greater reduction in power, as the full grazing incidence had not been reached, as this was obscured by the rubber suction cup holding the egg. The lower stroke only required minimal reduction, as at the very lowest point of the stroke the most severe grazing incidence was encountered.

Figure 4.8 shows a screen shot taken from an oscilloscope monitoring the electronic hardware used to achieve this task. The screen shot (figure 4.8) shows the signal from the feedback coil of the free running resonant scanner on the upper trace and the output stroke pulse on the lower trace, corresponding to a predetermined threshold being reached on the negative stroke of the resonant scanner. The output pulse is subsequently used to reduce the laser power at this point.

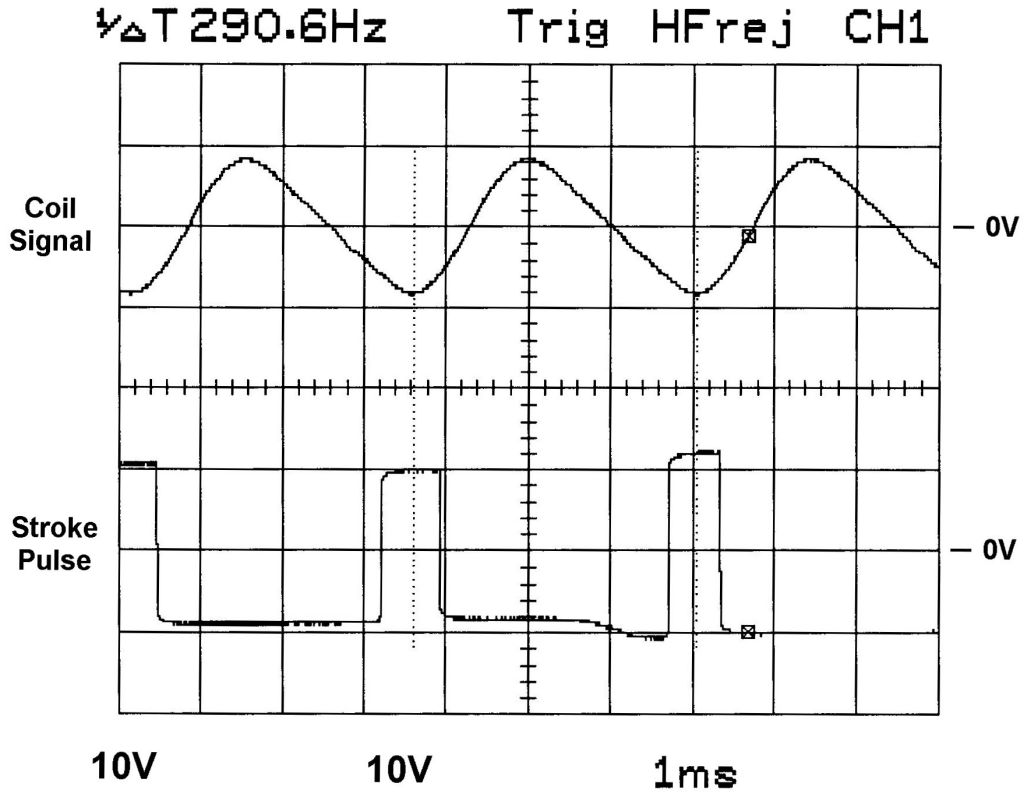

Fig. 4.8: Signal derivation for laser power.

This in theory was a perfectly valid system with the designed electronics performing exactly as predicted in the laboratory. However, when translated onto the research machine no change in laser power was noted. This was due in part to two reasons; firstly the optical response of the laser tube being too slow and secondly the frequency of the resonant scanner being too high in comparison to the pulse modulation frequency of the laser.

The lasers used for this project were Synrad Series 57-1. Table 4.1 shows the technical specification for this laser [50].

Table 4.1: Synrad laser series 57-1 specifications.	
Laser characteristic	**Value**
Wavelength (nominal)	10.6 μm
Power †	120 W
Power (minimum)	90 W
Power stability (30 s warm up)	± 8 %
Mode quality (TEM_{00} equivalent)	95 %
Beam diameter †	4 mm
Beam divergence †	3.5 mR
Polarisation	Linear
Modulation capability (optical)	4 kHz
Cooling water	2 GPM
Electrical input (28 VDC to 32 VDC)	65 A
Weight (laser head)	29 lbs
Weight (RF power supply)	22 lbs
Electrical control	TTL input (+3.5 V) to 10 kHz

† Typical.

The two parameters of importance from Table 4.1 for this problem are the optical modulation capability of 4 kHz and the electrical modulation capability of 10 kHz. From this it can immediately be seen that the system is capable of being electrically modulated far faster than the laser can optically respond.

With the resonant scanner's frequency of oscillation being approximately 300 Hz, this translates to a period of 3.3 ms per cycle, or 1.7 ms for the duration of a single stroke. The lasers were being driven by the manufacturer's recommended modulation frequency of 5 kHz. This frequency equates to a period of 0.2 ms.

Hence, during the duration of a single stroke of the resonant scanner (1.7 ms), 8.5 laser pulses will be seen. From this it can be seen that at the dwell points at either end of the resonant scanner's stroke only one or two laser pulses will be affected by the required downturn in power. Due to the optical response of the laser tube, the effect on a single pulse is likely to be minimal. Furthermore, if the laser pulse does respond to a degree and become reduced in duration, the laser will still potentially be on for a short period at the dwell point, so still causing damage.

Figure 4.9 shows the thermal response of the system's return stroke captured on thermally sensitive paper. This clearly shows the dwell points and increased power density at the end of each stroke. The pulse modulation of the laser can also be clearly seen on each stroke. The length of the strokes in figure 4.9 are 40 mm, each separated by 1 mm.

Fig. 4.9: Resonant scanner waveform on thermally sensitive paper.

As the lasers proved incapable of responding fast enough to the required modulation of laser power at the extremities of the resonant scanner's stroke, an alternative method of preventing damage due to the scanner's dwell points was required. The final solution for this was the adoption of simple masks placed in the optical path of the resonant scanners, masking off the upper and lower dwell points of the scanner's cycle. These masks were made from aluminium and acted as heat dumps, absorbing the excess energy deposited by the lasers at the dwell points of the resonant scanners.

The adoption of this method would now permit the use of a single laser system with associated beam splitting optics instead of the two laser used for the initial approach. Both approaches are still equally valid, with the final choice now being one of cost and ease of the engineering integration.

4.5.3 General Beam Manipulation

The complete laser beam delivery system required 21 individual optics in total, as shown in figure 4.10. The distances travelled by the laser beams were quite significant requiring precise optical alignment. The use of a CAD (Computer Aided Design) system proved very useful in simulating different scenarios of optical arrangements prior to the final designs being chosen [†]. To achieve the design requirements, a number of key issues had to be overcome. These will now be discussed individually.

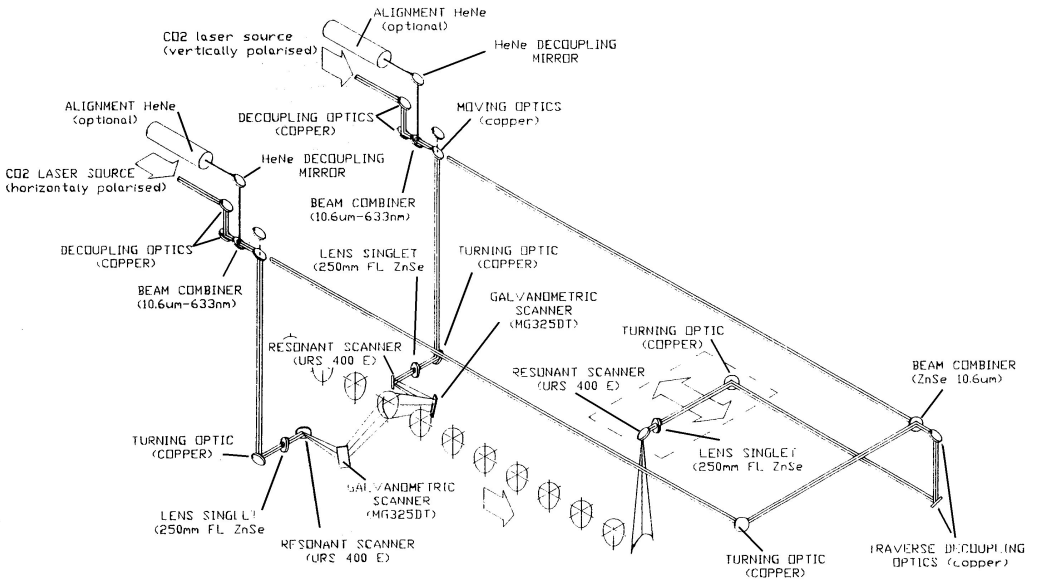

Fig. 4.10: Optical layout of trial machine.

[†] All project mechanical design was conducted by Chris Williams of ICN.

4.5.3.1 Beam Combining

Two individual laser sources were used to process each side of the eggs via the resonant scanners during the first stage of the processing. Once the eggs have been deposited in the clean tray, the transport carriage returns to pickup the next row of dirty eggs. However, the tops of the eggs obscured by the holding rubber suction cups have not yet been processed. These are to be processed on the return stroke of the transport carriage with the third resonant scanner.

At this point however both laser sources will be available to scan a single line of laser light. By combining these two laser sources together twice the laser power of the first processing cycle can be obtained and can hence process the return stroke at twice the speed of the initial cycle if desired. This maximises the use of the available laser power and could help to speedup the overall processing time for a complete cycle.

Eggs are normally stored in their trays with their narrower ends pointing down, that is their broader ends will be the ends that are held by the rubber suction cups and consequently could receive additional laser power on the return stroke. This is advantageous for this application however, as it has been shown from the previous discussion in section 4.2.1 (Egg Shell) that the broader end of the egg is the end that contains the greatest number of pores. Moreover, as bacteria have a known affinity for micro-cavities they are more likely to be found in higher concentrations in the egg's pores. Thus, the additional available laser power for the return stroke may prove to be extremely advantageous to the overall efficacy of the system's performance.

To combine the two CO_2 laser beams, the two beams have to be of opposing polarisation. As standard, lasers tend to be of fixed polarisations. However, due to the requirement of combining the two laser beams on the return stroke, a custom optical system had to be designed to rotate the polarisation of one of the laser beams, before the two beams could be combined with a single optic.

Figure 4.11 shows the optical arrangement used to combine the two laser beams, providing the correct incident angles for both laser beams onto the combining optic. The paths travelled by the laser beams in figure 4.11 are shown in blue.

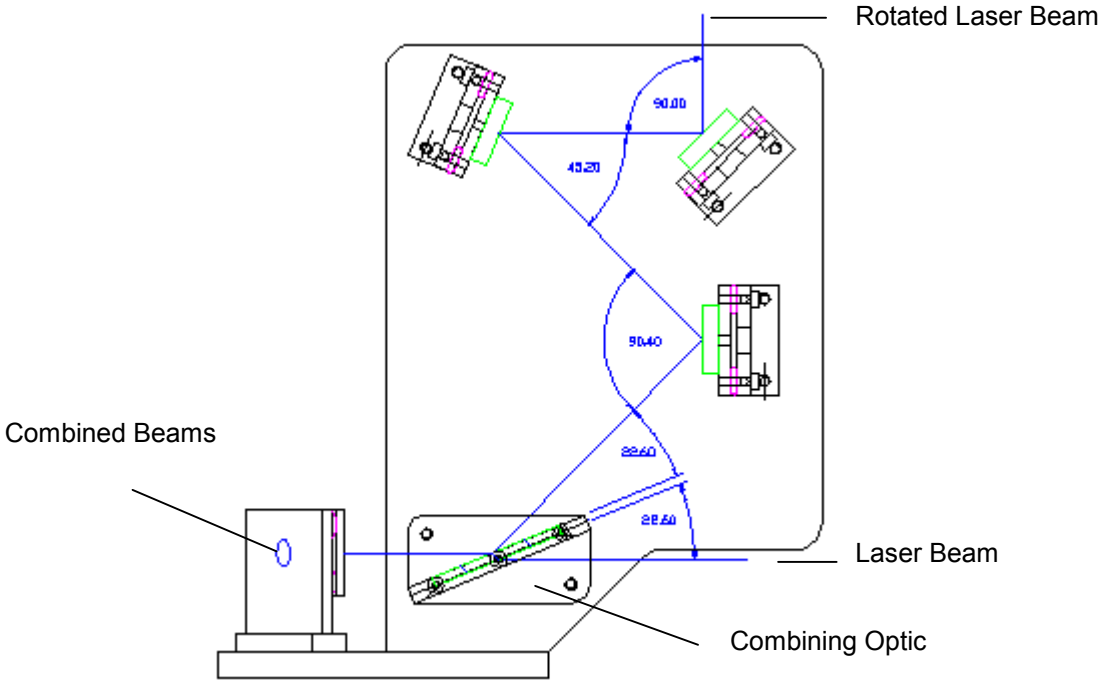

Fig. 4.11: Combining of two polarised laser beams.

4.5.3.2 Laser Decoupling

The positional accuracy of the laser's pointing stability was critical to this application due to the long beam paths involved in the system. Any error at the laser source would be magnified by the end of the beam path, giving rise to potential beam alignment problems.

The laser manufacturers did not have any data available quantifying the angular and positional pointing stability / accuracy of their CO_2 lasers. With this in mind, it was decided to incorporate full optical decoupling at each laser source. This gave the flexibility for complete and accurate alignment of each laser system, should their emitted beams be variable in position from laser to laser. Figure 4.12

shows the arrangement of this optical de-coupling, with the laser light paths shown in light blue.

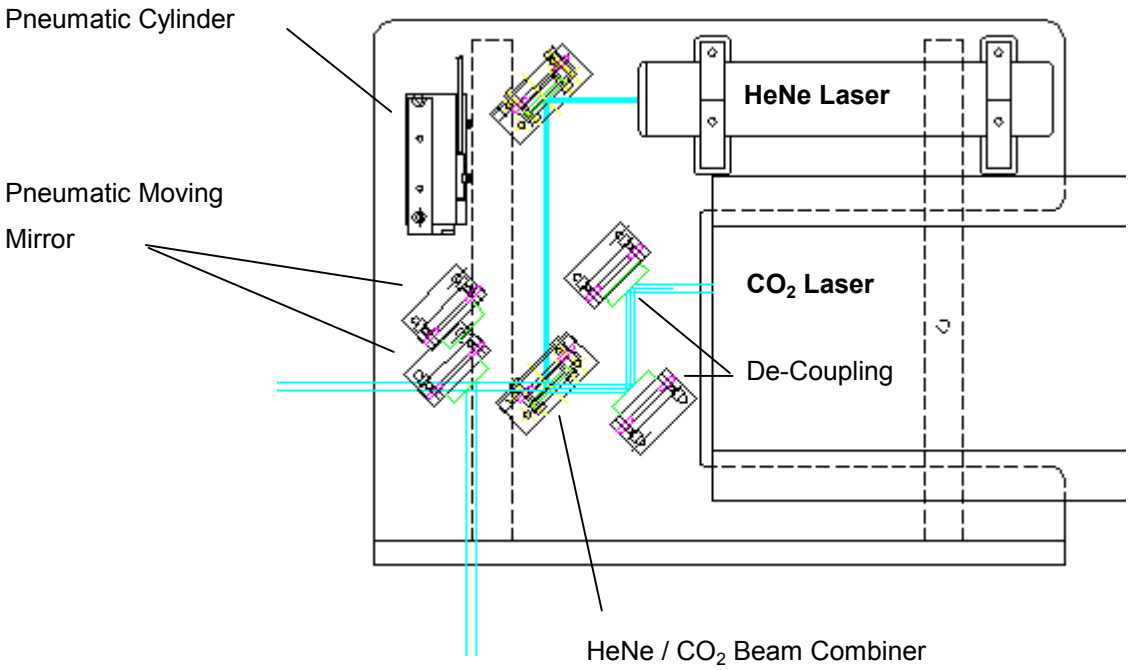

Fig. 4.12: Laser de-coupling, HeNe beam and pneumatic mirrors.

4.5.3.3 Focus Adjustment

A small amount of focus adjustment will be available, allowing the focal plane to be matched to the egg's surface or centreline (this only applies when the beam is normal to the egg surface). The focusing optic used had a focal length of 250 mm, which provided sufficient depth of focus to compensate for dynamic focal variations occurring due to the passing of the egg's profile. Being fixed, this was not easily modified to accommodate different focal length lenses. The optimum configuration of the focusing arrangement was investigated on the experimental prototype prior to final design.

4.5.4 Return Stroke Optimisation

Once the two laser beams have been combined, they must be directed at a moving optic attached to the traversing carriage. Beam alignment at this point was critical and optic stability crucial to the successful coverage of the egg tops.

At this stage of the beam delivery, any angular misalignment of the primary beam path will be greatly amplified and therefore any optic mounts or associated engineering must be of a substantial nature, coupled with fine adjustment resolution. This is why the positional accuracy of the laser tubes was of such importance.

As previously mentioned the beam path became quite complex and therefore the commissioning and alignment of the various optical components became quite onerous, and indeed hazardous. Hence, it was decided to include visible lasers (HeNe) to aid this task. These can be seen in figure 4.12.

4.5.4.1 Movement of Mirrors

To enable the two laser beams to be combined, (allowing the top of the eggs to be cleaned on the return stroke of the system), two mirrors, one on either side of the machine, had to be moved accurately and repeatably, in and out of position at each stroke of the machine. The optical arrangement for this can be seen in figure 4.12, which shows the position of the pneumatic cylinder and the two positions of the mirror attached to the cylinder with the two resulting beam paths.

The pneumatic cylinders were triggered by computer control, with pneumatic damping valves allowing the speed of operation to be adjusted. If the cylinders were to operate too fast, the mirror mounts attached to the cylinders would receive mechanical shocks, which may knock the system alignment out. Hence, the choice of pneumatic cylinders with damping capability was of particular importance for this application.

Figure 4.13 shows the schematic layout of the pneumatic cylinders controlled via a single valve, with the two damping valves per cylinder, giving complete control of the cylinder response.

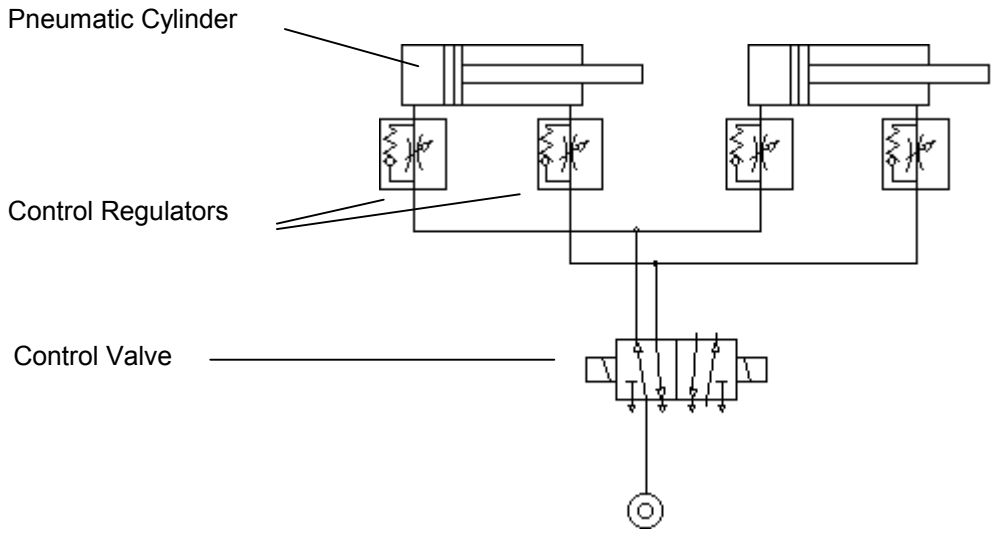

Fig. 4.13: Pneumatic circuit for mirror control.

Finally, as a safety requirement, each mirror had a micro-switch sensor at each extremity of movement. This gave positive feedback to the computer system, informing it that both mirrors were in their correct positions and that it was safe to continue.

4.5.5 Tracking Mirror Optimisation

4.5.5.1 Uniformity of Laser Coverage

The optimum position of the galvanometers relative to the eggs was established from empirical trials conducted on the initial experimental system. This was achieved by trials using "*blue hypercolour eggs*" and monitoring the uniformity of coverage as a result of the focal variations and variable grazing incidences, experienced as a direct result of the galvanometer positioning.

The blue hypercolour was a thermally sensitive dye that when cold (below 22 ºC) had a deep blue colour, but when heated became paler until turning completely white above 31 ºC. When the dye cooled down again, it would revert to its original deep blue colour. However, the typical change in colour from blue to white and back again would only last for approximately one second. Thus china eggs coated with this dye were videoed being laser processed for later analysis for uniformity of laser coverage The dye used was "Variotherm Turquoise 25C" supplied by Magna Colours Limited.

An alternative and more permanent system was later adopted for the analysis of uniformity of laser coverage. This system used thermally sensitive paper, but care had to be taken with the laser power to prevent ignition of the paper. Unfortunately, despite many efforts, the specific thermal response profile of this paper could not be obtained. Hence, the resulting images could only be used for qualitative analysis of laser coverage rather than any quantitative analysis. Section 4.5.2 (Resonant Scanner with Laser Power Modulation) shows one such example of a thermally sensitive paper scan pattern from the resonant scanner on the return stroke.

4.5.5.2 Galvanometer Positioning

Due to the complexities of utilising the same laser source for top and side coverage with the complex beam paths involved, once set, the galvanometer position will not be adjustable.

Another aspect of the galvanometer positioning was with the galvanometers close proximity to the eggs, with a possibility of the lower tail of the beam passing under the egg and reflecting off the opposing galvanometer mirror and into the suction cup holding the egg. This was alleviated by angling the galvanometers up at the egg from beneath at an angle of 30°. This angle helps to both scan the bottom of the egg, which otherwise would not get covered, and prevent the opposing scanner from causing extraneous damage. The addition of the laser

masks described in section 4.5.2 (Resonant Scanner with Laser Power Modulation) also helps to reduce the extraneous damage. This layout can be seen in figure 4.14.

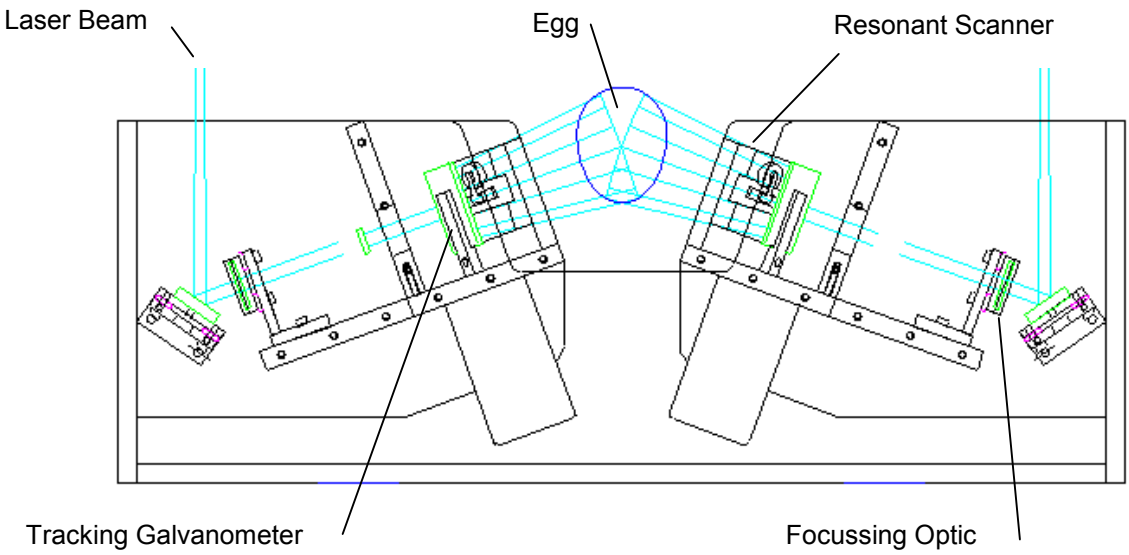

Fig. 4.14: Mechanical layout of tracking galvanometers.

4.5.5.3 Galvanometer Tracking

By increasing the tracking rate of the galvanometer at the leading and trailing edges, extra laser power can be imparted to these areas to compensate for the severe grazing incidences. The potential for actual egg size to be detected, with the galvanometer tracking the egg to a much closer tolerance thus improving the efficacy of the system also exists with this system.

Eggs are held at 80.5 mm centres, with the eggs being a maximum of 45 mm diameter, they therefore have minimum spaces between eggs of 35 mm. This represents a worst-case egg-space ratio of 60 % to 40%. Utilising the galvanometer tracking mirror, and allowing for a 45 mm diameter egg, the beam can be redirected onto the egg during the spaces, thus re-utilising approximately

35 % of the available laser power. During the course of this redirection of laser power, the incident angle of the beam, to the critical leading/trailing edges of the egg are also improved. Figure 4.15 shows the configuration of the resonant and galvonometric scanners in the trial machine.

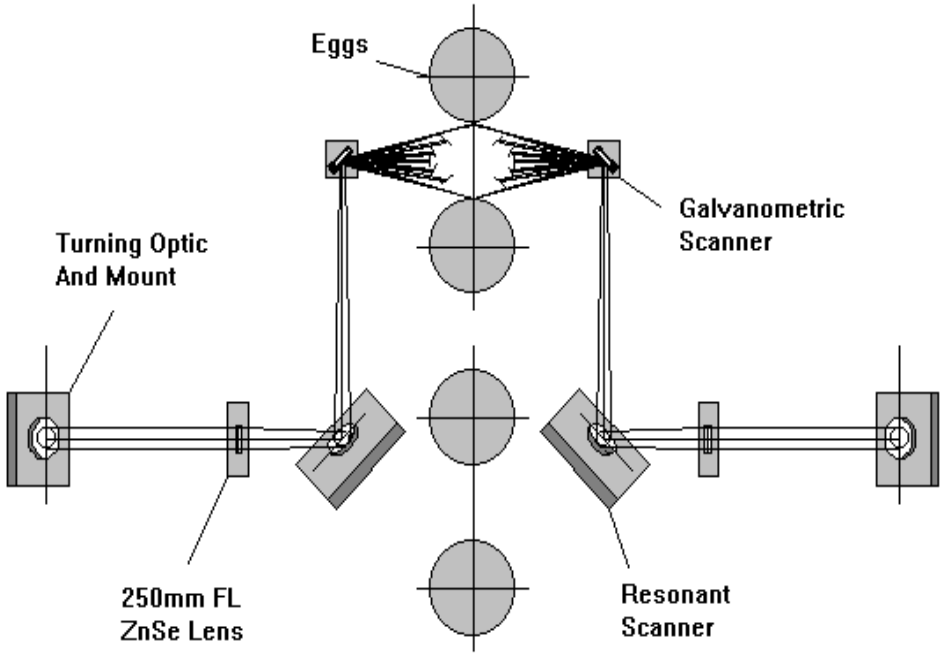

Fig. 4.15: Scanning configuration of eggs.

To allow flexibility it was desirable to be able to vary the scan rate (allowing tracking of fluctuations in the line speed, or different line speeds), and indeed, to change the scan profile, to more accurately match that of the egg's shape.

It was felt that the best way to proceed was to have a selection of pre-set profiles stored in non-volatile memory that could be output to a D/A (Digital to Analogue) converter at every pulse from a system Shaft Encoder. While the shaft encoder was not a very high resolution (1 mm = 0.884 pulses), the egg spacings were 80.5 mm and the egg widths 45 mm (maximum), if the galvanometer tracked each egg for 80.5 mm, this meant that approximately 91 pulses would be received from the encoder, compressed into a 45 mm spacing of the egg. That is, an increase of

the system's resolution from 1 mm to 0.505 mm, which in comparison to the laser beam's focused spot diameter of 0.675 mm, is not significant.

Figure 4.16 shows a computer-generated linear waveform matching that of the prototype system. The second waveform above shows a computer-generated signal that adjusts the tracking profile to compensate for the grazing incidences on the leading and trailing edges of the egg.

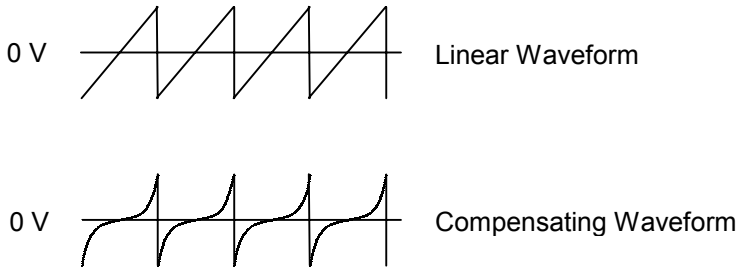

Fig. 4.16: Tracking mirror scan profiles.

Figure 4.17 shows the compensating waveform of figure 4.16 in relation to an egg.

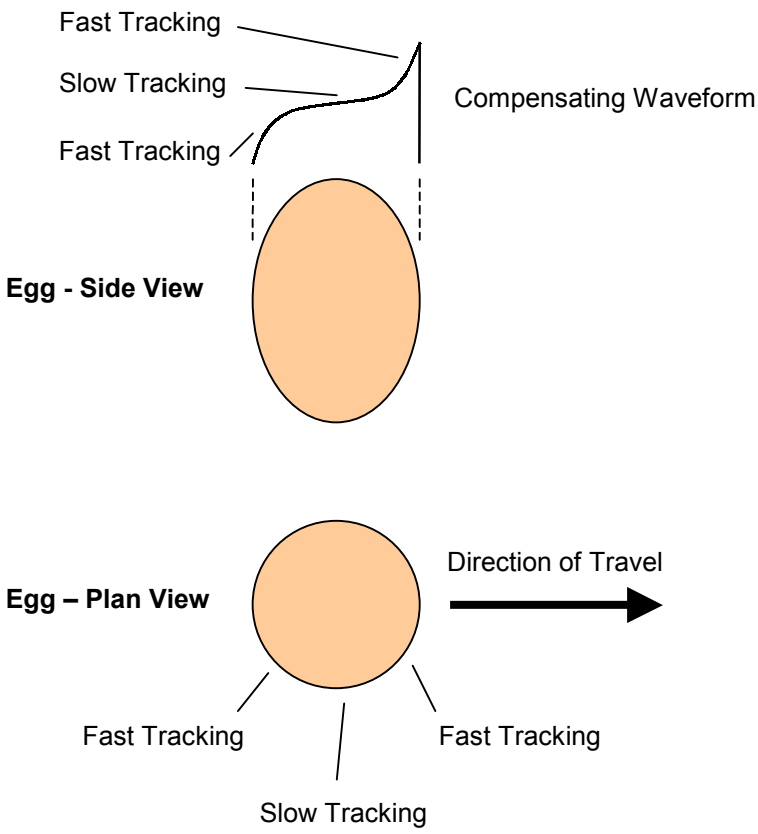

Fig. 4.17: Scan compensation to match egg profile.

4.6 General Construction

The basic mechanics of the research machine were fabricated from 316-grade stainless steel (food grade) by a Dutch company - Staalkat. Staalkat's primary business was the design and manufacture of commercial egg handling and processing equipment, making themselves ideally suited to the design and manufacture of the egg handling requirements of this project. The basic egg transport section of the machine designed by Staalkat could then be populated with the lasers, optics and control systems required to process the eggs.

With the aforementioned control sub assemblies designed and tested, the relevant assemblies were mounted into a control rack and connected to their relevant sub systems. The electronic modules were housed in two 19" rack unit(s) underneath the main machine (figure 4.18). The six DC power supply units required for the CO_2 lasers (three for each laser) were also housed in these spaces.

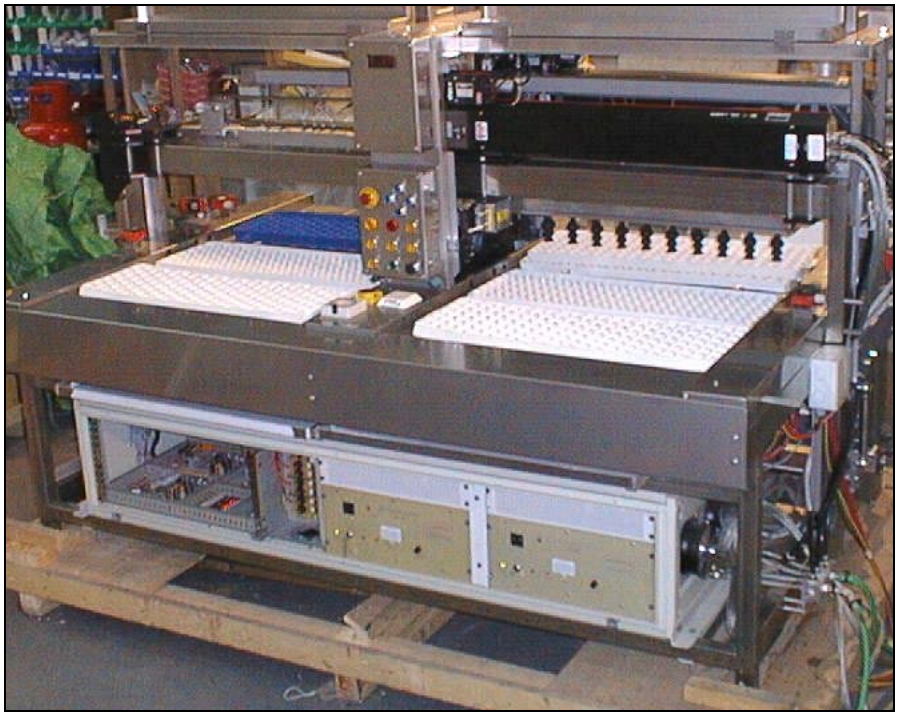

Fig. 4.18: Experimental machine.

The two lasers were wired in to the DC supplies, with the associated safety control relays and contactors being held in a separate control rack. The two racks being used to separate the high power side of the system from the low power control systems - which are inherently more vulnerable to noise interference.

The CO_2 lasers were water-cooled and required plumbing into a water feed capable of supplying 2 gallons per minute.

The final mechanical elements to be installed were the system's optics. All the optics needed careful alignment, with the aid of visible HeNe lasers co-aligned with invisible CO_2 lasers.

4.7 System Control

The complete software control system (see Appendix 3: Research System Control Program) was the last element to be completed in the system design. This was dependant on all of the above sub sections being in place and tested.

Various modules of the software were developed in the above sections during testing with these being reused in the final code, once tested and debugged. However, the final software coding and testing could only begin once all of the above had been completed. Much of this was of an iterative nature, while the machine's control protocol was debugged and fine-tuned. The software was written in 'C' [51] in a structured and modular format for the target embedded microprocessor, an 8051 derivative, the 80C652 [52].

In order to control all of the variable parameters on the trial machine a piece of remote control software was written in Microsoft's Visual Basic (see Appendix 4: Research System Remote Control Program). This software was designed to run on a laptop PC connected to the research machine through an RS-232 serial link at 9 600 baud via an easily accessible port located at the front of the machine for this purpose. With this software, every aspect of the machine could be both controlled and monitored. Figure 4.19 shows a screen shot of this software.

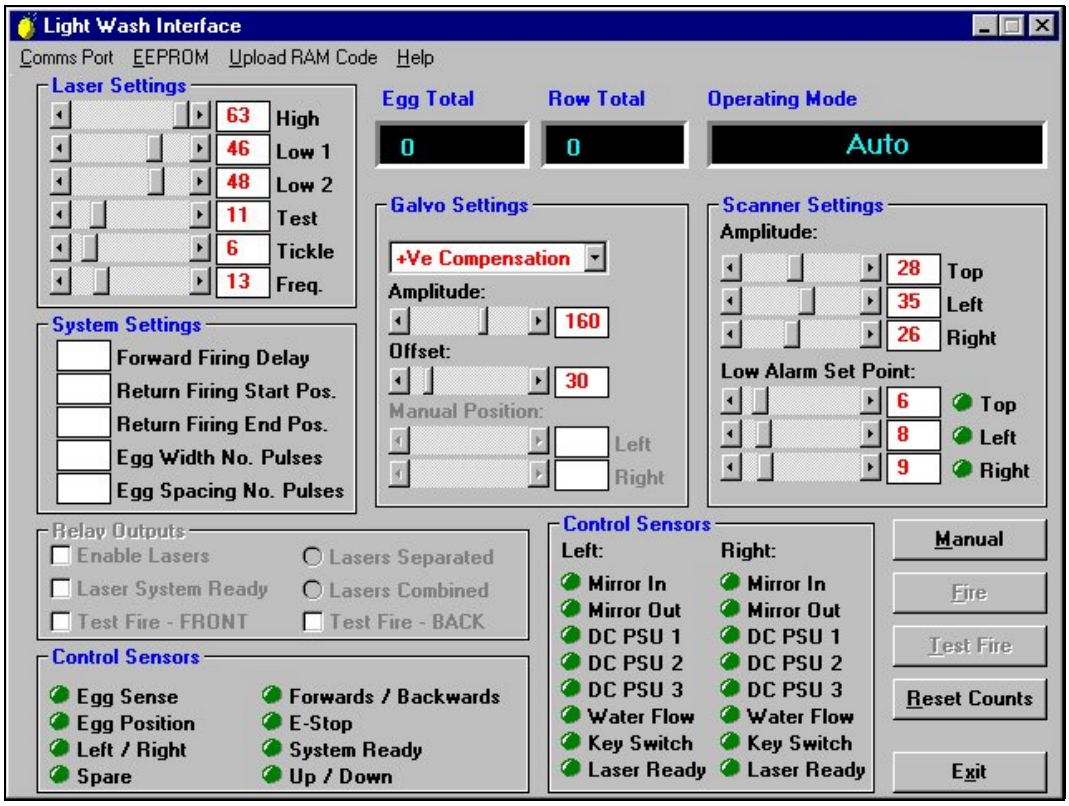

Fig. 4.19: Screen shot of trial machine's remote control software.

4.8 Summary

This chapter has described the philosophy behind the experimental system's design. With the system being designed to handle chicken eggs for breeding stock, the design requirements for these eggs were analysed first.

The egg's shells are approximately 0.3 mm thick [46] and are made from calcium carbonate ($CaCO_3$), but are brittle in nature [47]. The surface of the egg shell is covered in pores of 10 µm to 70 µm in diameter, with these pores being predominantly found at the broader end of the egg. The outside of the egg's shell is covered by a semi-permeable membrane (cuticle) offering the egg some natural protection against bacteria. The egg white and yolk will act as a thermal heat-sink during laser processing, conducting heat away from the egg's surface.

The average chicken egg is 45 mm in diameter at its widest point and 60 mm tall. In a commercial hatchery, they are stored in trays of 12 rows, each row containing 11 eggs. The eggs are at spacings of 80.5 mm between centres in the rows and are stored in the trays with their broad end facing up. To compensate for the delicate nature of the eggs, these are moved from their trays using rubber vacuum suction cups to prevent damage to the egg shell or cuticle.

To process a three-dimensional object such as an egg the natural spot generated by a laser first had to be manipulated into a line of laser light to help in the coverage of the egg in real time. A fixed optical method for this was evaluated, but proved prohibitively expensive (circa £6 000 per optic) and gave a poor energy density profile along the line's axis. The preferred approach was to use a resonant scanner running at 300 Hz to describe a line of light. This method cost only £300 per resonant scanner.

However, while the energy density profile of the line of laser light generated by this system helped to compensate for the natural curvature of the egg, other problems were introduced. At the ends of each stroke of the resonant scanner damage to the egg's cuticle was caused. To alleviate this a method of reducing the laser power at this point was required. Unfortunately, the chosen laser tubes could not respond to the required reduction in power fast enough. Hence, the adopted approach was to physically mask off the laser at the end of each stroke.

With the egg spacings of 80.5 mm and typical widths of 45 mm there exists a 60:40 egg-space ratio. Using tracking galvonometers to redirect the available laser power during the gaps on to the eggs, an increase in the efficient use of laser power of 35 % can be achieved. This method also reduced the grazing incidences on the eggs and improved uniformity of coverage.

The final optical train for the system required 21 individual optics. These were used for the general beam delivery as well as more specific tasks such as combining the two individual laser beams for the return stroke of the system. The combination of the two beams required the rotation of the polarisation of one laser

beam prior to the merging of the two beams. To combine the two laser beams two other mirrors had to be moved into position using damped pneumatic cylinders.

The mechanics of the experimental machine were constructed from 316-grade stainless steel (food grade). Trays of contaminated eggs were taken in on one side of the machine, passed through the central laser processing section of the machine with the clean eggs being deposited on to a clean tray on the opposite side of the machine.

On to the basic mechanical handling of the machine were added the lasers and their optical and electronic control systems. The primary control system for the machine was written in 'C' [51], together with an additional remote control program written in Microsoft's Visual Basic. This enabled a laptop computer complete control and access to every system parameter during the commissioning and testing of the machine.

5 : EXPERIMENTAL ANALYSIS

5.1 Introduction

Having completed a thorough study into the physiology of micro-organisms and the physical parameters that can lead to their sterilisation, attention was then turned to the optimal choice of laser for a commercial scale system. Through the body of existing research conducted in laboratory trials on a variety of lasers and a detailed analysis of the physical and physiological parameters involved, the CO_2 laser became the overwhelming choice for such a system.

With the choice of laser firmly established it then remained to develop a system that would be capable of delivering the required laser power in as uniform, efficient and controllable a way as possible. This required an iterative process of experimental procedures and theoretical analysis before the final design could be defined.

Prior to the manufacture of the trial machine a prototype laboratory mock-up was made incorporating all of the machine's key design features such as the resonant scanners and tracking galvanometers together with the proposed optical arrangement and laser power.

Following on from the successful trials of the aforementioned prototype system, the full trial system was then designed and manufactured.

.

5.2 Methodology

The experimental methodologies adopted for the initial prototype trials can be broadly categorised into two sections. The first being the preparation and subsequent analysis of the microbiological samples used for the tests. The second being the physical experimental configuration and operation of the trials.

These two methodologies will now be discussed individually.

5.2.1 Microbiological Methodology

In order to measure the effectiveness of a sterilisation method, a reliable and quantifiable method of measuring the number of micro-organisms on a given substrate is required.

As a reference for any tests, control samples will also be required. For example, while testing for the effectiveness of sterilisation, one set of contaminated samples should not be subject to sterilisation, this will form the control sample. The control sample will provide a base line for the comparison with measured results of sterilisation.

5.2.1.1 Microbial Contamination

To pre-contaminate the sample eggs prior to the laser trials, standard chicken eggs are submersed and washed in a broth containing the desired contaminent(s). These eggs are then transported to the test site in suitably secure and sealed packaging clearly labelled as “biohazard”.

5.2.1.2 Micro-organism Measurement

The number of remaining micro-organisms on the treated eggs and control samples were counted using the total viable count (TVC) plate count method at ADAS' microbiology department, Wolverhampton. This method is based on *BS5763: Part 1 1991 – General guidance for the enumeration of micro-organisms colony count technique at 30 ºC* [53].

To extract the bacteria from the surface of the processed and control eggs, these eggs were washed in a sterile broth (a separate broth for each egg). A 1 ml quantity of each broth sample was then used to aseptically inoculate a labelled petri dish (a dish of approximately 10 cm diameter and 1.5 cm in depth, with a loose fitting lid).

Decimal dilutions were then prepared by using a fresh sterile pipette to transfer 1 ml of the initial inoculum (broth) into 9 ml of maximum recovery dilutent (MRD). This procedure was repeated for as many decimal dilutions as were required.

The dilutions were then mixed using a vortex mixer for 5 s to 10 s.

1 ml of each required dilution was used to aseptically inoculate a labelled petri dish. (Inoculation is the introduction of micro-organisms into or onto a sterile medium.)

15 ml of tempered plate count agar (PCA) (46 ºC ± 1 ºC) was then added to each petri dish, carefully mixed and allowed to set. The time elapsing between the addition of the MRD and contact with the PCA should not exceed 20 minutes. (Agar is a gelling agent, and a long chain polysaccharide, which provides food for the growing micro-organisms. [54])

The petri dishes were then inverted and transferred to an incubator at 30 ºC ± 1 ºC for 72 h ± 3 h.

Using colony-counting equipment, all plates containing between 15 and 300 colonies were counted. Dishes containing not more than 300 colonies at two consecutive dilutions were retained, at least one of which must contain 15 colonies.

The number of micro-organisms per gram (calculated by a weighted mean from dishes of more than one dilution) is now given by the following equation [53]. This equation is used in preference to a simple arithmetic mean of replicate dishes from a single dilution to reduce the standard error.

$$\text{Number of micro-organisms per gram} = \frac{\sum c}{(n_1 + 0.1n_2)d} \qquad \textit{(Equation 5.1)}$$

Where:
- $\sum c$ is sum of all colonies counted from all the dishes retained
- n_1 is number of dishes retained in the lowest (1^{st}) dilution
- n_2 is number of dishes retained in the next higher (2^{nd}) decimal dilution
- d is dilution factor corresponding to the lowest (1^{st}) dilution

For plates with 1 to 15 colonies, the colonies were counted and multiplied by the dilution factor.

5.2.2 Experimental Configuration

The laser power for the initial prototype system was fixed at 140 Watts and could not be readily altered by computer control. As such, at the end of the top stroke of the resonant scanner, excess energy was applied to the egg, causing cuticle damage. This had no detrimental effect to the kill levels of micro-organisms, but

is potentially detrimental to the eggs in question, as previously discussed. Hence, on the final machine, a means of preventing this damage was included via the use of masks.

The tracking galvanometers helped maximise the efficient use of the available laser power with a linear tracking waveform, but with further control of the tracking waveforms, additional benefits could be obtained in achieving uniform dose levels at the leading and trailing edges of the eggs.

The initial prototype system was implemented using a linear conveyor belt travelling at between 150 $mm \cdot s^{-1}$ and 75 $mm \cdot s^{-1}$ and carrying seven eggs at a time with spacings of 80.5 mm between centres. The laser used was a 140 W radio frequency (RF) excited CO_2 laser (supplied by Domino Laser Industries) and had an output beam diameter of 5 mm. The laser beam was focused by a 250 mm Zinc Selenide (ZnSe) focal length lens, mounted 35 mm prior to the resonant scanner. The resonant scanner was free running at 294 Hz and its amplitude set to describe a 50 mm line on the passing eggs. The resonant scanner was mounted prior to the galvonometer, which itself was angled at 30° from the horizontal, ensuring coverage of both the side and underneath of the passing eggs. Three passes were done for each row of eggs, one pass each for the front and rear sides of the eggs and one pass for the top. The focus point of the laser was set to be between the closest surface of the egg and the centre of the egg, (allowing the depth of focus to accommodate the natural curvature of the egg), for two sets of trials. A further three sets of trials were conducted with the focus point of the laser set to be at the closest surface of the egg.

5.3 Results

Under the above conditions, a 99.988 % (3.9 log) kill rate was achieved on 10 eggs laboratory contaminated with *Salmonella enteritidis* against 10 untreated control samples at a linear velocity of 100 $mm \cdot s^{-1}$.

Table 5.1 shows a summary of the results from these trials.

Table 5.1: Kill rates

Velocity ($mm \cdot s^{-1}$)	Focus	Kill Rate (%)	Kill Rate (log)
100	Mid-point	99.988	3.9
150	Mid-point	99.968	3.5
75	Closest Surface	98.994	2.0
100	Closest Surface	99.908	3.0
150	Closest Surface	99.157	2.1

5.4 Analysis

The above results clearly demonstrate that the proposed system is capable of achieving high kill rates in a commercial-scale system, compared to previous studies that have all been conducted in static laboratory conditions.

The trials conducted with the mid-point focus setting all gave higher kill rates than those of the closest surface focus setting. This was expected, as the energy density of the focused spot for the closest surface focus setting will greatly decrease at the edges of the egg where the egg's surface is furthest from the point of focus. The energy density also decreases naturally at this point due to the increased grazing incidence at these extremities. The increased reduction in energy density at the edges of the egg in these circumstances will lead directly to a reduced kill rate.

Furthermore, a slight increase in kill rate was noticed for the lower translation velocity of 100 $mm \cdot s^{-1}$ with the mid-point focus setting. This is as was expected, as the lower velocity would cause greater overlap between consecutive laser spots

so leading to an overall increase in imparted energy. However, this effect was not consistent with the differing translation velocities for the trials conducted on the eggs with the laser focus set to the egg's surface. To investigate this further, more trials need to be conducted to reduce the statistical variance of the samples, which is thought to be the cause for discrepancy from the expected results.

A discussion of the statistical techniques used to collate and process the collected data and a full analysis of the experimental system's technical performance follows below.

5.4.1 Statistical Analysis

To automate and control the collection of the trial data with subsequent statistical analysis, a computer software package called MINITAB was used. From these results the relative kill rates were then calculated.

5.4.1.1 Data Manipulation

In order to extrapolate meaningful results from collected data it is sometimes necessary to use statistical tools to manipulate the raw data into a more coherent and manageable form. For the above results one such technique was adopted, this was the *trimmed mean*. A full description of which follows below.

If a sample set contains some `outliers' in its top or bottom ends, the mean or average value is strongly influenced (becomes biased towards) the outliers. To alleviate this potential problem, a modified mean called the trimmed mean is used. An x % trimmed mean is such that both the top $^{x}/_{2}$ % and the bottom $^{x}/_{2}$ % of the observations are discarded and the mean is calculated for the remaining observations. A trimmed mean is obviously less susceptible to the effects of extreme observations than is the arithmetic mean. It is therefore less susceptible to

sampling fluctuation than the mean for extremely skewed distributions. All calculations performed using MINITAB for trimmed means were done using the 10 % trimmed mean $X_{tr\ (10)}$ (the mean calculated after omitting the smallest 5 % and the largest 5 % of the observations).

5.4.1.2 Kill Rate Calculations

In all of the kill rate calculations performed the trimmed mean values (as described above) were used. The trimmed mean value for the organism count of the 10 untreated contaminated control eggs was 2 731 250.

Table 5.2 summarises the kill rate calculations for each batch of 10 eggs treated per test.

Table 5.2: Kill rate calculations

Velocity ($mm \cdot s^{-1}$)	Focus	Trimmed Mean	Kill Rate (%)	Kill Rate (log)
100	Mid-point	334	99.988	3.9
150	Mid-point	873	99.968	3.5
75	Surface	27 483	98.994	2.0
100	Surface	2 508	99.908	3.0
150	Surface	23 036	99.157	2.1

5.4.2 System Analysis

The following discussion provides a full analysis of the prototype trial system. Table 5.3 summarises all of the system parameters used for the above experiments.

Table 5.3: Prototype system parameters.

Parameter	Value
ZnSe Lens focal length (*f*)	250 mm
Laser wavelength (λ)	10.6 μm
Laser beam diameter (d_L)	5 mm
Laser power	140 W
Resonant scanner frequency	294 Hz
Linear velocity	150 mm·s^{-1}
Scan height	50 mm

The diameter of the focused laser spot is given by [31]:

$$d_s = \frac{4\lambda f}{\pi d_L} \qquad \textit{(Equation 5.2)}$$

Where:
- f is the focal length of the focusing lens
- λ is the wavelength of the laser light
- d_L is the laser beam diameter
- d_s is the focused laser spot diameter

$$\therefore d_s = \frac{4 \times 10.6 \times 10^{-6} \times 250 \times 10^{-3}}{\pi \times 5 \times 10^{-3}} = 675\ \mu\text{m} = 0.675\ \text{mm}$$

Power density in Watts per square centimetre is now given by:

$$\text{Area of focused spot} = \pi r^2 = 0.358 \text{ mm}^2 = 3.58 \times 10^{-3} \text{ cm}^2$$

$$\therefore \text{Power density} = \frac{140}{3.58 \times 10^{-3}} = 39\,106 \text{ W} \cdot \text{cm}^{-2} = 39 \text{ kW} \cdot \text{cm}^{-2}$$

(Assuming uniform beam spatial profile, TEM_{00})

Energy density in Joules per square centimetre is now given by:

$$\text{Time taken for a single stroke of the resonant scanner} = \frac{294^{-1}}{2} = 1.7 \text{ ms}$$

$$\therefore \text{Average velocity} = \frac{50 \text{ mm}}{1.7 \times 10^{-3}} = 29\,412 \text{ mm} \cdot \text{s}^{-1}$$

$$\therefore \text{Average time taken to travel single spot diameter} = \frac{0.675}{29\,412} = 23 \text{ μs}$$

$$\therefore \text{ Average energy density} = 39 \text{ kW} \cdot \text{cm}^{-2} \times 23 \text{ μs} = 0.897 \text{ J} \cdot \text{cm}^{-2}$$

However, as the resonant scanner oscillates with simple harmonic motion the maximum velocity over the surface of the egg during the scanner's cycle will correspond to the minimum delivered energy dose. This can be calculated as follows:

The distance away from the centre of the egg described by the resonant scanner at any point in time can be given by the following equation:

$$x = X.\cos(2\pi ft) \qquad \textit{(Equation 5.3)}$$

Where: X is the maximum resonant scanner displacement
f is the resonant scanner frequency
t is the time from start of oscillation
x is the distance from the centre at time t

Differentiating this equation with respect to time will now give us the velocity, v, at time t:

$$v = 2\pi fX.\sin(2\pi ft) \qquad \textit{(Equation 5.4)}$$

Clearly the maximum velocity will be given when $\sin(2\pi ft)$ equals 1:

$$\therefore \text{ Maximum velocity} = 2\pi \times 294 \times 25 \text{ mm} = 46\,181 \text{ mm.s}^{-1}$$

$$\therefore \text{ Minimum time taken to travel single spot diameter} = \frac{0.675}{46\,181} = 14.6\,\mu\text{s}$$

$$\therefore \text{ Minimum energy density} = 39 \text{ kW}\cdot\text{cm}^{-2} \times 14.6\,\mu\text{s} = 0.569 \text{ J}\cdot\text{cm}^{-2}$$

Now, the period of minimum velocity over the surface of the egg while traversing a single spot diameter will correspond to the maximum delivered energy dose. This can be calculated as follows:

The time taken to travel half of the laser spot diameter at the end of the scanner's stroke can be found by rearranging equation 5.3 and substituting:

$$t = \frac{\cos^{-1}\left(\frac{x}{X}\right)}{2\pi f} = \frac{\cos^{-1}\left(\frac{25 - \frac{0.675}{2}}{25}\right)}{2 \times \pi \times 294} = 89\,\mu s$$

∴ Time taken to traverse the laser spot will be 2 × 89 = 178 μs

∴ Maximum energy density = 39 kW·cm^{-2} × 178 μs = 6.942 J·cm^{-2}

The above calculations are summarised in Table 5.4.

Table 5.4: Summary of prototype system energy densities.

Energy density	**J·cm^{-2}**
Minimum	0.569
Average	0.897
Maximum	6.942

Previous theoretical calculations have shown that the energy density required to ablate a planar coating of bacteria is 0.13 J·cm^{-2}. As discussed in section 3.2.5 (Light Absorption in Water) however various effects of the substrate and micro-organisms being targeted may lead to a general increase in the energy density required.

During the development of the prototype system for the above trials, a small utility program was written in Microsoft's Visual Basic to automate the above calculations for a variety of trial configurations (see Appendix 2: Utility Program). A screen shot of this utility program can be seen in figure 5.1.

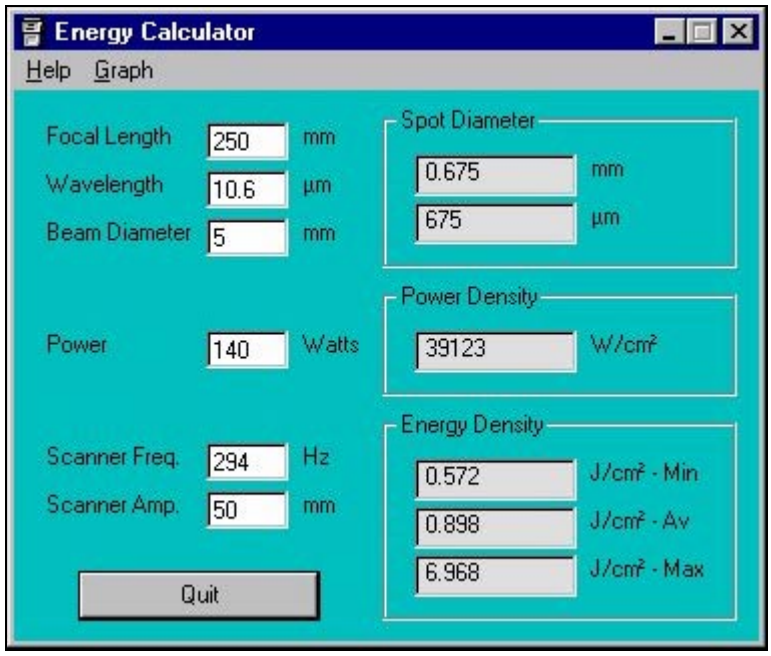

Fig. 5.1: Screen shot of energy density calculator.

An additional feature of the above utility program was the automatic generation of a graph modelling the response of the resonant scanner and associated energy density profile. This used the previously described equations for simple harmonic motion together with the laser system's key parameters. A screen shot of such a graph using the laser system's parameters for the prototype trial system can be seen in figure 5.2.

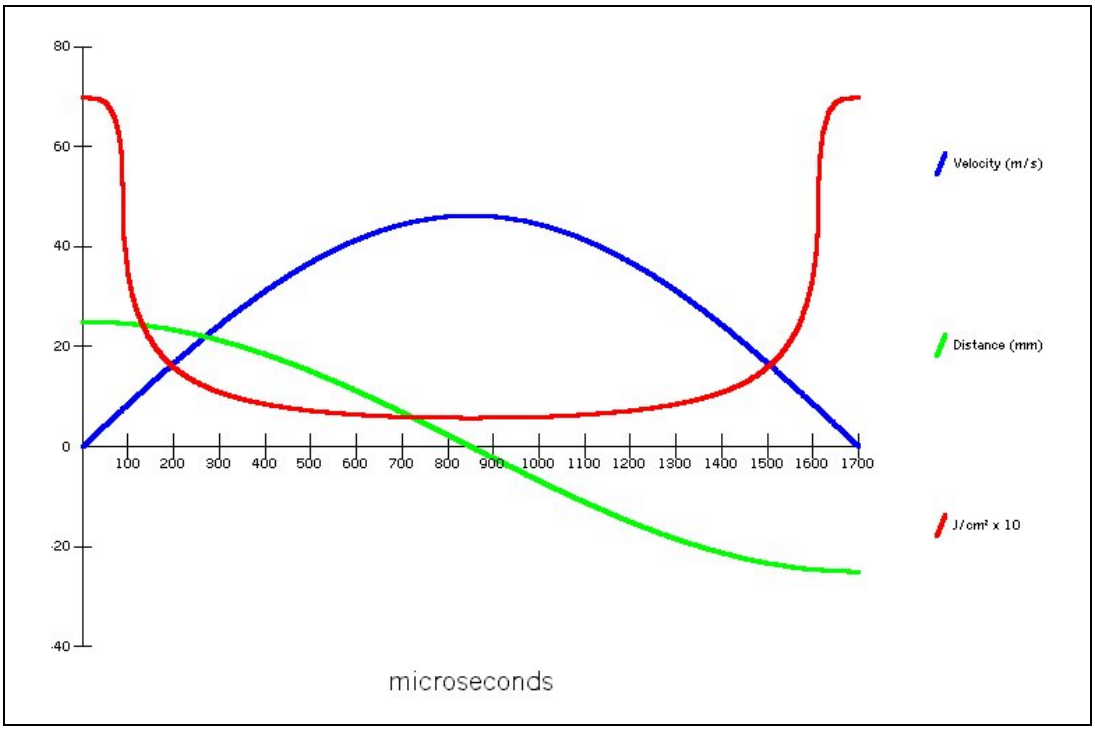

Fig. 5.2: Screen shot of energy density graph.

Figure 5.2 shows the velocity of the laser spot over the egg's surface due to the resonant scanner's oscillation (in blue). The velocity of this spot can be seen to vary from 0 $m \cdot s^{-1}$ to 46 $m \cdot s^{-1}$. The green line denotes the distance of the laser spot from the centre point of the described scan line. Comparing this to the velocity of the spot, it can be seen that the dwell points (zero velocity) of the resonant scanner's stroke occur at the extremities of the scanned line (25 mm from the centre), while the maximum velocity of the spot occurs at the centre of the scanner's stroke. Finally, the red line illustrates the energy density at the laser spot throughout the scan cycle. This clearly shows the much increased energy densities at the extremities of the scanner's cycle (the dwell points) as predicted. Indeed this profile matches very closely with the observed scan pattern caught on thermally sensitive paper (figure 4.9).

Total egg coverage now depends on the relative linear velocity of the egg to the scanned laser beam:

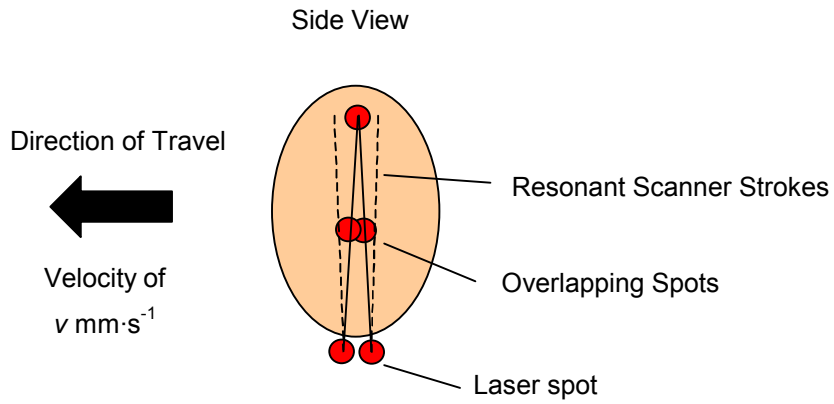

Fig. 5.3: Egg passing resonant scanner beam.

Distance travelled by egg during full cycle of scanner:

$= 150 \text{ mm·s}^{-1} \times 294^{-1} \text{ s}$

$= 0.510 \text{ mm}$

Hence, with a 0.675 mm diameter laser spot size there will be an overlap of 0.675 mm - 0.510 mm = 0.165 mm under worst-case conditions (at 150 mm·s^{-1}), guaranteeing complete egg coverage. Moreover, this overlap helps to compensate for the naturally lower energy density found at the edges of the laser spot due to the beam's Gaussian (TEM_{00}) spatial profile. However, at the same point that this minimum overlap occurs, the resonant scanner will be at the end of its stroke leading to an area of the egg shell being covered twice in rapid succession. This will in-turn give rise to a doubling of the applied energy density at this point from 6.942 J·cm^{-2} to 13.884 J·cm^{-2}. This is the point at which damage to the egg shell was observed.

Now, at the centre of the resonant scanner's stroke the least energy advantage will be obtained due to overlapping laser spots on consecutive strokes:

Distance travelled by egg during half cycle of scanner:

= 0.510 mm ÷ 2
= 0.255 mm

Thus, with a 0.675 mm diameter laser spot size there will be an overlap of 0.675 mm - 0.255 mm = 0.420 mm (at 150 mm·s^{-1}).

These calculations are all based on the eggs travelling at a linear velocity of 150 mm·s^{-1} with no tracking galvanometer compensation. As this velocity is increased or decreased, the relative decrease or increase in applied energy density due to laser spot overlap will change proportionally. As the linear velocity is increased above 150 mm·s^{-1} gaps in coverage may begin to appear (when used in conjunction with a laser spot diameter of 0.675 mm). The tracking of the galvonometers will help to increase the overlap at a given linear velocity.

Figure 5.4 shows a laser scan pattern caught on thermally sensitive paper. The laser power for this particular scan pattern had been turned down to prevent the thermally sensitive paper from igniting, as a consequence the modulation of the laser can be clearly seen by the pattern of light and dark patches. Additionally, this scan pattern was made with the use of the aluminium masks described in section 4.5.2 (Resonant Scanner with Laser Power Modulation). Thus, the above scan pattern does not exhibit the dwell points as previously seen in figure 4.9. This scan pattern was taken while the linear velocity was set at 150 mm·s^{-1}. A good uniformity of coverage can be seen from this scan pattern, taking into account the lower modulation level of the lasers.

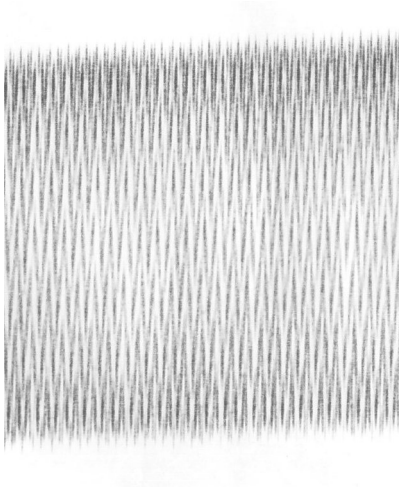

Fig. 5.4: Scan pattern showing uniformity of coverage.

5.5 Summary

This chapter has described the experimental methodology and subsequent analysis of the trial results for the proposed system. Prior to the commencement of the trials eggs to be processed were pre-contaminated with a bacterial broth containing *Salmonella enteritidis*. Once processed the remaining bacteria were extracted from the eggs, incubated and counted using the total viable count (TVC) plate count method taken from *BS5763: Part 1 1991 – General guidance for the enumeration of micro-organisms colony count technique at 30 ºC* [53].

The trial system processed eggs at linear velocities of between 150 $mm \cdot s^{-1}$ and 75 $mm \cdot s^{-1}$ with a 140 Watt CO_2 laser system. The laser was focused using a 250 mm focal length lens via a resonant scanner running at 294 Hz and describing a 50 mm line. A linear tracking galvonometer was used to further increase the efficiency of the available laser power. At the given linear velocities, complete coverage was demonstrated.

This system achieved a maximum kill rate of 99.988 % (3.9 log) for a linear velocity of 100 $mm \cdot s^{-1}$ (calculated using the trimmed mean values for bacterial counts) with an average energy density of 0.9 $J \cdot cm^{-2}$. This proving the efficacy of the proposed system.

6 : DISCUSSION

6.1 Introduction

The objective of this study was to determine the optimum choice of a generic laser system, and its operational parameters, to sterilise a variety of micro-organisms, on a range of substrates, with minimal damage being caused to the substrate. Additionally a practical embodiment of such a system was to be investigated to provide a uniform dose with consistent sterilisation results.

With this in mind, a study of micro-organisms and their requirements for growth and reproduction was conducted to gain an understanding of what processes could be brought upon these organisms to sterilise them. Following on from this a review of current sterilisation techniques looked at the different methods available to sterilise bacteria and micro-organisms.

With a good understanding of micro-organisms and various methods of sterilisation, the next stage was to choose the optimum laser system, and its parameters, to kill the broadest range of micro-organisms. This was achieved with a thorough analysis of the electromagnetic spectrum, commercially available lasers covering the light portion of the spectrum and their differing sterilisation effects. The general principals of light absorption, and more specifically light absorption in water, were reviewed together with a review of the existing body of knowledge relating to the sterilisation of micro-organisms by laser light. Finally, an analysis of the requirements for the laser's spatial and temporal profiles was conducted to ascertain the optimum parameters for the selected laser.

With the knowledge gained of the bacterial kill mechanisms with laser sources, and the choice of laser system and its key operational parameters being established, a practical design solution had to be found that could process items to be sterilised on a commercial-scale. A suitable practical requirement for such a system was the sterilisation of chicken eggs from contaminating *Salmonella* bacteria. The first step in designing the system being an analysis of the items to be sterilised and their specific handling requirements. The key technical solutions to

provide a uniform coverage of the eggs were discussed together with the specific design requirements for this particular implementation.

The final stage of the project was to conduct live biological trials on the completed machine. This required the correct preparation and analysis of biologically contaminated samples together with the physical experimental configuration. Following the trials, an in-depth analysis of the experimental results and the system's technical performance was conducted.

6.2 Discussion

The two most common methods of sterilisation for commercial purposes are chemical disinfectants and heat. One of the primary reasons for the choice of topic for this thesis, following the *Salmonella* scare with eggs in the United Kingdom in 1988, was to find a commercially acceptable method of sterilising the surface of hatching eggs that did not involve chemical disinfectants currently employed. That is, to develop a safe and efficient reagentless method of sterilisation.

While chemical disinfectant methods are cheap and commonplace, they do have some pitfalls. When the chemicals are fresh and being used at their optimum parameters (temperature and concentrations), they can prove very effective. However, as the system is used the chemicals become diluted and performance degrades. The safe disposal of the used chemicals is also becoming an increasing problem, particularly with such government regulations as COSHH (Control Of Substances Hazardous to Health).

Heat sterilisation methods are extremely efficient, do not use chemicals and hence do not leave any harmful residues. However, traditional heating methods also heat the whole substrate as well as the micro-organisms. For substrates that are heat sensitive, this method cannot be used and hence alternative methods such as chemical disinfectants must be sought.

Hence, the development of a sterilisation method that can rapidly heat the external surface of a substrate to kill bacteria, without raising the internal temperature of the substrate, is much desired.

Using light to sterilise items is not a new concept, with the two sterilisation mechanisms being photo-chemical (ultra-violet light) and photo-thermal (infrared light). The first method targets the cell's nucleic acids and proteins as chromophores, causing cell mutagenicity. The second method targets water as the primary chromophore, causing heating and ultimately cell death.

Vegetative bacteria are comprised 80 % water by weight. However, ultra-violet light is absorbed in water. Hence, a significant proportion of incident ultra-violet light energy will be absorbed by the water before it reaches the cell's proteins and nucleus, where the majority of its bactericidal activity would take place. Ultra-violet (excimer) lasers are relatively expensive compared to alternative laser technologies such as the infrared CO_2 laser, and use halogen gases as part of their lasing medium, which are not considered environmentally friendly. These lasers only operate in pulsed modes, thus presenting some problems for uniformity of coverage on moving product as would be typically required on a commercial sterilisation system.

The infrared light generated by a CO_2 laser however, targets the cell's water content, in which it is highly absorbed, as its chromophore. For this reason the sterilisation efficiency of a CO_2 laser compared to an equivalent excimer laser is much greater. The CO_2 lasers are also continuous wave devices, so aiding the uniform processing of moving objects. It is for these reasons that the continuous wave CO_2 laser was taken as the optimal choice for a generic laser-based sterilisation system.

By default, lasers generate a spot of laser light as a result of their lasing action. However, this spot of light needs to cover the complete surface area of a three-dimensional object as fast and efficiently as possible. From this viewpoint a spot of laser light is the worst starting point possible. To cover a range of three-dimensional objects it is far better to first manipulate the spot of laser light into a

line of laser light, which can then be utilised to treat an object through relative motion between the line of laser light and the object being treated. The manipulation of the laser beam into a line of laser light was achieved with a resonant scanner compensating for the natural curvature of the egg from the *y*-plane to the *z*-plane. This method would also be applicable for any other approximately spherical object requiring treatment.

As an additional aid to track moving product a tracking galvonometer was used to make the most efficient use of laser power during the gaps found between items on a typical processing line. This method also helped to compensate for the natural curvature of spherical objects from the *x*-plane to the *z*-plane.

Table 6.1 shows the key results of this study in comparison to two laboratory trials conducted by Watson *et al.* [7] and Mullarky *et al.* [42] with CO_2 lasers. However, it is very difficult to extract a high degree of coherence between these results for the following reasons.

- The substrates on which the trials were conducted varied. This study used chicken eggs while Watson *et al.* used nutrient agar and Mullarky *et al.* used porcine skin.

- The bacteria on which the trials were conducted varied. This study used *Salmonella enteritidis* while Watson *et al.* used *Escherichia coli* and Mullarky *et al.* used *Escherichia coli* and *Staphylococcus aureus*. While *Escherichia coli* and *Salmonella enteritidis* are genetically similar (about 60 % to 70 %), they may still be sufficiently different to have an effect on the results.

- A direct, quantifyable kill rate for Watson *et al.* does not exist. While their trials prove the CO_2 laser to be the optimal choice for bacterial sterilisation, the system provided a killing area of 1.2 cm^2 of a total laser beam area of 2.3 cm^2. Unfortunately, from this a kill rate cannot be extrapolated.

Table 6.1: Comparison of CO_2 laser sterilisation trials.

Laser Power (W)	Spot size (mm)	Power Density ($W \cdot cm^{-2}$)	Duration (s)	Energy Density ($J \cdot cm^{-2}$)	Kill rate (log)
140 [a]	0.7	39 000	23×10^{-6}	0.90	4
600 [b]	8.6	262	30×10^{-3}	7.88	Unknown
25 [c]	3	354	10	3 540	7

[a] This thesis.

[b] Watson *et al.*

[c] Mullarky *et al*.

From Table 6.1 it can seen that the variety of laser power levels and spot sizes for the different trials contribute to roughly equal power densities for Watson *et al.* and Mullarky *et al.* but a power density of two times greater order of magnitude for this study.

While this trial had the highest power density, the corresponding energy density was the lowest due to the extremely short duration. However, this low energy density still proved highly efficient in bacterial sterilisation. The time in which the energy is imparted to the bacteria is thought to be very significant. The shorter the duration for a given energy density being advantageous as the thermal shock to the bacteria is greater and the energy has less time to diffuse into the surrounding substrate.

The effects of cold thermal shocks to precipitate increased sterilisation in comparison to slow cooling have been documented [13], hence it is logical to assume that hot thermal shocks should also increase sterilisation efficiency compared to slower increases in temperature.

6.3 Conclusions

The overriding conclusion from this body of work is that not only is the CO_2 laser the best theoretical choice of laser for a generic bacterial sterilisation system, but the prior evidence gained in various practical laboratory experiments supports this theory. The CO_2 laser is also the most common and widely available laser system for commercial applications on the market today. Furthermore, the migration of this proven laboratory technology into a working commercial system capable of achieving bacterial kill rates of 99.988 % (3.9 log) at its first iteration shows the long term potential of this technology.

In comparison with existing methods of sterilising hatching eggs, such as chemical disinfectants, the proposed system is equally efficient at killing bacteria. However, the laser system has some disadvantages in being substantially more costly and has a lower throughput than the conventional chemical systems. The laser system can also offer many benefits as it can be switched on almost instantly, without having to wait half an hour or more for the chemical bath to achieve correct operating temperature. Moreover, the laser system does not use any harmful chemicals and as such, chemical disposal costs are not required.

While only one practical embodiment of this technology has been demonstrated in this study, the potential for other embodiments clearly exists utilising the core elements of the design. This highlights the true long-term potential of this technology to be a valid adjunct to existing commercial sterilisation technologies in the future.

6.4 Future Work

While the overall conclusion of this study shows that a generic laser system can provide efficient bacterial sterilisation on a commercial scale, much work needs to be done to gain a fuller and deeper understanding of the exact mechanisms involved.

Such an understanding would enable predictions of sterilisation efficiency to be made based on the bacteria likely to be encountered on a given substrate. Ideally these predictions should be able to describe the required laser energy density levels, and other system parameters, to provide a minimum predetermined kill level on a given substrate, without causing any detrimental effects to the substrate or product being treated.

Figure 6.1 shows a graph of what may be expected from an operating window of desired kill rate versus energy density. As the energy density level increases, the kill rates experienced will also increase. For any sterilisation system there will be a minimum acceptable kill rate (K_{min}), which must be consistently obtained. This minimum kill rate will correspond to a minimum energy density level (E_{min}). However, above a certain energy density level damage may be sustained to the substrate being processed. In this case it is the energy density level above which the egg's cuticle is compromised (E_{cd}).

From figure 6.1, the operating window can be seen shaded in grey. Energy densities below this level will have less than acceptable kill rates, while energy density levels above this window will cause damage to the substrate being processed. Ideally such a graph should be able to be predicted for any bacterial / substrate combination.

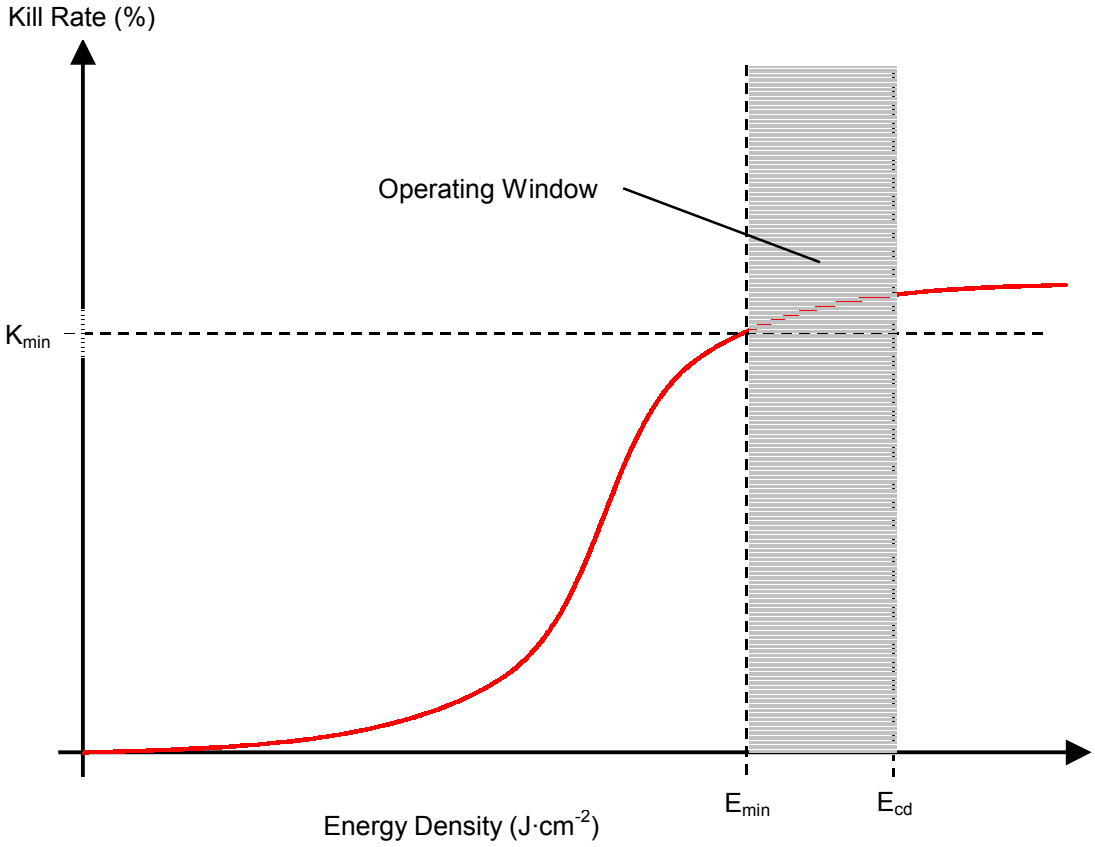

Fig. 6.1: Graph of system operating parameters.

To increase the confidence levels on the results presented in this thesis, trials should be conducted on a greater number of samples, preferably with a broader range of system parameters and contaminating micro-organisms. The larger sample size will be statistically more robust, so increasing the confidence and validity of the presented results. The broader range of system parameters and contaminating micro-organisms will help to give a better understanding of the operating window within which a typical system should operate.

The majority of existing trials to date have been conducted on artificially contaminated samples. To prove the efficiency of laser sterilisation systems in real world scenarios, quantifiable trials based on naturally contaminated samples need to be conducted. For the example of chicken eggs used in this thesis, hatchability

is one such quantifiable metric. In this case, the successful sterilisation of the egg's surface should directly translate into an increase in hatchability.

Eggs free of contaminating micro-organisms (particularly *Salmonella enteritidis*) allow disease-free chicks to be incubated. Eggs contaminated with micro-organisms can lead to a reduction of chick viability through contracted diseases. These diseases may be contracted via the micro-organisms passing through the pores of the egg shell and into the developing embryo. Alternatively, the emerging chick may be exposed to the micro-organisms on the shell's surface as it breaks out of the shell. Any increase in hatchability would give a strong financial case for the commercial implementation of such a system.

In a sterilisation procedure it is desirable for the treated substrate to remain sterilised for the greatest duration subsequent to treatment. Clearly, organism re-growth would be a problem for any sterilisation system. While areas of laser-sterilised substrate have been shown to exhibit no organism re-growth in laboratory trials [7], further work needs to be conducted on substrates and mechanisms more indicative of commercial applications.

The use and efficiency of a laser based sterilisation system will be highly dependent on the substrate being treated. Such factors that may have an effect might be for example; surface texture**,** thermal conductivity, colour and chemical composition. With this in mind a wide variety of substrates need to be treated to build a knowledge base for reference and comparison.

In addition to experimental evidence, it is desirable to model the sterilisation process as accurately as possible. Such a model should be able to predict system parameters for a given substrate and processing requirements, and be in close agreement with the body of empirical evidence. Such modelling may incorporate thermal modelling of a complex, three-dimensional, layered substrate where the rate of energy transfer of the impinging laser beam becomes highly important. Some work to this aim has been attempted by Wang *et al.* [55] with the thermal model of a laser interacting with a liquid suspension of *Escherichia coli*.

In addition to the energy density being applied to a substrate to promote efficient sterilisation, the time in which this energy is imparted is thought to be extremely important. Further work to research this hypothesis by both experimental study and theoretical modelling is suggested. Taking this to extremes, a study conducted by Gribin *et al.* [56] has shown that extremely short (30 ns) high power pulses from a Nd:YAG laser can produce "micro-blasts" in liquid leading to the mechanical destruction of bacteria.

In the real world, items needing to be sterilised will come in all shapes and sizes. In order to provide uniform dose levels for irregular three-dimensional shapes complex scanner systems will need to be devised, with different shaped objects requiring lines of laser light with differing energy density profiles. These will present many engineering challenges in order to both satisfy the above requirements while keeping the overall system costs as low as possible.

7 : BIBLIOGRAPHY

[1] ELION, H. A. *Laser Systems and Applications*. Pergamon Press, 1967.

[2] SAKS, N. M. and ROTH, C. A. Ruby Laser as a Microsurgical Instrument. *Science*, 1963, Vol. 141, pp. 46-47.

[3] MCGUFF, P. E. and BELL, E. J. The Effect of Laser Energy Radiation on Bacteria. *Medical and Biological Illustration*, 1966, pp. 191-194.

[4] ADRIAN, J. C. and GROSS, A. A new Method of Sterilization: the CO_2 Laser. *Journal of Oral Pathology*, 1979, 8, pp. 60-60.

[5] SCHULTZ, R. J. *et al*. Bactericidal Effects of the Nd:YAG Laser: in vitro Study. *Lasers in Surgery and Medicine*, 1986, 6, pp. 445-448.

[6] POWELL, G. L. and WHISENANT, B. K. Comparison of Three Lasers for Dental Instrument Sterilization. *Lasers in Surgery and Medicine*, 1991, 11, pp. 69-71.

[7] WATSON, I. A. *et al*. Comparative Bactericidal Activities of Lasers Operating at Seven Different Wavelengths. *Journal of Biomedical Optics*, October 1996, 1(4), pp. 466-472.

[8] PRODUCT LITERATURE. Pall Gelman.

[9] GUTHE, K. F. *The Physiology of Cells*. Macmillan, 1968.

[10] SISTROM, W. R. *Microbial Life*. Holt, Rinehart, Winston, 1969.

[11] SMITH, K. M. *Viruses*. Cambridge University Press, 1962.

[12] STEVENSON, G. *The Biology of Fungi, Bacteria and Viruses*. Arnold, 1970.

[13] HAWKER, L. E. and LINTON, A. H. *Micro-organisms*. Arnold, 1979.

[14] SYKES, J. B. *The Oxford Concise Dictionary*. 7th Edition. Oxford University Press, 1984.

[15] GAYFORD, C. G. *Energy and Cells*. Macmillion, 1986.

[16] DOYLE, M. P., BEUCHAT, L. R. and MONTVILLE, T. J. *Food Microbiology Fundamentals and Frontiers*. ASM Press, 1997.

[17] *Salmonella Information*. URL: http://www.salmonella.org/info.html [01 May 2002]

[18] MAURICE, J. The Rise and Rise of Food Poisoning. *New Scientist*, 17 December 1994, pp. 28-33.

[19] O'DOHERTY, J. Validating Radiation Sterilization. *Medical Device Technology*, July/August 1997.

[20] MORTIMER, C. E. *Chemistry, a Conceptual Approach*. D. Van Nostrand, 1975.

[21] DUNN, J. *et al*. Sterilization using Pulsed White Light. *Medical Device Technology*, July/August 1997.

[22] DUNN, J. Pulsed Light and Pulsed Electric Field for Foods and Eggs. *Poultry Science,* 1996, 75, pp. 1133-1136.

[23] PÄTZEL, R. Once a Scientific Tool the Excimer now fills many roles. *The Photonics Design and Applications Handbook*. Laurin Publishing Co. Inc., 1995, pp. 288-295.

[24] MACMILLAN, J. D. *et al*. Lethal Photosensitization of Microorganisms with Light from a Continuous-wave Gas Laser. *Photochemistry and Photobiology*, 1966, Volume 5, pp. 555-565.

[25] WILSON, M., DOBSON, J. and HARVEY, W. Sensitization of Oral Bacteria to Killing by Low-Power Laser Radiation. *Current Microbiology*, 1992, Vol. 25, pp. 77-81.

[26] GRAHAM, G. and MIELNIK,T. J. Industrial Low Temperature Gas Plasma Sterilisation. *Medical Device Technology*, July/August 1997.

[27] McBEE, L. E. Innovative Methods of Energy Transfer. *Poultry Science,* 1996, 75, pp. 1137-1140.

[28] LAWRENCE, C. W. *Cellular Radiobiology*. Arnold, 1971.

[29] BROOKES, M. Day of the Mutators. *New Scientist*, 14 February 1998, pp. 38-42.

[30] CARDARELLI, F. *Scientific Unit Conversion*. Springer, 1998.

[31] WILSON, J. and HAWKES, J. F. B. *Lasers, Principles and Applications*. Prentice Hall, 1987.

[32] GWYNNE, P. *The Photonics Design and Applications Handbook*. Laurin Publishing Co. Inc., 1995.

[33] LERMAN, S. *Radiant Energy and the Eye*. Macmillan, 1980.

[34] YARBOROUGH, J. M. Taking the Confusion out of Matching Medical Lasers. *The Photonics Design and Applications Handbook*. Laurin Publishing Co. Inc., 1995, pp. 307-310.

[35] KAROUTIS, A. D. *et al*. Bactericidal Effect of ArF Excimer Laser Radiation. on Gram-Negative Bacteria. *Lasers in the Life Sciences*, 1996, 7(1), pp. 59-70.

[36] HAGMANN *et al*. *Method of Surface-cleaning and/or Sterilizing Optical Components, Especially Contact Lenses*. August 1995. United States Patent 5,439,642.

[37] FRUCHT-PERY, J. *et al*. The Effect of the ArF Excimer Laser on Candida albicans in vitro. *Graefe's Archive for Clinical and Experimental Othalmology*, 1993, pp. 413-415.

[38] HIBST, R. *et al.* Er:YAG laser for endodontics: efficiency and safety. *SPIE*, 1997, Volume 3192, pp. 14-21.

[39] WATSON, I. A. *et al*. Nd:YAG Laser Sterilization of Escherichia coli and Bacillus stearothermophilus. Japan: *CLEO/LEOS Pacific Rim*, July 1995, Paper THL4.

[40] WARD, G. D. *et al*. Inactivation of Bacteria and Yeasts on Agar Surfaces with High Power Nd:YAG Laser Light. *Letters in Applied Microbiology*, 1996, Vol. 23, pp. 136-140.

[41] ROONEY, J. *et al*. Laboratory Investigation of the Bactericidal Effect of a Nd:YAG Laser. *British Dental Journal*, 22 January 1994, pp. 61-64.

[42] MULLARKY, M. B., NORRIS, C. W. and GOLDBERG, I. D. The Efficiency of the CO_2 Laser in the Sterilization of Skin Seeded with Bacteria: Survival at the Skin Surface and in the Plume Emissions. *Laryngoscope*, February 1985, 95, pp. 186-187.

[43] AL-QATTAN, M. M. *et al*. Wound Sterilization: CO_2 Laser Versus Iodine. *British Journal of Plastic Surgery*, 1989, Vol. 42, pp. 380-384.

[44] PRATT, Jr., G. W. *Method of Altering Biological and Chemical Activity of Molecular Species*. March 1976. United States Patent 3,941,670.

[45] HOOKS, T. W. *et al*. Use of Carbon Dioxide Laser in Sterilization of Endodontic Reamers. *Journal of Oral Surgery*, March 1980, pp. 263-265.

[46] OVERFIELD, N. D. *Quality Testing of Eggs: Reference Book 428*. Her Majesty's Stationary Office, 1982.

[47] *Eggcyclopedia – Shell.* American Egg Board. URL: http://www.aeb.org/eggcyclopedia/shell.html [10 May 2002]

[48] PHIL CANNING. personal communication. ADAS, 24 March 1995.

[49] *Eggcyclopedia – Albumen.* American Egg Board. URL: http://www.aeb.org/eggcyclopedia/albumen.html [10 May 2002]

[50] INFRARED GAS LASERS - INSTRUCTION MANUAL. Synrad. 31 May 1991.

[51] KERNIGHAN, B. W. and RITCHIE, D. M. *The C Programming Language*. 2nd Edition. Prentice Hall, 1988.

[52] ANON. *80C51-based 8-bit Microcontrollers: Data Book IC20*. Philips Semiconductors, March 1994.

[53] INNTERNAL CORRESPONDENCE. Total Viable Count by the Plate Count Method. ADAS, June 1994.

[54] PAWSEY, R. K. *Techniques with bacteria*.Hutchinson Educational, 1974.

[55] WANG, R. K. *et al*. Temperature Distribution in Escherichia coli Liquid Suspensions During Irradiation by a High-powered Nd:YAG Laser for Sterilization Applications. *Journal of Biomedical Optics*, July 1997, 2(3), pp. 295-303.

[56] GRIBIN, S. *et al*. Liquid Disinfection Using Power Impulse Laser. *Proceedings of the SPIE*, 1996, Volume 2714, pp. 229-237.

8 : APPENDICES

8.1 Appendix 1: Prototype Trial Results

The following data for the prototype trial were taken from the output of the Minitab computer statistical analysis program used to collate and analyse these results.

- C1 are the data for 10 untreated control eggs.
- C2 are the data for 10 eggs treated at 150 $mm \cdot s^{-1}$ and mid focus
- C3 are the data for 10 eggs treated at 100 $mm \cdot s^{-1}$ and mid focus
- C4 are the data for 10 eggs treated at 100 $mm \cdot s^{-1}$ and surface focus
- C5 are the data for 10 eggs treated at 150 $mm \cdot s^{-1}$ and surface focus
- C6 are the data for 10 eggs treated at 75 $mm \cdot s^{-1}$ and surface focus

Eggs contained within data sets C2 and C3 were treated with the laser focus set to half-way between the egg's surface and centre point, while eggs contained within data sets C4 to C6 were treated with the laser focus at the egg's surface.

The following is a brief description of the symbols used in the Minitab output:

N	The number of observations.
MEAN	The arithmetic mean of the N observations.
MEDIAN	The median of the N observations.
TRMEAN	The trimmed mean of the N observations.
STDEV	The standard deviation of the N observations.
SEMEAN	The standard error of the arithmetic mean.
MINIMUM	The smallest observation of N observations.
MAXIMUM	The largest observation of N observations.
Q1	The first quartile of N observations.
Q3	The third quartile of N observations.

```
MTB > describe c1-c6

      N     MEAN   MEDIAN    TRMEAN    STDEV     SEMEAN
C1   10  3285540  1200000   2731250  3991435   12662203
C2   10     1049      190       873     1335        422
C3   10     3667       90       334    10668       3373
C4   10    55008       95      2508   166989      52807
C5   10    27132    14200     23036    32316      10219
c6   10    46007     6500     27483    82164      25983

     MIN       MAX      Q1         Q3
C1  5400  11000000  145000    7025000
C2    10      3500      30       2325
C3     0     34000      20       1125
C4    20    530000      30       5625
C5    30     87000     273      58500
c6   210    240000     255      65250

MTB > print c1-c6

  ROW         1     2      3       4      5       6

    1   6700000  3500     90      20     30   37000
    2    190000    10     90      30  23000  150000
    3    100000    30   1200  530000     40     380
    4  11000000   290     20      30  55000     210
    5   1200000    30    110     250  69000   19000
    6      5400    90  34000    1500   2500    8800
    7   8000000  1600      0     120  87000    4200
    8    160000  2700   1100   18000   5400     210
    9   1200000    40     20      70  29000     270
   10   4300000  2200     40      60    350  240000

MTB > Save  'H:\MINITAB\egg3010.mtw'.

Worksheet saved into file: H:\MINITAB\egg3010.mtw
MTB > stop
*** Minitab Release 8.2 *** Minitab, Inc. ***
```

8.2 Appendix 2: Utility Program

The following listings detail the software developed for the energy density calculator / modelling utility program written in Microsoft's Visual Basic language (version 4).

Being of a visual / object oriented nature, much of the software code generated by the Visual Basic program pertains to the description of the visual objects (buttons, textboxes and graphs for example). As this code is quite voluminous and does little to add to the calculations and algorithms used within the described utility, these have been omitted from the listings below for the sake of brevity.

8.2.1 global.bas

```
Global scanner_frequency
Global scanner_amplitude
Global pwr_density
Global spot_size
Global min_energy
Global av_energy
```

8.2.2 energy.frm

```
Function arccos(x)
    arccos = Atn(-x / Sqr(-x * x + 1)) + 2 * Atn(1)
End Function


Private Sub About_Click()
    aboutnrg.Show 1
End Sub
```

```
Private Sub amplitude_Change()
If Val(amplitude.Text) > 0 And Val(frequency.Text) > 0 Then
    av    =    (Val(power_density.Caption)    *    Val(spot_diameter.Caption))    /
(Val(amplitude.Text) * Val(frequency.Text) * 2)
     av_energy = av
   energy_av.Caption = " " + Format(Str$(av), "0.000")
    Min    =    (Val(power_density.Caption)    *    Val(spot_diameter.Caption))    /
(Val(amplitude.Text) * Val(frequency.Text) * 3.14159)
   energy_min.Caption = " " + Format(Str$(Min), "0.000")
   halfamp = Val(amplitude.Text) / 2
   Max      =      (Val(power_density.Caption)      *      arccos((halfamp      -
(Val(spot_diameter.Caption) / 2)) / halfamp)) / (Val(frequency.Text) * 3.14159)
   energy_max.Caption = " " + Format(Str$(Max), "0.000")
Else
   energy_av.Caption = ""
   energy_min.Caption = ""
   energy_max.Caption = ""
End If
    scanner_amplitude = Val(amplitude.Text)
End Sub


Private Sub beam_diameter_Change()
If Val(beam_diameter.Text) > 0 Then
    spot  =  (4  *  Val(wavelength.Text)  *  Val(focal_length.Text))  /  (3.14159  *
Val(beam_diameter.Text) * 1000)
    spot_diameter.Caption = " " + Format(Str$(spot), "0.000")
    spot_um.Caption = " " + Format(Str$(spot * 1000), "0")
Else
    spot_diameter.Caption = ""
    spot_um.Caption = ""
    power_density.Caption = ""
End If
End Sub


Private Sub focal_length_Change()
If Val(beam_diameter.Text) > 0 Then
    spot  =  (4  *  Val(wavelength.Text)  *  Val(focal_length.Text))  /  (3.14159  *
Val(beam_diameter.Text) * 1000)
    spot_diameter.Caption = " " + Format(Str$(spot), "0.000")
    spot_um.Caption = " " + Format(Str$(spot * 1000), "0")
Else
    spot_diameter.Caption = ""
    spot_um.Caption = ""
    power_density.Caption = ""
End If
End Sub


Private Sub Form_Load()
    Move (Screen.Width - Width) \ 2, (Screen.Height - Height) \ 2
    wavelength.Text = "10.6"
    beam_diameter.Text = "5"
    focal_length.Text = "250"
    power.Text = "140"
    frequency.Text = "294"
    amplitude.Text = "50"
End Sub


Private Sub frequency_Change()
If Val(amplitude.Text) > 0 And Val(frequency.Text) > 0 Then
    av    =    (Val(power_density.Caption)    *    Val(spot_diameter.Caption))    /
(Val(amplitude.Text) * Val(frequency.Text) * 2)
    av_energy = av
   energy_av.Caption = " " + Format(Str$(av), "0.000")
    Min    =    (Val(power_density.Caption)    *    Val(spot_diameter.Caption))    /
(Val(amplitude.Text) * Val(frequency.Text) * 3.14159)
   min_energy = Min
   energy_min.Caption = " " + Format(Str$(Min), "0.000")
   halfamp = Val(amplitude.Text) / 2
   Max      =      (Val(power_density.Caption)      *      arccos((halfamp      -
(Val(spot_diameter.Caption) / 2)) / halfamp)) / (Val(frequency.Text) * 3.14159)
   energy_max.Caption = " " + Format(Str$(Max), "0.000")
Else
```

```
   energy_av.Caption = ""
   energy_min.Caption = ""
   energy_max.Caption = ""
End If
    scanner_frequency = Val(frequency.Text)
End Sub


Private Sub Graph1_Click()
    Graph_form.Show 1
End Sub


Private Sub power_Change()
If Val(spot_diameter.Caption) > 0 Then
    pwr = Val(power.Text) / (3.14159 * ((Val(spot_diameter.Caption) / 2) ^ 2))
    power_density.Caption = " " + Format(Str$(pwr * 100), "#####")

Else
    power_density.Caption = ""
End If
End Sub


Private Sub power_density_Change()
If    Val(amplitude.Text)    >    0    And    Val(frequency.Text)    >    0    And
Val(spot_diameter.Caption) > 0 Then
    av    =    (Val(power_density.Caption)    *    Val(spot_diameter.Caption))    /
(Val(amplitude.Text) * Val(frequency.Text) * 2)
   energy_av.Caption = " " + Format(Str$(av), "0.000")
    Min    =    (Val(power_density.Caption)    *    Val(spot_diameter.Caption))    /
(Val(amplitude.Text) * Val(frequency.Text) * 3.14159)
   energy_min.Caption = " " + Format(Str$(Min), "0.000")
   halfamp = Val(amplitude.Text) / 2
   Max      =      (Val(power_density.Caption)      *      arccos((halfamp      -
(Val(spot_diameter.Caption) / 2)) / halfamp)) / (Val(frequency.Text) * 3.14159)
   energy_max.Caption = " " + Format(Str$(Max), "0.000")
Else
   energy_av.Caption = ""
   energy_min.Caption = ""
   energy_max.Caption = ""
End If
    pwr_density = Val(power_density.Caption)
End Sub


Private Sub Quit_button_Click()
    End
End Sub


Private Sub spot_diameter_Change()
If Val(spot_diameter.Caption) > 0 Then
    pwr = Val(power.Text) / (3.14159 * ((Val(spot_diameter.Caption) / 2) ^ 2))
    power_density.Caption = " " + Format(Str$(pwr * 100), "#####")
Else
    power_density.Caption = ""
End If
    spot_size = Val(spot_diameter.Caption)
End Sub


Private Sub wavelength_Change()
If Val(beam_diameter.Text) > 0 Then
    spot  =  (4  *  Val(wavelength.Text)  *  Val(focal_length.Text))  /  (3.14159  *
Val(beam_diameter.Text) * 1000)
    spot_diameter.Caption = " " + Format(Str$(spot), "0.000")
    spot_um.Caption = " " + Format(Str$(spot * 1000), "0")
Else
    spot_diameter.Caption = ""
    spot_um.Caption = ""
        power_density.Caption = ""
End If
End Sub
```

8.2.3 graph_form.frm

```
Private Sub Command1_Click()
    graph_present = 0
    Unload Me
End Sub


Private Sub Command2_Click()
    Graph1.GraphType = 6
End Sub


Function arccos(x)
    If x = 1 Then
        arccos = 0
    ElseIf x = -1 Then
        arccos = 3.14159
    Else
        arccos = Atn(-x / Sqr(-x * x + 1)) + 2 * Atn(1)
    End If
End Function


Private Sub Form_Load()

    Move (Screen.Width - Width) \ 2, (Screen.Height - Height) \ 2
    Graph1.DataReset = 1
    Graph1.ThisPoint = 1
    halfamp = scanner_amplitude / 2
    period = 1 / scanner_frequency
    steps = 0.000001
    y = 0

    For x = 0 To period / 2 Step steps
        velocity = 2 * 3.14159 * scanner_frequency * halfamp * Sin(2 * 3.14159 * x
* scanner_frequency)
        distance = halfamp * Cos(2 * 3.14159 * x * scanner_frequency)

        If (distance + (spot_size / 2)) > halfamp Then
            d1 = halfamp - ((distance + (spot_size / 2)) - halfamp)
        Else
            d1 = distance + (spot_size / 2)
        End If
        If (distance - (spot_size / 2)) < -halfamp Then
            d2 = -halfamp - ((distance - (spot_size / 2)) + halfamp)
        Else
            d2 = distance - (spot_size / 2)
        End If
        t1 = arccos(d1 / halfamp) / (2 * 3.14159 * scanner_frequency)
        t2 = arccos(d2 / halfamp) / (2 * 3.14159 * scanner_frequency)
        tend = 1 / (2 * scanner_frequency)
        If (distance + (spot_size / 2)) > halfamp Then t1 = -t1
        If (distance - (spot_size / 2)) < -halfamp Then t2 = (tend - t2) + tend
        t3 = t2 - t1
        en = pwr_density * t3 * 10

        y = y + 1

        If y > 2 Then
            Graph_form.Graph1.NumPoints = Graph_form.Graph1.NumPoints + 1
        End If

        If y > 1 Then
            Graph_form.Graph1.ThisPoint = Graph_form.Graph1.ThisPoint + 1
        End If

        Graph1.ThisSet = 1
        Graph_form.Graph1.GraphData = velocity / 1000
```

```
        Graph1.ThisSet = 2
        If Graph1.GraphType = 7 And distance < 0 Then distance = -distance
            Graph_form.Graph1.GraphData = distance

        Graph1.ThisSet = 3
        Graph_form.Graph1.GraphData = en

    Next x
End Sub


Private Sub Form_Unload(Cancel As Integer)
        graph_present = 0
End Sub
```

8.3 Appendix 3: Research System Control Program

The following section details the control program for the trial machine. This program was written in 'C' for the 8051 series embedded micro-controller. The program is broken down into the following modules, each handling a specific group of closely related functions.

Table Appendix 3.1: *List of 'C' program files for trial system*

'C' File	Description
buzzer.c	Sounds and alerts for onboard buzzer.
eeprom.c	I^2C serial E^2PROM reading and writing routines.
galvos.c	Galvonometer control routines.
i2c.c	I^2C system bus control routines.
lcd_text.c	LCD display routines.
rs232.c	Low level RS-232 serial communications routines.
time.c	System delay / timing routines.
egg1.c	Main control program.

Associated with each 'C' file is a corresponding header file or '.h' file, which describes the functions to be found in the particular 'C' file together with definitions specific to that file.

The code for these files is listed below:

8.3.1 buzzer.h

```
#ifdef _BUZZER_

    void okbuzzer (void);               // okbuzzer sound
    void click (void);                  // key click, card click sound
    void buzzer (void);                 // error buzzer sound
    extern delay_us (unsigned char);

#else

    extern void okbuzzer (void);        // okbuzzer sound
    extern void click (void);           // key click, card click sound
    extern void buzzer (void);          // error buzzer sound

#endif

sbit BUZZER1        = 0xb4;             // buzzer, pin 3.4
sbit BUZZER2        = 0xb5;             // buzzer, pin 3.5
```

8.3.2 buzzer.c

```
#define _BUZZER_

#include <buzzer.h>

/*************************************************************************/
/*      Name:           click                                            */
/*      Description:    sound a small click on the buzzer                */
/*      Calls:          delay_us                                         */
/*      Input:          -                                                */
/*      Return Value:   -                                                */
/*************************************************************************/

void click (void)
{
    unsigned char x;
    x = 5;                                  /* set to loop 10 times    */
    while (x)
    {
        BUZZER1 = 1;
        BUZZER2 = 0;
        delay_us (0xff);                    /* delay 255us             */
        delay_us (0xff);                    /* delay 255us             */
        BUZZER1 = 0;
        BUZZER2 = 1;
        delay_us (0xff);                    /* delay 255us             */
        delay_us (0xff);                    /* delay 255us             */
        x--;                                /* dec. loop counter       */
    }

    BUZZER1 = 0;
    BUZZER2 = 0;
}


/*************************************************************************/
/*      Name:           buzzer                                           */
/*      Description:    sounds error beep on buzzer                      */
/*      Calls:          delay_us                                         */
/*      Input:          -                                                */
/*      Return Value:   -                                                */
/*************************************************************************/

void buzzer (void)
{
    unsigned char x;
    x = 200;                                /* set loop 200 times      */

   while (x)
    {
        BUZZER1 = 1;
        BUZZER2 = 0;

        delay_us (0xff);                    /* delay 255us             */
        delay_us (0xff);                    /* delay 255us             */
        delay_us (0xff);                    /* delay 255us             */

        BUZZER1 = 0;
        BUZZER2 = 1;

        delay_us (0xff);                    /* delay 255us             */
        delay_us (0xff);                    /* delay 255us             */
        delay_us (0xff);                    /* delay 255us             */
        x--;                                /* dec. loop counter       */
    }

    BUZZER1 = 0;
    BUZZER2 = 0;
}
```

```
/*****************************************************************************/
/*      Name:           okbuzzer                                             */
/*      Description:    sounds acknowledge sound on buzzer                   */
/*      Calls:          delay_us                                             */
/*      Input:          -                                                    */
/*      Return Value:   -                                                    */
/*****************************************************************************/

void okbuzzer (void)
{
    unsigned char x;
    unsigned char y;
    y = 3;

    while (y)
    {
        if (y == 3)
            x = 25;                         /* set loop for 25              */
        if (y == 2)
            x = 50;                         /* set loop for 50              */
        if (y == 1)
            x = 100;                        /* set loop for 100             */
        while (x)
        {
            BUZZER1 = 1;
            BUZZER2 = 0;

            delay_us (80*y);
            delay_us (80*y);

            BUZZER1 = 0;
            BUZZER2 = 1;

            delay_us (80*y);
            delay_us (80*y);

            x--;                            /* dec. loop counter            */
        }

        y--;                                /* dec. loop for 3 tone         */
    }

    BUZZER1 = 0;
    BUZZER2 = 0;
}

/*****************************************************************************/
/* End Of Module                                                             */
/*****************************************************************************/
```

8.3.3 eeprom.h

```
#ifdef _I2C_EEPROM_

   #define CHECK_BYTE                        0xaa        // check byte for eeprom

   #define OK                                0x00        // check for status
   #define FAIL                              0x01        // check for status

   #define DATA_SENT_OK                      0x01        // data sent ok
   #define BUS_ERROR                         0x05        // i2c bus error
   #define EEPROM_READ                       0x06        // eeprom read requested
   #define DATA_RECEIVED_OK                  0x07        // eeprom data read ok
   #define TRANSMIT_DATA                    0x0a      // start to transmitt data ok

   #define ENS1_NOTSTA_STO_NOTSI_AA_CR0      0xd4        // i2c control

   #define I2C_TIMEOUT                        0xffff       // counter delay for i2c
timout error

   extern data unsigned char  slave_address;            // slave address
   extern data unsigned char  i2c_transmit_len;         // length of i2c message
   extern xdata unsigned char i2c_data_tx[0x0f];           // i2c data transmit
buffer
   extern xdata unsigned char i2c_data_rx[0x0f];        // i2c data receive buffer
   extern data unsigned char  tx_ok;                    // i2c TX status flag
   extern data unsigned char  buffer_full;              // buffer data flag

   extern void delay_10ms (unsigned char);  // delay time in 10ms increments
   extern void decode_rs232_data (void);    // decode rs-232 data while waiting

   unsigned char i2c_eeprom_write (unsigned char, unsigned char, unsigned char);//
data to e2prom
   unsigned char i2c_eeprom_read (unsigned char, unsigned char);          // data
from e2prom
   unsigned char i2c_eeprom_check (unsigned char, unsigned char, unsigned char);
   int i2c_eeprom_int_read (unsigned char, unsigned char);
   unsigned char i2c_eeprom_int_write (unsigned char, unsigned char, int);
   unsigned char i2c_eeprom_char_write (unsigned char, unsigned char, unsigned
char);
   unsigned char i2c_eeprom_count_write (unsigned char, unsigned char, unsigned
long);
   unsigned long i2c_eeprom_count_read (unsigned char, unsigned char);


#else

   extern unsigned char i2c_eeprom_write (unsigned char, unsigned char, unsigned
char);// data to e2prom
   extern unsigned char i2c_eeprom_read (unsigned char, unsigned char);       //
data from e2prom
   extern unsigned char i2c_eeprom_check (unsigned char, unsigned char, unsigned
char);
   extern int i2c_eeprom_int_read (unsigned char, unsigned char);
   extern unsigned char i2c_eeprom_int_write (unsigned char, unsigned char, int);
   extern unsigned char i2c_eeprom_char_write (unsigned char, unsigned char,
unsigned char);
   extern unsigned char i2c_eeprom_count_write (unsigned char, unsigned char,
unsigned long);
   extern unsigned long i2c_eeprom_count_read (unsigned char, unsigned char);

#endif
```

8.3.4 eeprom.c

```
#define _I2C_EEPROM_

#include <reg652.h>                     // sfrs for 80C652

#include "time.h"                       // delay routines header file
#include "eeprom.h"                     // eeprom routines header file


/*************************************************************************/
/*      Name:           i2c_eeprom_write                                 */
/*      Description:    sets data for i2c eeprom, then sends             */
/*      Input:          eeprom_addr, i2c address of eeprom               */
/*                      eeprom_word_addr, word address of data to be saved */
/*                      eeprom_data, data for eeprom                     */
/*      Return Values:  unsigned char -  0 = storage OK                  */
/*                      unsigned char - >0 = corrupted storage           */
/*      Return Value:   -                                                */
/*************************************************************************/

unsigned char i2c_eeprom_write (eeprom_addr, eeprom_word_addr, eeprom_data)
unsigned char eeprom_addr;
unsigned char eeprom_word_addr;
unsigned char eeprom_data;
{
    xdata unsigned int i = 0;           // i2c timeout counter

    slave_address = eeprom_addr;        // set slave address to eeprom value

    i2c_transmit_len = 2;               // set length of transmission string

    i2c_data_tx[0] = eeprom_word_addr;  // set eeprom address to use
    i2c_data_tx[1] = eeprom_data;       // set data to save in eeprom address
    i2c_data_tx[2] = 0;                 // end of string

    STA = 1;                            // set i2c transmition in motion

    tx_ok = TRANSMIT_DATA;              // set transmission status variable

    while ((tx_ok != DATA_SENT_OK) && (i < I2C_TIMEOUT))
    {
        //if (buffer_full == 1)         // check for rs232 data
        //  decode_rs232_data();        // if so, decode it
        i++;                            // increment timeout counter
    }

    if (tx_ok != DATA_SENT_OK)
    {
        tx_ok = BUS_ERROR;              // reinitilase i2c bus
        S1CON = ENS1_NOTSTA_STO_NOTSI_AA_CR0;
        SI = 0;                         // clear i2c interrupt flag
        return FAIL;
    }
    else
    {
        delay_10ms (1);                 // allow eeprom writing time, 7ms
        return OK;
    }
}
```

```
/*************************************************************************/
/*      Name:           i2c_eeprom_read                                  */
/*      Description:    reads data from i2c eeprom                       */
/*      Input:          eeprom_addr, i2c address of eeprom               */
/*                      eeprom_word_addr, word address of data to be read*/
/*      Return Value:   data byte from eeprom                            */
/*************************************************************************/

unsigned char i2c_eeprom_read (eeprom_addr, eeprom_word_addr)
unsigned char eeprom_addr;
unsigned char eeprom_word_addr;
{
    xdata unsigned int i = 0;           // i2c timeout counter

    slave_address = eeprom_addr;        // set slave address to eeprom value

    i2c_transmit_len = 1;               // set length of transmission string

    i2c_data_tx[0] = eeprom_word_addr;  // set eeprom address to use
    i2c_data_tx[1] = 0;                 // end of string

    STA = 1;                            // set i2c transmition in motion

    tx_ok = EEPROM_READ;                // set transmission status variable

    while ((tx_ok != DATA_RECEIVED_OK) && (i < I2C_TIMEOUT))
    {
        //if (buffer_full == 1)         // check for rs232 data
        //  decode_rs232_data();        // if so, decode it
        i++;                            // increment timeout counter
    }

    if (tx_ok == DATA_RECEIVED_OK)
        return (i2c_data_rx[0]);
    else
    {
        tx_ok = BUS_ERROR;              // reinitilase i2c bus
        S1CON = ENS1_NOTSTA_STO_NOTSI_AA_CR0;
        SI = 0;                         // clear i2c interrupt flag
        return (0);                     // return null data
    }
}


/*************************************************************************/
/*  Function:       i2c_eeprom_check                                     */
/*  Opertion:       checks data in given address is correct              */
/*  Inputs:         unsigned char - base address of eeprom               */
/*                  unsigned char - start address for checking           */
/*                  unsigned char - number  of bytes to check            */
/*  Return Values:  unsigned char -  0 = storage OK                      */
/*                  unsigned char - >0 = corrupted storage               */
/*************************************************************************/

unsigned char i2c_eeprom_check (eeprom_addr, address_start, no_bytes)
unsigned char eeprom_addr;
unsigned char address_start;
unsigned char no_bytes;
{
    xdata unsigned char checksum = CHECK_BYTE;  // start with check byte

    do
    {
        checksum ^= i2c_eeprom_read (eeprom_addr, address_start++);
        no_bytes--;
    } while (no_bytes > 0);                     // get all bytes required

    if (checksum != i2c_eeprom_read (eeprom_addr, address_start))
        return FAIL;
    else
        return OK;
}
```

```
/*********************************************************************/
/*  Name:           i2c_eeprom_int_read                              */
/*  Description:    reads a two byte integer value from the eeprom   */
/*  Inputs:         unsigned char - base address for i2c eeprom      */
/*                  unsigned char - start address for reading        */
/*  Return Values:  int - return value of integer read               */
/*********************************************************************/

int i2c_eeprom_int_read (unsigned char eeprom_addr, unsigned char address)
{
    xdata int integer_read;

    integer_read = i2c_eeprom_read (eeprom_addr, address++);          // read low
byte
    integer_read += i2c_eeprom_read (eeprom_addr, address) * 256;    // read hi
byte

    return integer_read;                              // return read value
}


/*********************************************************************/
/*  Name:           i2c_eeprom_int_write                             */
/*  Description:    writes a two byte integer value to the eeprom, plus*/
/*                  check digit, check written value, and returns value*/
/*  Inputs:         unsigned char - base address for eeprom          */
/*                  unsigned char - start address for writing        */
/*                  int - data for writting                          */
/*  Return Values:  unsigned char - 0 =  written ok                  */
/*                  unsigned char - !0 = not written ok              */
/*********************************************************************/

unsigned char i2c_eeprom_int_write (eeprom_addr, address, int data_word)
unsigned char eeprom_addr;
unsigned char address;
{
    xdata unsigned char integer_write;
    xdata unsigned char checksum = CHECK_BYTE;  // start with check byte


    integer_write = data_word % 256;            // get hi byte
    checksum ^= integer_write;                  // calc. check byte
    i2c_eeprom_write (eeprom_addr, address++, integer_write);   // write hi byte
    integer_write = data_word / 256;            // get low byte
    checksum ^= integer_write;                  // calc. check byte
    i2c_eeprom_write (eeprom_addr, address++, integer_write);   // write low byte
    i2c_eeprom_write (eeprom_addr, address, checksum);          // write check sum

    address -= 2;
    if (i2c_eeprom_int_read (eeprom_addr, address) == data_word
        && i2c_eeprom_check (eeprom_addr, address, 2) == OK)
        return OK;                              // check data written is ok
    else
        return FAIL;                            // return non zero if not
}
```

```
/*********************************************************************/
/*  Name:           i2c_eeprom_char_write                            */
/*  Description:    writes a byte to the eeprom, plus check digit,   */
/*                  check written value, and returns value           */
/*  Inputs:         unsigned char   - base address of eeprom         */
/*                  unsigned char    - start address for reading     */
/*                  unsigned char   -  data for writting             */
/*  Return Values:  unsigned char   -  0 =  written ok               */
/*                  unsigned char   - !0 = not written ok            */
/*********************************************************************/

unsigned char i2c_eeprom_char_write (eeprom_addr, address, data_byte)
unsigned char eeprom_addr;
unsigned char address;
unsigned char data_byte;
{
    unsigned char checksum = CHECK_BYTE;            // start with check byte

    checksum ^= data_byte;                          // calc. check byte

    i2c_eeprom_write (eeprom_addr, address++, data_byte);   // write byte
    i2c_eeprom_write (eeprom_addr, address, checksum);      // write check sum

    address--;

    if ((i2c_eeprom_read (eeprom_addr, address) == data_byte)
        && (i2c_eeprom_check (eeprom_addr, address, 1) == OK))
        return OK;                                  // check data written is OK
    else
        return FAIL;                                // return non zero if not
}


/*********************************************************************/
/*  Name:           i2c_eeprom_count_write                           */
/*  Description:    writes a three byte value to the eeprom, plus    */
/*                  check digit, check written value, and returns value*/
/*  Inputs:         unsigned char   - base address for eeprom        */
/*                  unsigned char    - start address for writing     */
/*                  unsigned long   - data for writing               */
/*  Return Values:  unsigned char   -  0 =  written ok               */
/*                  unsigned char   - !0 = not written ok            */
/*********************************************************************/

unsigned char i2c_eeprom_count_write (eeprom_addr, address, data_word)
unsigned char eeprom_addr;
unsigned char address;
unsigned long data_word;
{
    unsigned long old_data;
    unsigned char count_write;
    unsigned char checksum = CHECK_BYTE;        // start with check byte

    data_word &= 0x00ffffffUL;                  // ignore last byte of long word
    old_data = data_word;                       // save data for later calc.
    count_write = data_word % 65536UL;          // find lo byte value
    checksum ^= count_write;                    // calc. rolling checkbyte
    i2c_eeprom_write (eeprom_addr, address++, count_write);     // save lo byte
    data_word /= 256UL;                         // knock off lo byte
    count_write = data_word % 256UL;            // find mid byte value
    checksum ^= count_write;                    // calc. rolling checkbyte
    i2c_eeprom_write (eeprom_addr, address++, count_write);      // save mid byte
value
    count_write = data_word / 256UL;            // find hi byte
    checksum ^= count_write;                    // calc. rolling checkbyte
    i2c_eeprom_write (eeprom_addr, address++, count_write);     // save hi byte
    i2c_eeprom_write (eeprom_addr, address, checksum);          // save checkbyte

    address -= 3;
    if (i2c_eeprom_count_read (eeprom_addr, address) == old_data
        && i2c_eeprom_check (eeprom_addr, address, 3) == OK)
        return OK;                                  // check data written is OK
    else
        return FAIL;                                // if not return zero
}
```

```
/*********************************************************************/
/*  Name:           i2c_eeprom_count_read                            */
/*  Description:    reads a three byte value from the eeprom         */
/*  Inputs:         unsigned char   - start address for reading      */
/*  Return Values:  unsigned long   - return value of read           */
/*********************************************************************/

unsigned long i2c_eeprom_count_read (eeprom_addr, address)
unsigned char eeprom_addr;
unsigned char address;
{
    unsigned long count_read;

    count_read =   (unsigned long) i2c_eeprom_read (eeprom_addr, address++);
    count_read  +=  ((unsigned  long)  i2c_eeprom_read  (eeprom_addr,  address++)  *
256UL);
    count_read  +=  ((unsigned  long)  i2c_eeprom_read  (eeprom_addr,  address)     *
65536UL);

    return count_read;
}


/*********************************************************************/
/* End Of Module                                                     */
/*********************************************************************/
```

8.3.5 galvos. h

```
#ifdef _GALVOS_

   #ifndef PIO2
       #define PIO2 XBYTE[0x0004]           // base address of PIO 2
   #endif

   sbit A0_PIO2 = 0x91;                     // address 0 pin for PIO 2
   sbit A1_PIO2 = 0x92;                     // address 1 pin for PIO 2

   void set_left_dtoa (unsigned char);      // set left D/A value for left galvo
   void set_right_dtoa (unsigned char);     // set right D/A value for right galvo
   void set_dtoas (unsigned char);          // set both D/A values for both galvos

#else

   extern void set_left_dtoa (unsigned char);    // set left D/A value for left
galvo
   extern void set_right_dtoa (unsigned char);  // set right D/A value for right
galvo
   extern void set_dtoas (unsigned char);        // set both D/A values for both
galvos

#endif
```

8.3.6 galvos.c

```
#define _GALVOS_

#include <absacc.h>

#include "galvos.h"        // galvo control functions


/*************************************************************************/
/*      Name:           set_left_dtoa                                    */
/*      Description:    send set value to d to a converter 1             */
/*      Input:          unsigned char, value                             */
/*      Return Value:   -                                                */
/*************************************************************************/

void set_left_dtoa (unsigned char value)
{
    A0_PIO2 = 0;
    A1_PIO2 = 0;                        // address lines for left dtoa data

    PIO2 = value;                       // set dtoa value

    A1_PIO2 = 1;                        // address line for control port
    PIO2 |= 0x10;                       // send enable line high
    PIO2 &= 0xef;                       // reset enable line low
}


/*************************************************************************/
/*      Name:           set_right_dtoa                                   */
/*      Description:    send set value to d to a converter 2             */
/*      Input:          unsigned char, value                             */
/*      Return Value:   -                                                */
/*************************************************************************/

void set_right_dtoa (unsigned char value)
{
    A1_PIO2 = 0;
    A0_PIO2 = 1;                        // address line for data port

    PIO2 = value;                       // set dtoa value

    A1_PIO2 = 1;
    A0_PIO2 = 0;                        // address line for control port
    PIO2 |= 0x20;                       // send enable line high
    PIO2 &= 0xdf;                       // reset enable line low
}


/*************************************************************************/
/*      Name:           set_dtoas                                        */
/*      Description:    send set value to d to a converters 1 and 2      */
/*      Input:          unsigned char, value                             */
/*      Return Value:   -                                                */
/*************************************************************************/

void set_dtoas (unsigned char value)
{
   set_left_dtoa  (value);             // set value of left d/a  - 1
   set_right_dtoa (value);             // set value of right d/a - 2
}


/*************************************************************************/
/* End Of Module                                                         */
/*************************************************************************/
```

8.3.7 i2c.h

```
#ifdef _I2C_

   #define ENS1_NOTSTA_STO_NOTSI_AA_CR0        0xd4 // i2c control
   #define ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0     0xc4 // i2c control
   #define ENS1_NOTSTA_NOTSTO_NOTSI_NOTAA_CR0  0xc0 // i2c control
   #define ENS1_STA_NOTSTO_NOTSI_AA_CR0        0xe4 // i2c control

   #define DATA_SENT_OK         0x01    // data sent ok
   #define NACK_ADDRESS         0x02    // nack received after address tx
   #define NACK_POST_ADDRESS    0x03    // nack during transmission
   #define LOST_ARBITRATION     0x04    // lost arbitration on transmit
   #define BUS_ERROR            0x05    // i2c bus error
   #define EEPROM_READ          0x06    // eeprom read requested
   #define DATA_RECEIVED_OK     0x07    // eeprom data read ok
   #define IO_READ              0x08    // read i2c i/o
   #define TRANSMIT_DATA        0x0a    // start to transmitt data ok

   #define GENERAL_CALL         0x00    // i2c general call address
   #define READ                 0x01    // read bit in slave address
   #define WRITE                0x00    // write bit in slave address

   #define OK                   0x00    // check for status
   #define FAIL                 0x01    // check for status

   #define ETX                  0x03    // end of text

   #define I2C_TIMEOUT          0xffff  // counter timeout delay for i2c routines


   extern void decode_rs232_data (void);              // decode rs-232 data while
waiting
   extern data unsigned char  buffer_full;          // buffer data flag

   xdata unsigned char i2c_data_tx[0x0f];           // i2c data transmit buffer
   xdata unsigned char i2c_data_rx[0x0f];           // i2c data receive buffer
   data unsigned char  slave_address;               // slave address
   data unsigned char  tx_ok                = 0;    // i2c TX status flag
   data unsigned char  i2c_transmit_len     = 0;    // length of i2c message
   data unsigned char  i2c_rx_data_count    = 0;    // i2c data count for RX
   data unsigned char  i2c_tx_data_count    = 0;    // i2c data count for TX

   void i2c_int (void);                             // i2c interrupt routine
   unsigned char i2c_dac (unsigned char, unsigned char, unsigned char);       //
data to dacs
   unsigned char i2c_e2pot (unsigned char, unsigned char, unsigned char);    //
data to e2pots
   unsigned char i2c_io_read (unsigned char);       // data from i2c io


#else

   extern void i2c_int (void);                      // i2c interrupt routine
   extern unsigned char i2c_dac (unsigned char, unsigned char, unsigned char);  //
data to dacs
   extern unsigned char i2c_e2pot (unsigned char, unsigned char, unsigned char);//
data to e2pots
   extern unsigned char i2c_io_read (unsigned char);// data from i2c io

#endif
```

8.3.8 i2c.c

```
#define _I2C_

#include <reg652.h>        // special function registers of 80C652

#include "i2c.h"           // i2c routines header file


/*************************************************************************/
/*      Name:           i2c_int                                          */
/*      Description:    i2c serial interrupt routine                     */
/*      Input:          -                                                */
/*      Return Value:   -                                                */
/*************************************************************************/

void i2c_int(void) interrupt 5
{
    switch (S1STA)
    {

/************************************************************/
/* GENERALMODE                                              */
/************************************************************/
        /*******************************************************/
        /* State:  00h - Bus error                             */
        /* Action: enter not addressed slave mode & relesae bus */
        /*         STO reset.                                  */
        /*******************************************************/
        case 0x00:  tx_ok = BUS_ERROR;
                    S1CON = ENS1_NOTSTA_STO_NOTSI_AA_CR0;
                    break;

/************************************************************/
/* MASTER TRANSMITTER MODE                                  */
/************************************************************/
        /*******************************************************/
        /* State:  08h - Start condition has been transmitted   */
        /* Action: tx slave address, receive ack bit            */
        /*******************************************************/
        case 0x08:  S1DAT = slave_address;
                    i2c_tx_data_count = 0;
                    S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                    break;

        /*******************************************************/
        /* State:  10h - Repeated start has been transmitted    */
        /* Action: tx slave address, receive ack bit            */
        /*******************************************************/
        case 0x10:  S1DAT = slave_address;
                    i2c_tx_data_count = 0;
                    S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                    break;

        /*******************************************************/
        /* State:  18h - previous state 8 or 10, slave address  */
        /*         was transmitted, ack bit received            */
        /* Action: send first data byte, ack received           */
        /*******************************************************/
        case 0x18:  if (tx_ok == TRANSMIT_DATA ||
                        tx_ok == LOST_ARBITRATION ||
                        tx_ok == EEPROM_READ)
                        S1DAT = i2c_data_tx[i2c_tx_data_count++];
                    S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                    break;

        /*******************************************************/
        /* State:  20h - previous state 8 or 10, slave address  */
        /*         was transmitted, nack received               */
        /* Action: Transmit STOP condition                      */
        /*******************************************************/
        case 0x20:  tx_ok = NACK_ADDRESS;
                    S1CON = ENS1_NOTSTA_STO_NOTSI_AA_CR0;
```

```
                break;

        /*******************************************************/
        /* State:  28h - S1DAT transmitted, ack received       */
        /* Action: Transmit next data, if data is last byte    */
        /*         transmit STOP condition                     */
        /*******************************************************/
        case 0x28:  if (tx_ok == TRANSMIT_DATA || tx_ok == LOST_ARBITRATION)
                    {
                        if (i2c_tx_data_count < i2c_transmit_len)
                        {
                            S1DAT = i2c_data_tx[i2c_tx_data_count++];
                        S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                        }
                        else
                        {
                            tx_ok = DATA_SENT_OK;
                            S1CON = ENS1_NOTSTA_STO_NOTSI_AA_CR0;
                        }
                    }

                    if (tx_ok == EEPROM_READ)   /* now read data form eeprom */
                    {
                        slave_address += READ;
                        S1CON = ENS1_STA_NOTSTO_NOTSI_AA_CR0;
                    }

                    break;

        /*******************************************************/
        /* State:  30h - S1DAT transmitted, nack received      */
        /* Action: transmit STOP condition                     */
        /*******************************************************/
        case 0x30:  tx_ok = NACK_POST_ADDRESS;
                    S1CON = ENS1_NOTSTA_STO_NOTSI_AA_CR0;
                    break;

        /*******************************************************/
        /* State:  38h - lost arbitration in slave address wri. */
        /*         or data                                     */
        /* Action: release bus, enter not addressed slave mode  */
        /*         START transmitted when bus free             */
        /*******************************************************/
        case 0x38:  tx_ok = LOST_ARBITRATION;
                    S1CON = ENS1_STA_NOTSTO_NOTSI_AA_CR0;
                    break;



/*************************************************************/
/* MASTER RECEIVER MODE                                      */
/*************************************************************/
        /*******************************************************/
        /* State:  40h - previous state 8 or 10, slave address  */
        /*         and read transmitted, ack received          */
        /* Action: data will be received and ack returned      */
        /*******************************************************/
        case 0x40:  i2c_rx_data_count = 0;
                    i2c_data_rx[0] = 0;
                    S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                    break;

        /*******************************************************/
        /* State:  48h - slave address and read transmitted,    */
        /*         nack received                               */
        /* Action: STOP condition generated                    */
        /*******************************************************/
        case 0x48:  S1CON = ENS1_NOTSTA_STO_NOTSI_AA_CR0;
                    break;

        /*******************************************************/
        /* State:  50h - data received, ack returned           */
        /* Action: read S1DAT. If last data send nack else     */
        /*         send ack                                    */
        /*******************************************************/
        case 0x50:  i2c_data_rx[i2c_rx_data_count++] = S1DAT;
                    if (i2c_rx_data_count < 1)
```

```
                    S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                else
                {
                    i2c_data_rx[i2c_rx_data_count] = 0;
                    S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_NOTAA_CR0;
                }
                break;

        /*********************************************************/
        /* State:  58h - data received, nack returned            */
        /* Action: read S1DAT and generate STOP condition        */
        /*********************************************************/
        case 0x58:  i2c_data_rx[i2c_rx_data_count++] = S1DAT;
                    i2c_data_rx[i2c_rx_data_count] = 0;
                    tx_ok = DATA_RECEIVED_OK;
                    S1CON = ENS1_NOTSTA_STO_NOTSI_AA_CR0;
                    break;


/*************************************************************/
/* SLAVE RECEIVER MODE                                       */
/*************************************************************/
        /*********************************************************/
        /* State:  60h - own address received and write, send    */
        /*         ack                                           */
        /* Action: data will be received, ack returned           */
        /*********************************************************/
        case 0x60:  i2c_rx_data_count = 0;
                    i2c_data_rx[0] = 0;
                    S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                    break;

        /*********************************************************/
        /* State:  68h - lost arbitration in address and r/w as  */
        /*         master, own address and write received, send  */
        /*         ack                                           */
        /* Action: data will be received and ack returned. STA   */
        /*         is set to restart master mode after the bus   */
        /*         is free again                                 */
        /*********************************************************/
        case 0x68:  i2c_rx_data_count = 0;
                    i2c_data_rx[0] = 0;
                    S1CON = ENS1_STA_NOTSTO_NOTSI_AA_CR0;
                    break;

        /*********************************************************/
        /* State:  70h - general call received, ack returned     */
        /* Action: data will be received, ack returned           */
        /*********************************************************/
        case 0x70:  i2c_rx_data_count = 0;
                    i2c_data_rx[0] = 0;
                    S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                    break;

        /*********************************************************/
        /* State:  78h - lost arbitration in slave and r/w as    */
        /*         master, general call received, ack returned   */
        /* Action: data will be received and ack returned. STA   */
        /*         is set to restart master mode after the bus   */
        /*         is free again                                 */
        /*********************************************************/
        case 0x78:  tx_ok = LOST_ARBITRATION;
                    i2c_rx_data_count = 0;
                    i2c_data_rx[0] = 0;
                    S1CON = ENS1_STA_NOTSTO_NOTSI_AA_CR0;
                    break;

        /*********************************************************/
        /* State:  80h - previously slave addressed ok, data     */
        /*         received, ack returned                        */
        /* Action: read data. if last data, excess will be       */
        /*         received and nack returned, else next data    */
        /*         received and ack returned                     */
        /*********************************************************/
        case 0x80:  i2c_data_rx[i2c_rx_data_count++] = S1DAT;
                    if (i2c_data_rx[i2c_rx_data_count-2] != ETX)
                        S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_NOTAA_CR0;
```

```
                else
                {
                    i2c_data_rx[i2c_rx_data_count] = 0;
                    S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                }
                break;

        /*********************************************************/
        /* State:  88h - previously slave addressed ok, data     */
        /*         received, nack returned                       */
        /* Action: dont save data, enter not addressed slave     */
        /*         mode. Recognition of own address. General     */
        /*         call recognised if applicable.                */
        /*********************************************************/
        case 0x88:  S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                    break;

        /*********************************************************/
        /* State:  90h - previously addressed with general call */
        /*         data received, ack returned                   */
        /* Action: read data. After general call, only one byte */
        /*         will be received with ack, 2nd byte will be  */
        /*         nack. data will be received and nack returned*/
        /*********************************************************/
        case 0x90:  i2c_data_rx[i2c_rx_data_count++] = S1DAT;
                    S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                    break;

        /*********************************************************/
        /* State:  98h - previously addressed with general call */
        /*         data received nack returned                   */
        /* Action: don't save data, enter not addressed slave   */
        /*         mode. Recognition of own address & general   */
        /*         call if applicable                            */
        /*********************************************************/
        case 0x98:  S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                    break;

        /*********************************************************/
        /* State:  a0h - STOP or repeated START received while  */
        /*         still addressed as slave                      */
        /* Action: don't save data, enter not addressed slave   */
        /*         mode. recognise own address and general call */
        /*         if applicable                                 */
        /*********************************************************/
        case 0xa0:  i2c_data_rx[i2c_rx_data_count] = 0;
                    // rxdata();
                   if (tx_ok == TRANSMIT_DATA)
                        S1CON = ENS1_STA_NOTSTO_NOTSI_AA_CR0;
                    else
                        S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                    break;


/*************************************************************/
/* SLAVE TRANSMITTER MODE                                   */
/*************************************************************/
        /*********************************************************/
        /* State:  a8h - slave address received and read, ack   */
        /*         sent                                          */
        /* Action: data will be transmitted, ack received       */
        /*********************************************************/
        case 0xa8:  S1DAT = i2c_data_tx[i2c_tx_data_count++];
                    S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                    break;

        /*********************************************************/
        /* State:  b0h - lost arbitration in slave address and  */
        /*         r/w as master. own address received and read */
        /*         ack returned                                  */
        /* Action: data will be transmitted. ack received. STA  */
        /*         SET to restart master mode after the bus is  */
        /*         free again                                    */
        /*********************************************************/
        case 0xb0:  S1DAT = i2c_data_tx[i2c_tx_data_count++];
                    S1CON = ENS1_STA_NOTSTO_NOTSI_AA_CR0;
                    break;
```

```
        /*****************************************************/
        /* State:  b8h - data transmitted, ack received      */
        /* Action: data will be transmitted, ack received    */
        /*****************************************************/
        case 0xb8:  S1DAT = i2c_data_tx[i2c_tx_data_count++];
                    S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                    break;

        /*****************************************************/
        /* State:  c0h - data transmitted, nack received     */
        /* Action: enter not addressed slave mode            */
        /*****************************************************/
        case 0xc0:  S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                    break;

        /*****************************************************/
        /* State:  c8h - last data has been transmitted (AA=0)  */
        /*         ack received                              */
        /* Action: enter not addressed slave mode            */
        /*****************************************************/
        case 0xc8:  S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;
                    break;

        default:    SI = 0;         // clear i2c interrupt flag
    }
}


/*************************************************************************/
/*      Name:           i2c_dac                                          */
/*      Description:    sets data for i2c dac, then sends                */
/*      Input:          dac_addr, i2c address of dac                     */
/*                      dac_instr, instruction byte for dac sub address  */
/*                      dac_data, data for dac output                    */
/*      Return Value:   -                                                */
/*************************************************************************/

unsigned char i2c_dac (dac_addr, dac_instr, dac_data)
unsigned char dac_addr;
unsigned char dac_instr;
unsigned char dac_data;
{
    xdata unsigned int i = 0;           // i2c timeout counter

    slave_address = dac_addr;           // set slave address to dac value
    i2c_transmit_len = 2;               // set length of transmission string

    i2c_data_tx[0] = dac_instr;         // set dac number to use
    i2c_data_tx[1] = dac_data;          // set data for chosen dac number
    i2c_data_tx[2] = 0;                 // end of string

    STA = 1;                            // set i2c transmition in motion
    tx_ok = TRANSMIT_DATA;              // set transmission status variable

    while ((tx_ok != DATA_SENT_OK) && (i < I2C_TIMEOUT))
    {
        //if (buffer_full == 1)         // check for rs232 data
        //  decode_rs232_data();        // if so, decode it
        i++;                            // increment timeout counter
    }

    if (tx_ok != DATA_SENT_OK)
    {
        tx_ok = BUS_ERROR;              // reinitilase i2c bus
        S1CON = ENS1_NOTSTA_STO_NOTSI_AA_CR0;
        SI = 0;                         // clear i2c interrupt flag
        return FAIL;
    }
    else
        return OK;
}
```

```
/*************************************************************************/
/*      Name:           i2c_e2pot                                        */
/*      Description:    sets data for i2c e2pot, then sends              */
/*      Input:          e2pot_addr, i2c address of e2pot                 */
/*                      e2pot_instr, instruction byte for e2pot          */
/*                      e2pot_data, data for e2pot setting               */
/*      Return Value:   -                                                */
/*************************************************************************/

unsigned char i2c_e2pot (e2pot_addr, e2pot_instr, e2pot_data)
unsigned char e2pot_addr;
unsigned char e2pot_instr;
unsigned char e2pot_data;
{
    xdata unsigned int i = 0;           // i2c timeout counter

    slave_address    = e2pot_addr;      // set slave address to e2pot value
    e2pot_instr     += 0xa0;            // set command to write volatile wiper
    e2pot_data       = 63 - e2pot_data; // 63 is maximum val, this inverts wiper
response
    i2c_transmit_len = 2;               // set length of transmission string

    i2c_data_tx[0] = e2pot_instr;       // set instruction to use
    i2c_data_tx[1] = e2pot_data;        // set data for chosen instruction
    i2c_data_tx[2] = 0;                 // end of string

    STA = 1;                            // set i2c transmition in motion
    tx_ok = TRANSMIT_DATA;              // set transmission status variable

    while ((tx_ok != DATA_SENT_OK) && (i < I2C_TIMEOUT))
    {
        //if (buffer_full == 1)         // check for rs232 data
        //  decode_rs232_data();        // if so, decode it
        i++;                            // increment timeout counter
    }

    if (tx_ok != DATA_SENT_OK)
    {
        tx_ok = BUS_ERROR;              // reinitilase i2c bus
        S1CON = ENS1_NOTSTA_STO_NOTSI_AA_CR0;
        SI = 0;                         // clear i2c interrupt flag
        return FAIL;
    }
    else
        return OK;

    //e2pot_instr += 0x40;              // set to save WCR value in NVR
    //slave_address = e2pot_addr;       // reset slave address to e2pot value
    //i2c_transmit_len = 2;             // reset length of transmission string
    //i2c_data_tx[0] = e2pot_instr;     // reset instruction to use
    //i2c_data_tx[1] = e2pot_data;      // set data for chosen instruction
    //i2c_data_tx[2] = 0;               // end of string
    //STA = 1;                          // set i2c transmition in motion
    //tx_ok = TRANSMIT_DATA;            // set transmission status variable
    //while(tx_ok != DATA_SENT_OK);     // wait for data sent

    //delay_10ms (1);
}
```

```
/*************************************************************************/
/*      Name:           i2c_io_read                                      */
/*      Description:    reads data from i2c io                           */
/*      Input:          io_addr, i2c address of io                       */
/*      Return Value:   data byte from io                                */
/*************************************************************************/

unsigned char i2c_io_read (unsigned char io_addr)
{
   xdata unsigned int i = 0;           // i2c timeout counter

    slave_address = io_addr;            // set slave address to io value
    slave_address += READ;              // set to read i2c io

    i2c_transmit_len = 0;               // set length of transmission string
    i2c_data_tx[0] = 0;                 // end of string
    STA = 1;                            // set i2c transmition in motion

    tx_ok = IO_READ;                    // set transmission status variable

    while ((tx_ok != DATA_RECEIVED_OK) && (i < I2C_TIMEOUT))
    {
        //if (buffer_full == 1)         // check for rs232 data
        //  decode_rs232_data();        // if so, decode it
        i++;                            // increment timeout counter
    }

    if (tx_ok == DATA_RECEIVED_OK)
       return (i2c_data_rx[0]);
    else
    {                                   // maybe best to retry here!!!!!
        tx_ok = BUS_ERROR;              // reinitilase i2c bus
        S1CON = ENS1_NOTSTA_STO_NOTSI_AA_CR0;
        SI = 0;                         // clear i2c interrupt flag
        return (0);                     // return null data
    }
}


/*************************************************************************/
/* End Of Module                                                         */
/*************************************************************************/
```

8.3.9 lcd_text.h

```
#ifdef _LCD_TEXT_

    #ifndef PORT_A
        #define PORT_A  XBYTE[0x0000]       // port b address of 8255
    #endif

    #ifndef PORT_C
        #define PORT_C  XBYTE[0x0002]       // port c address of 8255
    #endif

    extern void delay_10ms (unsigned char); // delay time in 10ms increments
    extern void delay_us (unsigned char);   // delay time in us increments

    void initialise_lcd (void);
    void send_lcd_control_byte (unsigned char);
    void send_lcd_data_byte (char);
    void print_lcd (char*, char, char);
    void clear_lcd (void);

#else

    extern void initialise_lcd (void);
    extern void send_lcd_control_byte (unsigned char);
    extern void send_lcd_data_byte (char);
    extern void print_lcd (char*, char, char);
    extern void clear_lcd (void);

    #define CURRENT             0           // current position of LCD cursor
    #define LINE_1              1           // line 1 of LCD
    #define LINE_2              2           // line 2 of LCD
    #define LINE_3              3           // line 3 of LCD
    #define LINE_4              4           // line 4 of LCD

#endif
```

8.3.10 lcd_text.c

```
#define _LCD_TEXT_

#include <absacc.h>
#include <string.h>        // string functions
#include "lcd_text.h"      // lcd text routines header file


/*************************************************************************/
/*      Name:           initialise_lcd                                   */
/*      Description:    Initialises lcd                                  */
/*      Input:          -                                                */
/*      Return Value:   -                                                */
/*************************************************************************/

void initialise_lcd (void)
{
    send_lcd_control_byte (48);                 // initialise lcd
    send_lcd_control_byte (48);                 // initialise lcd
    send_lcd_control_byte (48);                 // initialise lcd
    send_lcd_control_byte (60);                 // set function mode
    send_lcd_control_byte (8);                  // display off
    send_lcd_control_byte (1);                  // clear display
    send_lcd_control_byte (6);                  // entry mode
    send_lcd_control_byte (12);                 // display on
    send_lcd_control_byte (1);                  // clear display
    send_lcd_control_byte (2);                  // return home
}


/*************************************************************************/
/*      Name:           send_lcd_control_byte                            */
/*      Description:    sends a single control byte to the lcd           */
/*      Input:          value to be sent to lcd as control byte -        */
/*                      control_value (unsigned char)                    */
/*      Return Value:   -                                                */
/*************************************************************************/

void send_lcd_control_byte (unsigned char control_value)
{
    PORT_C &= 0x1f;                             // clear RS, R/W and E

    PORT_A = control_value;                     // send value to port A

    PORT_C |= 0x20;                             // set E high
    PORT_C &= 0xdf;                             // set E low

    if (control_value < 0x04)
       delay_10ms (1);
    else
       delay_us (110);                         // settling delay, 110us
}


/*************************************************************************/
/*      Name:           send_lcd_data_byte                               */
/*      Description:    sends a single data byte to the lcd              */
/*      Input:          value to be sent to lcd as data byte - data_value*/
/*      Return Value:   -                                                */
/*************************************************************************/

void send_lcd_data_byte (char data_value)
{
    PORT_C &= 0x1f;                             // clear RS, R/W and E
    PORT_C |= 0x80;                             // set RS high
    PORT_A = data_value;                        // send value to port A

    PORT_C |= 0x20;                             // set E high
    PORT_C &= 0xdf;                             // set E low

    delay_us (110);                             // settling delay, 110us
}
```

```
/*************************************************************************/
/*      Name:           print_lcd                                        */
/*      Description:    print specified string on specified line of display*/
/*      Input:          text string - print_text, line number - line_no  */
/*      Return Value:   -                                                */
/*************************************************************************/

void print_lcd (char* print_text, char line_no, char char_no)
{
    unsigned char offset ;
    unsigned char char_count ;
    unsigned char i ;

    switch (line_no)               // set character offset position on display
    {
       case  1:    offset = 128 ;
                   break ;
       case  2:    offset = 192 ;
                   break ;
       case  3:    offset = 148 ;
                   break ;
       case  4:    offset = 212 ;
                   break ;
       default:    offset =   0 ;
                   break ;
    }

    if (offset != 0)
    {
       offset += char_no ;
        send_lcd_control_byte (offset);
    }

    char_count = strlen (print_text);

    for (i = 0 ; i < char_count ; i++)
        send_lcd_data_byte (print_text[i]);
}


/*************************************************************************/
/*      Name:           clear_lcd                                        */
/*      Description:    clears the lcd screen                            */
/*      Input:          -                                                */
/*      Return Value:   -                                                */
/*************************************************************************/

void clear_lcd (void)
{
    send_lcd_control_byte (1);
}


/*************************************************************************/
/* End Of Module                                                         */
/*************************************************************************/
```

8.3.11 rs232.h

```
#ifdef _RS232_

   #define SOH
         0x01
      // start of header - (transmission)
   #define ETX
      0x03                                                    // end of text
   #define EOT
         0x04
      // end of transmission
   #define SUB
         0x1b
      // substitute - transparancey byte

   extern data unsigned char  buffer_full;                  // buffer data flag
   extern data unsigned char  in_buff_len;          // in buffer length
   extern xdata unsigned char rs232_in_buffer[];           // rs232 input buffer

   xdata unsigned char rs232_out_buffer[0x1f];      // rs232 output buffer
   data unsigned char  serial_int_count = 0;          // serial int. routine RX
   data unsigned char  serial_out_count = 0;          // serial int. routine TX

   void serial_int (void);                               // serial port interrupt
   void send_rs232_data (unsigned char*, unsigned char);    // send data string to
rs232

#else

   extern void serial_int (void);                          //      serial      port
interrupt
   extern void send_rs232_data (unsigned char*, unsigned char);// send data string
to rs232

#endif
```

8.3.12 rs232.c

```
#define _RS232_

#include <reg652.h>        // special function registers for 80C652

#include "rs232.h"         // serial communications functions


/*************************************************************************/
/*      Name:           serial_int                                       */
/*      Description:    serial interrupt routine                         */
/*      Input:          -                                                */
/*      Return Value:   -                                                */
/*************************************************************************/

void serial_int (void) interrupt 4
{
    if (RI == 1)                        // if data received flag is set
    {
        if (buffer_full == 1)           // if full buffer
        {
            buffer_full = 0;            // reset to empty buffer flag
            rs232_in_buffer[0] = 0;     // clear buffer
            serial_int_count = 0;       // clear in byte count
            in_buff_len = 0;            // clear in buffer length to 0
        }

        if (SBUF != EOT)                      // if input buffer is not end of
transmission
        {
            in_buff_len++;              // count next byte
            rs232_in_buffer[serial_int_count++] = SBUF; // save next byte
            rs232_in_buffer[serial_int_count] = 0;        // re-terminate end of
string
        }

        if (SBUF == EOT)                    // if input buffer has reached end of
transmission
        {
            in_buff_len++;              // count next byte
            rs232_in_buffer[serial_int_count++] = SBUF; // save next byte
            rs232_in_buffer[serial_int_count] = 0;        // re-terminate end of
string
            serial_int_count = 0;       // reset in counter
            buffer_full = 1;            // set flag to tell in buffer is full
        }

        RI = 0;                         // reset received flag
    }

    if (TI == 1)                        // if data sent flag is set
    {
        if (serial_out_count == 0)
        {
            if (rs232_out_buffer[0] == SOH)
            {
                SBUF = rs232_out_buffer[serial_out_count++];
            }
        }
        else
        {
            SBUF = rs232_out_buffer[serial_out_count++];
            if (rs232_out_buffer[serial_out_count-1] == EOT)
            {
                rs232_out_buffer[0] = 0;
                serial_out_count = 0;
            }
        }

        TI = 0;                         // reset sent flag
    }
}
```

```
/*************************************************************************/
/*      Name:           send_rs232_data                                  */
/*      Description:    send data string to rs232 port under interrupts  */
/*      Input:          char data_string - data string to be sent        */
/*                      unsigned char data_length - no. of bytes of data */
/*      Return Value:   -                                                */
/*************************************************************************/

void send_rs232_data (data_string, data_length)
unsigned char data_string[30];
unsigned char data_length;
{
    unsigned int i;
    unsigned char a;
    unsigned char shift;

    // ensure last data has already been sent first
    while ((serial_out_count != 0) && (i < 0xffff))
        i++;                            // increment timeout counter

    if (serial_out_count != 0)          // waited too long, return
       return;

    rs232_out_buffer[0] = SOH;          // 1st byte to send - start of header

    for (a = 0, shift = 1; a < data_length; a++)
    {
        if (data_string[a] == EOT ||    // substitution byte if EOT found
            data_string[a] == SUB ||    // substitution byte if SUB found
            data_string[a] == 0)        // substitution byte if NUL found
        {
            rs232_out_buffer[a+shift] = SUB;
            rs232_out_buffer[a+shift+1] = data_string[a] ^ 0x20;
            shift++;
        }
        else
            rs232_out_buffer[a+shift] = data_string[a];
    }

    rs232_out_buffer[a+shift]   = EOT;  // send end of transmission byte
    rs232_out_buffer[a+shift+1] = 0;    // terminate string

    TI = 1;                             // initiate interrupt sending
}


/*************************************************************************/
/* End Of Module                                                         */
/*************************************************************************/
```

8.3.13 time.h

```
#ifdef _TIME_

      void delay_10ms (unsigned char);          // delay time in 10ms increments
      void delay_us   (unsigned char);            // delay time in us increments

    //extern void decode_rs232_data (void);     // decode received data
    //extern data unsigned char buffer_full;    // ready to decode data flag

#else

    extern void delay_10ms (unsigned char);     // delay time in 10ms increments
      extern void delay_us   (unsigned char);     // delay time in us increments

#endif
```

8.3.14 time.c

```
#define _TIME_

#include <reg652.h>    // special function registers for the 80C652

#include "time.h"      // time routines header file


/*************************************************************************/
/*      Name:           delay_10ms                                       */
/*      Description:    delays for number of 10ms passed, upto 255 maximum */
/*      Calls:          _                                                */
/*      Input:          number of 10ms to delay by - delay_10ms_value    */
/*      Return Value:   -                                                */
/*************************************************************************/

void delay_10ms (unsigned char delay_10ms_value)
{
    while (delay_10ms_value)
    {
        TH0 = 0xd8;
        TL0 = 0xf0;
        TR0 = 1;

        while (!TF0)
        {
            //if (buffer_full == 1)                // check for rs232 data
            //    decode_rs232_data ();            // if so, decode it
       }

        TR0 = 0;
        TF0 = 0;

        delay_10ms_value--;
    }
}


/*************************************************************************/
/*      Name:           delay_us                                         */
/*      Description:    delays for number of us passed (upto 255 maximum)  */
/*      Calls:          _                                                */
/*      Input:          number of us to delay by - delay_us_value        */
/*      Return Value:   -                                                */
/*************************************************************************/

void delay_us (unsigned char delay_us_value)
{
    TH0 = 0xff;
    TL0 = (0xff - delay_us_value);
    TR0 = 1;

    while (!TF0)
       ;

    TR0 = 0;
    TF0 = 0;
}


/*************************************************************************/
/* End Of Module                                                         */
/*************************************************************************/
```

8.3.15 egg1.c

```
/*************************************************************************/
/*     EGG1.C:  ADAS development program using the C-51 COMPILER         */
/*************************************************************************/
/*                                                                       */
/*      Project:  ADAS Laser Egg Cleaning Machine                        */
/*      Author:   Richard M. Farrar     Creation Date:  19/07/96         */
/*      Filename: egg1.c                Language:       C                */
/*                                                                       */
/*      Compiler: Keil C-51             Assembler:                       */
/*      Version:  5.02c                 Version:                         */
/*                                                                       */
/*************************************************************************/
/*      Modification History                                             */
/*************************************************************************/
/*                                                                       */
/*      Version:                   Date:                                 */
/*                                                                       */
/*      Modification:                                                    */
/*                                                                       */
/*************************************************************************/

/*************************************************************************/
/*      Header Files To Be Included:-                                    */
/*************************************************************************/

#define _MAIN_                          // define module for includes

#include <reg652.h>                     // sfrs for 80C652
#include <ctype.h>                      // character functions
#include <string.h>                     // string functions
#include <absacc.h>                     // macro definitions

#include "lcd_text.h"                   // lcd print functions
#include "time.h"                       // time delay functions
#include "eeprom.h"                     // i2c eeprom functions
#include "i2c.h"                        // i2c functions
#include "galvos.h"                     // galvo control functions
#include "rs232.h"                      // serial communications functions

//#define DEV                           // conditional compilation for dev. rig


/*************************************************************************/
/*      Definitions:-                                                    */
/*************************************************************************/

#ifndef PORT_A
    #define PORT_A          XBYTE[0x0000]   // port a address of 8255
#endif

#ifndef PORT_B
    #define PORT_B          XBYTE[0x0001]   // port b address of 8255
#endif

#ifndef PORT_C
    #define PORT_C          XBYTE[0x0002]   // port c address of 8255
#endif

#ifndef PORT_CONTROL
    #define PORT_CONTROL    XBYTE[0x0003]   // control address of 8255
#endif

#ifndef PIO2
   #define PIO2             XBYTE[0x0004]   // base address of PIO 2
#endif

#define GREEN_LED           XBYTE[0x00f8]   // green led address
#define RED_LED             XBYTE[0x00f0]   // red led address

#define BASE_10             10              // base radix for itoa conversion
```

```
#define EGGS_PER_ROW        11              // number of eggs per row
#define ROWS_PER_TRAY       12              // 12 rows per tray

#define ROW_LENGTH          1200            // row length in pulses

#define OFF                 0x00            // define off state to be zero
#define ON                  0x01            // switches coloured led on
#define RETURN_ON           0x02            // return stroke on
#define RETURN_OFF          0x03            // retrun stroke off

#define OK                  0x00            // check for status
#define FAIL                0x01            // check for status

#define COMBINED            0x01            // two laser sources combined
#define SEPARATED           0X00            // two laser sources separated

#define STATIC_PROFILE      0x00            // no scan profile
#define LINEAR_PROFILE      0x01            // linear scan profile
#define COMPENSATED_PROFILE 0x02            // compensated profile

#define MANUAL_MODE         0x00            // manual mode
#define AUTOMATIC_MODE      0x01            // automatic mode

#define EGG_CLEANED         0xfe            // galvo has been triggered
#define EGG_WAITING         0xff            // waiting to be triggered

#define ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0    0xc4 // i2c control

#define SOH                 0x01        // start of header - (transmission)
#define ETX                 0x03        // end of text
#define EOT                 0x04        // end of transmission
#define SUB                 0x1b        // substitute - transparancey byte

#define LEFT_SCANNER        0x40        // i2c address of scanner dtoa 1
#define RIGHT_SCANNER       0x42        // i2c address of scanner dtoa 2
#define TOP_SCANNER         0x44        // i2c address of scanner dtoa 3
#define SCANNER_AMPLITUDE   0x00        // sub address of DAC0
#define SCANNER_ALARM       0x01        // sub address of DAC1
#define POS_THRESHOLD       0x02        // sub address of DAC2
#define NEG_THRESHOLD       0x03        // sub address of DAC3

#define DIGITAL_POT1        0x50        // i2c address of digital pot 1
#define LASER_HIGH_POWER    0x00        // laser high power address
#define LASER_LOW_POWER1    0x04        // laser low power 1 address
#define LASER_CLOCK_FREQ    0x08        // laser power clock frequency addr.

#define DIGITAL_POT2        0x52        // i2c address of digital pot 2
#define LASER_LOW_POWER2    0x00        // laser low power 2 address
#define LASER_TICKLE_POWER  0x04        // laser tickle power address
#define LASER_TEST_POWER    0x08        // laser test power address

#define EEPROM              0xa0        // i2c base address of eeprom
#define I2C_PORT1           0x4e        // i2c base address of 8-bit I/O_1
#define I2C_PORT2           0x4c        // i2c base address of 8-bit I/O_2

#define set_test_fire1()    PORT_C |= 0x10  // closed - Relay 1
#define clear_test_fire1()  PORT_C &= 0xef  // opened - Relay 1
#define set_test_fire2()    PORT_C |= 0x08  // closed - Relay 2
#define clear_test_fire2()  PORT_C &= 0xf7  // opened - Relay 2
#define clear_laser_ready() PORT_C &= 0xfe  // opened - Relay 5
#define set_laser_ready()   PORT_C |= 0x01  // closed - Relay 5


#define STAALKAT_CHECK      0x01    // staalkat machine ready signal
#define STAALKAT_READY      0x00    // staalkat machine ready
#define STAALKAT_NOTREADY   0x01    // staalkat machine not ready

#define FW_BW               0x02    // forward backward signal
#define FORWARDS            0x02    // forwards signal
#define BACKWARDS           0x00    // backwards signal

#define L_R                 0x04    // left right signal
#define LEFT                0x00    // left signal
#define RIGHT               0x04    // right signal
```

```
#define U_D                 0x80    // up down signal
#define UP                  0x00    // down position of carraige
#define DOWN                0x80    // up position of carraige

#define SENSE_EGG           0x08    // egg sense signal
#define EGG_PRESENT         0x00    // egg sensed
#define EGG_POSITION        0x10    // egg position to check
#define E_STOP              0x20    // staalkat e-stop signal
#define POSITION_OK         0x00    // position sensed

sbit    ENCODER1            =   0xb2;   // INT0
sbit    ENCODER2            =   0xb4;   // T0
sbit    A0_PIO2             =   0x91;   // address 0 pin for PIO 2
sbit    A1_PIO2             =   0x92;   // address 1 pin for PIO 2
sbit    LEFT_SCANNER_ALARM  =   0x93;   // low alarm input for left scanner
sbit    RIGHT_SCANNER_ALARM =   0x94;   // low alarm input for right scanner
sbit    TOP_SCANNER_ALARM   =   0x95;   // low alarm input for top scanner


/************************************************************************/
/*      Variables:-                                                     */
/************************************************************************/

xdata unsigned int  position        = 0;        // counter, encoder offset


// rs-232 variables

data unsigned char  in_buff_len     = 0;
data unsigned char  buffer_full     = 0;        // buffer data flag
xdata unsigned char rs232_in_buffer[0x1f];      // rs232 input buffer


// egg variables

xdata unsigned long eggs_cleaned;               // total number of eggs cleaned
xdata unsigned long rows_cleaned;               // total number of rows cleaned
xdata unsigned char egg_position[EGGS_PER_ROW+1]; // log of eggs processed in row


// system control variables

xdata unsigned char system_mode;               // manual / automatic mode
xdata unsigned char mirror_state = 0xff;
xdata unsigned char error_status;
xdata unsigned char galvo_count = 0;           // counter, galvo position
xdata unsigned char marking_egg = OFF;         // flag, egg being fired upon
xdata unsigned char system_ok = OK;            // system safe to fire flag


// system configuration settings

struct scanner_data     {   unsigned char amplitude;         // amplitude setting
of scanner, max 63
                            unsigned char low_alarm;         // low alarm point of
scanner
                            unsigned char upper_threshold;   // lower laser power
1, upper threshold, max 63
                            unsigned char lower_threshold;   // lower laser power
2, lower threshold, max 63
                        };

struct laser_data       {   unsigned char high;              // laser power high,
max 63
                            unsigned char low1;              // laser power low 1,
max 63
                            unsigned char low2;              // laser power low 2,
max 63
                            unsigned char test;              // laser power test,
max 63
                            unsigned char tickle;            // laser power tickle,
max 63
                            unsigned  char  frequency;             //  laser  power
frequency, max 63
                        };
```

```
struct eeprom_data      {   unsigned char pulse_delay;       // delay between egg
and mark position
                            unsigned char start_pos;          // position to start
firing on return stroke
                            unsigned char stop_pos;           // position to stop
firing on return stroke
                            unsigned char egg_width;        // egg width
                            unsigned char egg_spacing;         // spacing between
eggs

                            unsigned char galvo_max_amp;          // galvo maximum
amplitude
                            unsigned char galvo_offset;         // offset for both
galvos
                            unsigned char galvo_scan_profile;     // galvo scan
profile

                            struct scanner_data top_scanner;
                            struct scanner_data left_scanner;
                            struct scanner_data right_scanner;

                            struct laser_data laser_power;
                        };

xdata struct eeprom_data system_settings;                        // eeprom system
settings structure
xdata struct eeprom_data eeprom_settings;

code struct eeprom_data system_defaults =   {   140,        // pulse delay
                                                70,           // position to start
firing on return stroke
                                                100,          // position to stop
firing on return stroke, 950, hence *10!
                                                45,         // egg width in pulses
                                                75,             // egg spacing in
pulses
                                                0xff,              // galvo max.
amplitude
                                                80,         // galvo offset
                                                LINEAR_PROFILE,   //   galvo   scan
profile
                                                {
                                                    16,           // top scanner
amplitude, 0x98
                                                    2,         // top scanner low
alarm
                                                    16,       // top scanner upper
threshold, 0x90
                                                    15        // top scanner lower
threshold
                                                },
                                                {
                                                    40,          // left scanner
amplitude, 0xe0
                                                    2,         // left scanner low
alarm
                                                    24,      // left scanner upper
threshold, 0xd8
                                                    15       // left scanner lower
threshold
                                                },
                                                {
                                                    40,          // right scanner
amplitude, 0xe0
                                                    2,         // right scaner low
alarm
                                                    24,     // right scanner upper
threshold, 0xd8
                                                    15      // right scanner lower
threshold
                                                },
                                                {             // 63 is max. value
allowed for these parameters
                                                    30,       // high laser power,
50?
                                                    30,       // low1 laser power,
5?
```

```
                                                30,       // low2 laser power,
5?
                                                5,        // test laser power,
5?
                                                4,       // tickle laser power,
4?
                                                47         // laser frequency,
47?
                                            }
                                        };

// fixed code data tables

code unsigned char scan_profile[80] =   {  0,   1,  2,  3,  4,  5,  6,  7,  8,  9,
                                           10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
                                           20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
                                           30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
                                           40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
                                           50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
                                           60, 61, 62, 63, 64, 65, 66, 67, 68, 69,
                                           70, 71, 72, 73, 74, 75, 76, 77, 78, 79
                                        };


code char *lcd_text[12] =   {  {"                    "},   // 0: blank line
                               {" Ver 1.0 - 20/05/00 "},   // 1: version screen
                               {"--------------------"},   // 2: line
                               {"Status: INITIALISING"},   // 3: init screen
                               {"Status: ERROR       "},   // 4: error status
                               {"Status: ARMED       "},   // 5: armed status
                               {"Status: DISARMED    "},   // 6: disarmed status
                               {"Status: FIRING      "},   // 7: firing status
                               {"Couldn't arm lasers "},   // 8: can't arm
                               {"Eggs  : 0           "},   // 9: number of eggs
                               {"?|?|?|?|?|?|?|?|?|? "},   // 10: separters
                               {"Rows  : 0           "}    // 11: number of rows
                            };


/*************************************************************************/
/*      Function Prototypes:-                                            */
/*************************************************************************/

unsigned char initialise_system (void);     // initialise system
unsigned char check_faults (void);          // check system faults

void combine_lasers (void);                 // move mirror positions in
void separate_lasers (void);                // move mirror positions out
void arm_system (void);                     // set laser enable signal
void disarm_system (void);                  // clear laser enable signal
unsigned char fire_lasers (void);           // set fire laser signal
void stop_firing_lasers (void);             // clear laser fire signal

void encoder_int (void);                    // encoder interrupt
void decode_rs232_data (void);              // decode received rs232 data
char* itoa (unsigned long, unsigned char);  // converts number to string

void set_galvos (unsigned char);            // set galvo levels


/*************************************************************************/
/*************************     MAIN PROGRAM      *************************/
/*************************************************************************/
/*      Calls: initialise                                                */
/*************************************************************************/

void main(void)
{
    unsigned char j, k          = 0;    // local loop variables
    xdata unsigned long l       = 0;
    unsigned char egg_count     = 0;    // counter
    unsigned char egg_pos       = 0;    // counter for egg positions sensed
    unsigned char egg_pos_flag  = 0xff; // flag, egg position state
    unsigned char egg_sensed    = 0xff; // flag, egg found
    unsigned char handing       = 0xff; // flag, handing of current row
    unsigned char height        = 0xff; // flag, carriage height
    unsigned char estop         = 0xff; // flag, e-stop status
```

```
    unsigned char staalkat      = 0xff; // flag, staalkat machine OK
    unsigned char cleaned_eggs  = 0;    // counter, number of eggs cleaned
    unsigned char temp_str[10];         // temporary string

    error_status = initialise_system ();

    // error if anything other than eeprom fails
    if (error_status > 1)               // failed initialisation
    {
       clear_lcd ();

       while (1)
       {
           print_lcd (lcd_text[4], LINE_1, 0); // error status
           print_lcd (lcd_text[2], LINE_2, 0); // separating line
           print_lcd ("Initialise failed...", LINE_3, 0);
           print_lcd ("Can't run machine!  ", LINE_4, 0);

           delay_10ms (100);                   // flash display
           clear_lcd ();
           delay_10ms (50);
       }
    }
    arm_system ();

    // display egg count
    print_lcd (lcd_text[9], LINE_2, 0);
    strcpy (temp_str, itoa (eggs_cleaned, BASE_10));
    print_lcd (temp_str, LINE_2, 8);

    // display row count
    print_lcd (lcd_text[11], LINE_3, 0);
    strcpy (temp_str, itoa (rows_cleaned, BASE_10));
    print_lcd (temp_str, LINE_3, 8);

    // clear the queue
    for (j = 0; j < (EGGS_PER_ROW+1); j++)
        egg_position[j] = EGG_WAITING;


/************** main program loop starts here       ***************/

    while(1)
    {
        if (buffer_full == 1)                   // check for rs232 data
            decode_rs232_data ();               // if so, decode it

        if (system_mode == AUTOMATIC_MODE)
        {
           // egg position on carraige sensed
           if ((PORT_B & EGG_POSITION) == POSITION_OK && egg_pos_flag != ON)
           {
               egg_pos++;
               egg_pos_flag = ON;
               print_lcd ("~", LINE_4, 10);    // show position flag on LCD
           }


           // egg position on carraige not sensed
           if ((PORT_B & EGG_POSITION) != POSITION_OK && egg_pos_flag != OFF)
           {
               egg_pos_flag = OFF;
               print_lcd (".", LINE_4, 10);    // show position flag on LCD
           }


            // new egg sensed - leading edge
            if (((PORT_B & SENSE_EGG) == EGG_PRESENT) &&
               egg_sensed != ON &&
               height == UP &&
               mirror_state == SEPARATED)
            {
               egg_sensed = ON;                 // new egg found
               print_lcd ("~", LINE_4, 12);    // show egg flag on LCD

               egg_position[egg_count++] = system_settings.pulse_delay;
            }
```

```
            // end of egg sensed - trailing edge
            if (((PORT_B & SENSE_EGG) != EGG_PRESENT) && egg_sensed != OFF)
            {
               egg_sensed = OFF;
               print_lcd (".", LINE_4, 12);    // show egg flag on LCD
            }


            // fire on new egg if in position
            if ((egg_position[cleaned_eggs] == 0) &&
                ((PORT_B & FW_BW) == FORWARDS) &&
                marking_egg == OFF &&
                system_ok == OK)
            {
               galvo_count = 0;                  // double check galvo in correct
position
               set_galvos (galvo_count);

               marking_egg = ON;
               if (fire_lasers () != OK)
                   marking_egg = OFF;
            }


            // if whole egg marked
            if (galvo_count > system_settings.egg_spacing)
            {
                stop_firing_lasers ();
                galvo_count = 0;                    // reset galvo position
                set_galvos (galvo_count);           // flyback to start

                eggs_cleaned++;     // save to eeprom here or at end of row??
                marking_egg = OFF;
                egg_position[cleaned_eggs++] = EGG_CLEANED;

                strcpy (temp_str, itoa (eggs_cleaned, BASE_10));
                print_lcd (temp_str, LINE_2, 8);    // display egg count

                delay_10ms (1);
                send_rs232_data ("w", 1);           // tell host PC egg cleaned
            }


           // separate lasers for forward stroke
           if ((PORT_B & FW_BW) == FORWARDS && mirror_state != SEPARATED)
           {
                separate_lasers ();

                for (j = 0; j < EGGS_PER_ROW; j++)  // clear queue
                    egg_position[j] = EGG_WAITING;

                  //?????????????????????????????? test
                strcpy (temp_str, itoa (position, BASE_10));
                print_lcd (temp_str, LINE_3, 14);   // display position????
                  //?????????????????????????????? end test

                if (system_ok == OK)
                    cleaned_eggs    = 0;
                egg_count       = 0;
                egg_pos         = 0;
                position        = 0;            // reset encoder position

               print_lcd (" ", LINE_4, 4);     // show direction flag on LCD
            }


            // combine lasers for return stroke
            if ((PORT_B & FW_BW) == BACKWARDS && mirror_state != COMBINED)
            {
               combine_lasers ();

               //?????????????????????????????? test
               strcpy (temp_str, itoa (position, BASE_10));
               print_lcd (temp_str, LINE_3, 14);   // display position????
               //?????????????????????????????? end test
```

```
    egg_pos         = 0;            // reset egg position counter
    position        = 0;            // reset encoder position

    print_lcd ("~", LINE_4, 4);     // show direction flag on LCD

    if (system_ok == OK)
        rows_cleaned++;
    strcpy (temp_str, itoa (rows_cleaned, BASE_10));
    print_lcd (temp_str, LINE_3, 8);// display row count

    // 80ms delay with 2 statements below
    i2c_eeprom_count_write (EEPROM, 0x34, eggs_cleaned);
    i2c_eeprom_count_write (EEPROM, 0x38, rows_cleaned);

    temp_str[0] = 'x';

    l = eggs_cleaned;
    temp_str[1] = l % 65536UL;
    l /= 256UL;                      // knock off lo byte
    temp_str[2] = l % 256UL;
    temp_str[3] = l / 256UL;

    l = rows_cleaned;
    temp_str[4] = l % 65536UL;
    l /= 256UL;                      // knock off lo byte
    temp_str[5] = l % 256UL;
    temp_str[6] = l / 256UL;
    temp_str[7] = 0;
    send_rs232_data (temp_str, 7);
 }


// left handed row detected
if ((PORT_B & L_R) == LEFT && handing != LEFT)
{
    handing = LEFT;
    print_lcd ("L", LINE_4, 0);     // show handing flag on LCD
}


// right handed row detected
 if ((PORT_B & L_R) == RIGHT && handing != RIGHT)
 {
    handing = RIGHT;
    print_lcd ("R", LINE_4, 0);     // show handing flag on LCD
 }


// carraige up detected
if ((PORT_B & U_D) == UP && height != UP)
{
    height = UP;
    print_lcd ("´", LINE_4, 2);     // show height flag on LCD
}


// carraige down detected
if ((PORT_B & U_D) == DOWN && height != DOWN)
{
    height = DOWN;
    stop_firing_lasers ();          // double check for safety

    print_lcd ("^", LINE_4, 2);     // show height flag on LCD
}


// check for staalkat e-stop active
if ((PORT_B & E_STOP) == E_STOP && estop != ON)
{
    estop = ON;
    print_lcd ("E", LINE_4, 6);     // show e-stop flag on LCD
}
```

```
            // check for staalkat e-stop clear
            if ((PORT_B & E_STOP) != E_STOP && estop != OFF)
            {
                estop = OFF;
                print_lcd (".", LINE_4, 6);     // show e-stop flag on LCD
            }

            // check for staalkat ready
            if  ((PORT_B  &  STAALKAT_CHECK)  ==  STAALKAT_READY  &&  (staalkat  !=
STAALKAT_READY))
            {
                staalkat    = STAALKAT_READY;
                system_ok   = OK;

                egg_pos     = 0;                // reset egg position counter
                position    = 0;                // reset encoder position

                if (mirror_state == SEPARATED)
                {
                    for (j = 0; j < EGGS_PER_ROW; j++)  // clear queue
                         egg_position[j] = EGG_WAITING;

                    eggs_cleaned = eggs_cleaned - cleaned_eggs;
                    cleaned_eggs    = 0;
                    egg_count       = 0;

                    strcpy (temp_str, itoa (eggs_cleaned, BASE_10));
                    print_lcd (temp_str, LINE_2, 8); // display egg count

                    temp_str[0] = 'x';

                    l = eggs_cleaned;
                    temp_str[1] = l % 65536UL;
                    l /= 256UL;                         // knock off lo byte
                    temp_str[2] = l % 256UL;
                    temp_str[3] = l / 256UL;

                    l = rows_cleaned;
                    temp_str[4] = l % 65536UL;
                    l /= 256UL;                         // knock off lo byte
                    temp_str[5] = l % 256UL;
                    temp_str[6] = l / 256UL;
                    temp_str[7] = 0;
                    send_rs232_data (temp_str, 7);
                }
            }

            // check for staalkat not ready
            if  ((PORT_B  &  STAALKAT_CHECK)  ==  STAALKAT_NOTREADY  &&  (staalkat  !=
STAALKAT_NOTREADY))
            {
                staalkat = STAALKAT_NOTREADY;
            }


            // fire enable on return stroke
            // do we want to fire on individual eggs here ????

            // what happens here in an error condition????

            if (mirror_state        == COMBINED &&
                height              == UP  &&
                marking_egg         == OFF &&
                position            >  system_settings.start_pos &&
                cleaned_eggs        >  0 &&
                system_ok           == OK)
            {
                marking_egg = RETURN_ON;
                if (fire_lasers () != OK)
                    marking_egg = OFF;
            }
```

```
            // fire disable on return stroke
            if (mirror_state == COMBINED &&
                height == UP &&
                marking_egg == RETURN_ON &&
                (egg_pos > 9 ||
                position > (system_settings.stop_pos * 10)))
            {
                stop_firing_lasers ();
                marking_egg = RETURN_OFF;
            }

            // check faults
            j = check_faults();

            if (j == 0)
            {
                set_laser_ready();
                if (error_status != 0)
                {
                    error_status = 0;
                    print_lcd (lcd_text[5],  LINE_1, 0);    // system armed
                    print_lcd (lcd_text[9],  LINE_2, 0);    // egg counter
                    print_lcd (lcd_text[11], LINE_3, 0);    // row counter

                    strcpy (temp_str, itoa (rows_cleaned, BASE_10));
                    print_lcd (temp_str, LINE_3, 8);   // display row count

                    strcpy (temp_str, itoa (eggs_cleaned, BASE_10));
                    print_lcd (temp_str, LINE_2, 8);   // display egg count
                }
            }
            else
            {
                stop_firing_lasers ();
                clear_laser_ready();
                system_ok = FAIL;
                marking_egg = OFF;
                error_status = j;
            }
        }
    }
}

/*****************************************************************************/
/*      Name:          check_faults                                          */
/*      Description:   checks faults for system                              */
/*      Input:         -                                                     */
/*      Return Value:  unsigned char - error code                            */
/*****************************************************************************/

unsigned char check_faults (void)
{
   unsigned char i;
   unsigned char error = 0;

   // check all scanners are working

   i = (P1 & 0x38);

   switch (i)
   {
       case 0x08:  print_lcd (">>LHS scanner fault ", LINE_2, 0);
                   break;
       case 0x10:  print_lcd (">>RHS scanner fault ", LINE_2, 0);
                   break;
       case 0x20:  print_lcd (">>Top scanner fault ", LINE_2, 0);
                   break;
       case 0x18:  print_lcd (">>L/R scanner fault ", LINE_2, 0);
                   break;
       case 0x28:  print_lcd (">>L/T scanner fault ", LINE_2, 0);
                   break;
       case 0x30:  print_lcd (">>R/T scanner fault ", LINE_2, 0);
                   break;
       case 0x38:  print_lcd (">>All scanner fault ", LINE_2, 0);
                   break;
       default:    break;
   }
```

```
if (i != 0)
{
    error = 1;
    print_lcd (lcd_text[2], LINE_3, 0); // line
    print_lcd (lcd_text[4], LINE_1, 0); // status: error
}


// check laser DC power supplies

#ifndef DEV
    i = ((i2c_io_read (I2C_PORT2)) & 0x3f);
#else
    i = 0;
#endif

if (i > 0)
{
    if (error != 0)
        delay_10ms (100);
    else
    {
        print_lcd (lcd_text[4], LINE_1, 0);
        print_lcd (lcd_text[2], LINE_3, 0);
    }

    print_lcd (">>PSU Fault # ......", LINE_2, 0);
    error += 2;

    if ((i & 0x01) == 0x01)
        print_lcd ("1", LINE_2, 14);

    if ((i & 0x02) == 0x02)
        print_lcd ("2", LINE_2, 15);

    if ((i & 0x04) == 0x04)
        print_lcd ("3", LINE_2, 16);

    if ((i & 0x08) == 0x08)
        print_lcd ("4", LINE_2, 17);

    if ((i & 0x10) == 0x10)
        print_lcd ("5", LINE_2, 18);

    if ((i & 0x20) == 0x20)
        print_lcd ("6", LINE_2, 19);
}


// check laser ok signals

#ifndef DEV
    i = ((i2c_io_read (I2C_PORT1))  & 0x77);
#else
    i = 0;
#endif

if (i > 0)
{
    if (error != 0)
        delay_10ms (100);
    else
        print_lcd (lcd_text[4], LINE_1, 0);

    print_lcd (">>Laser1:            ", LINE_2, 0);
    print_lcd (">>Laser2:            ", LINE_3, 0);
    error += 4;

    if ((i & 0x01) == 0x01)     // 2:laser ready
        print_lcd ("Laser Rdy", LINE_3, 10);

    if ((i & 0x02) == 0x02)     // 2:key switch ok
        print_lcd ("Key Sw.  ", LINE_3, 10);

    if ((i & 0x04) == 0x04)     // 2:water ok
        print_lcd ("Low Flow ", LINE_3, 10);
```

```
        if ((i & 0x10) == 0x10)     // 1:laser ready
            print_lcd ("Laser Rdy", LINE_2, 10);

        if ((i & 0x20) == 0x20)     // 1:key switch ok
            print_lcd ("Key Sw.  ", LINE_2, 10);

        if ((i & 0x40) == 0x40)     // 1:water ok
            print_lcd ("Low Flow ", LINE_2, 10);

        if (error > 4)
            delay_10ms (100);
    }


    // check mirror positions
    A1_PIO2 = 1;
    A0_PIO2 = 0;                    // set to read PIO2 Port C
    i = (PIO2 & 0x0f);

    if (mirror_state == SEPARATED && position > 100 &&
        (PORT_B & U_D) == UP)       // check mirrors for forward stroke
    {
        if (i != 0x0a)
        {
            error += 8;

            if ((i & 0x05) == 0x01)
               print_lcd (">>Left Mirror fault ", LINE_2, 0);

            if ((i & 0x05) == 0x04)
               print_lcd (">>Right Mirror fault", LINE_2, 0);

            if ((i & 0x05) == 0x05)
               print_lcd (">>Both Mirror fault ", LINE_2, 0);
        }
    }

    if (marking_egg == RETURN_ON)    // check mirrors for return stroke
    {
        if (i != 0x05)
        {
            error += 8;

            if ((i & 0x0a) == 0x02)
               print_lcd (">>Left Mirror fault ", LINE_2, 0);

            if ((i & 0x0a) == 0x08)
               print_lcd (">>Right Mirror fault", LINE_2, 0);

            if ((i & 0x0a) == 0x0a)
               print_lcd (">>Both Mirror fault ", LINE_2, 0);
        }
    }

    if (error >= 8)
    {
        print_lcd (lcd_text[2], LINE_3, 0); // line
        print_lcd (lcd_text[4], LINE_1, 0); // status: error
    }

    if (error > 8)
        delay_10ms (100);

    return error;
}


/*************************************************************************/
/*      Name:          initialise_system                                 */
/*      Description:   Initialises system components and variables       */
/*      Input:         -                                                 */
/*      Return Value:  -                                                 */
/*************************************************************************/
```

```
unsigned char initialise_system (void)
{
    xdata unsigned char error = 0;          // error status,0 = OK
    xdata unsigned char i;                  // local loop variable

    xdata char temp_str[6];                 // temporary string


    // configure port settings
    PORT_CONTROL        = 0x82;             // configure 8255 ports
    A0_PIO2             = 1;                // A0 pin for second 8255
    A1_PIO2             = 1;                // A1 pin for second 8255
    PIO2                = 0x81;             // configure 2nd 8255 PIO


    // configure processor registers
    SCON                = 0x50;             // initialize UART, 8,N,1
    PS                  = 0;                // uart interrupt priority
    TMOD                = 0x21;
    TCON                = 1;                // external int. edge triggered
    TH1                 = 0xfd;             // set timer period
    TL1                 = 0;
    TR1                 = 1;                // start timer 1
    IE                  = 0xb1;             // enable serial int. and ext. 1 int.
    RI                  = 0;                // clear serial RX int. flag
    TI                  = 0;                // clear serial TX int. flag
    IE0                 = 0;                // clear ext. int 0 flag
    SI                  = 0;
    T1                  = 0;                // disable scanner trigger


    // configure i2c set-up
    SDA                 = 1;                // enable for SDA line
    SCL                 = 1;                // enable for SCL line
    PS1                 = 0;                // i2c interrupt priority
    S1ADR               = 0x31;             // address + general call set
    S1CON = ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0;// configure I2C port


    // enable resonant scanner alarm inputs
    LEFT_SCANNER_ALARM  = 1;                // enable input line for alarm
    RIGHT_SCANNER_ALARM = 1;                // enable input line for alarm
    TOP_SCANNER_ALARM   = 1;                // enable input line for alarm

    delay_10ms (20);                        // wait for PSU to settle
    initialise_lcd ();                      // initialise lcd


    // set-up lcd characters
    send_lcd_control_byte (72);             // lcd up character design, 0x01
    send_lcd_data_byte (0);
    send_lcd_data_byte (4);
    send_lcd_data_byte (14);
    send_lcd_data_byte (21);
    send_lcd_data_byte (4);
    send_lcd_data_byte (4);
    send_lcd_data_byte (0);
    send_lcd_data_byte (0);

    send_lcd_data_byte (0);                 // lcd down character design, 0x02
    send_lcd_data_byte (4);
    send_lcd_data_byte (4);
    send_lcd_data_byte (21);
    send_lcd_data_byte (14);
    send_lcd_data_byte (4);
    send_lcd_data_byte (0);
    send_lcd_data_byte (0);

    send_lcd_data_byte (14);                // lcd filled blob char design, 0x03
    send_lcd_data_byte (31);
    send_lcd_data_byte (31);
    send_lcd_data_byte (31);
    send_lcd_data_byte (31);
    send_lcd_data_byte (31);
    send_lcd_data_byte (14);
    send_lcd_data_byte (0);
```

```
    send_lcd_data_byte (0);                 // lcd tick mark, 0x04
    send_lcd_data_byte (1);
    send_lcd_data_byte (1);
    send_lcd_data_byte (2);
    send_lcd_data_byte (18);
    send_lcd_data_byte (12);
    send_lcd_data_byte (4);
    send_lcd_data_byte (0);


    // show lcd boot-up screen
    print_lcd (lcd_text[3], LINE_1, 0);     // initialising status
    print_lcd (lcd_text[2], LINE_2, 0);     // separating line
    print_lcd (lcd_text[1], LINE_3, 0);     // version number, date
    print_lcd (lcd_text[2], LINE_4, 0);     // separating line

    delay_10ms (100);                       // pause on init screen


    // reset variables
    system_mode         = AUTOMATIC_MODE;   // set system to automatic mode
    system_settings     = system_defaults;  // get default settings


    i = 0;                                      // error count for checking eeprom
validity

   // get system settings from eeprom
   if (i2c_eeprom_check (EEPROM, 0x00, 1) == OK)
   {
       eeprom_settings.pulse_delay = i2c_eeprom_read (EEPROM, 0x00);
       system_settings.pulse_delay = eeprom_settings.pulse_delay;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x02, 1) == OK)
   {
       eeprom_settings.start_pos = i2c_eeprom_read (EEPROM, 0x02);
       system_settings.start_pos = eeprom_settings.start_pos;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x04, 1) == OK)
   {
       eeprom_settings.stop_pos = i2c_eeprom_read (EEPROM, 0x04);
       system_settings.stop_pos = eeprom_settings.stop_pos;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x06, 1) == OK)
   {
       eeprom_settings.egg_width = i2c_eeprom_read (EEPROM, 0x06);
       system_settings.egg_width = eeprom_settings.egg_width;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x08, 1) == OK)
   {
       eeprom_settings.egg_spacing = i2c_eeprom_read (EEPROM, 0x08);
       system_settings.egg_spacing = eeprom_settings.egg_spacing;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x0a, 1) == OK)
   {
       eeprom_settings.galvo_max_amp = i2c_eeprom_read (EEPROM, 0x0a);
       system_settings.galvo_max_amp = eeprom_settings.galvo_max_amp;
   }
   else
       i++;
```

```
   if (i2c_eeprom_check (EEPROM, 0x0c, 1) == OK)
   {
        eeprom_settings.galvo_offset = i2c_eeprom_read (EEPROM, 0x0c);
        system_settings.galvo_offset = eeprom_settings.galvo_offset;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x0e, 1) == OK)
   {
       eeprom_settings.galvo_scan_profile = i2c_eeprom_read (EEPROM, 0x0e);
       system_settings.galvo_scan_profile = eeprom_settings.galvo_scan_profile;
   }
   else
       i++;


   // get top resonant scanner settings from eeprom
   if (i2c_eeprom_check (EEPROM, 0x10, 1) == OK)
   {
       eeprom_settings.top_scanner.amplitude = i2c_eeprom_read (EEPROM, 0x10);
       system_settings.top_scanner.amplitude                                    =
eeprom_settings.top_scanner.amplitude;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x12, 1) == OK)
   {
    eeprom_settings.top_scanner.low_alarm = i2c_eeprom_read (EEPROM, 0x12);
    system_settings.top_scanner.low_alarm = eeprom_settings.top_scanner.low_alarm;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x14, 1) == OK)
   {
        eeprom_settings.top_scanner.upper_threshold   =   i2c_eeprom_read   (EEPROM,
0x14);
        system_settings.top_scanner.upper_threshold                             =
eeprom_settings.top_scanner.upper_threshold;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x16, 1) == OK)
   {
    eeprom_settings.top_scanner.lower_threshold = i2c_eeprom_read (EEPROM, 0x16);
    system_settings.top_scanner.lower_threshold                             =
eeprom_settings.top_scanner.lower_threshold;
   }
   else
       i++;


   // get left resonant scanner settings from eeprom
   if (i2c_eeprom_check (EEPROM, 0x18, 1) == OK)
   {
       eeprom_settings.left_scanner.amplitude = i2c_eeprom_read (EEPROM, 0x18);
       system_settings.left_scanner.amplitude                                   =
eeprom_settings.left_scanner.amplitude;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x1a, 1) == OK)
   {
    eeprom_settings.left_scanner.low_alarm = i2c_eeprom_read (EEPROM, 0x1a);
    system_settings.left_scanner.low_alarm                                  =
eeprom_settings.left_scanner.low_alarm;
   }
   else
       i++;
```

```
   if (i2c_eeprom_check (EEPROM, 0x1c, 1) == OK)
   {
    eeprom_settings.left_scanner.upper_threshold = i2c_eeprom_read (EEPROM, 0x1c);
    system_settings.left_scanner.upper_threshold                                    =
eeprom_settings.left_scanner.upper_threshold;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x1e, 1) == OK)
   {
    eeprom_settings.left_scanner.lower_threshold = i2c_eeprom_read (EEPROM, 0x1e);
    system_settings.left_scanner.lower_threshold                                    =
eeprom_settings.left_scanner.lower_threshold;
   }
   else
       i++;


   // get right resonant scanner settings from eeprom
   if (i2c_eeprom_check (EEPROM, 0x20, 1) == OK)
   {
       eeprom_settings.right_scanner.amplitude = i2c_eeprom_read (EEPROM, 0x20);
       system_settings.right_scanner.amplitude                                     =
eeprom_settings.right_scanner.amplitude;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x22, 1) == OK)
   {
    eeprom_settings.right_scanner.low_alarm = i2c_eeprom_read (EEPROM, 0x22);
    system_settings.right_scanner.low_alarm                                         =
eeprom_settings.right_scanner.low_alarm;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x24, 1) == OK)
   {
    eeprom_settings.right_scanner.upper_threshold   =   i2c_eeprom_read   (EEPROM,
0x24);
    system_settings.right_scanner.upper_threshold                                   =
eeprom_settings.right_scanner.upper_threshold;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x26, 1) == OK)
   {
    eeprom_settings.right_scanner.lower_threshold   =   i2c_eeprom_read   (EEPROM,
0x26);
    system_settings.right_scanner.lower_threshold                                   =
eeprom_settings.right_scanner.lower_threshold;
   }
   else
       i++;


   // get laser settings from eeprom
   if (i2c_eeprom_check (EEPROM, 0x28, 1) == OK)
   {
       eeprom_settings.laser_power.high = i2c_eeprom_read (EEPROM, 0x28);
       system_settings.laser_power.high = eeprom_settings.laser_power.high;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x2a, 1) == OK)
   {
    eeprom_settings.laser_power.low1 = i2c_eeprom_read (EEPROM, 0x2a);
    system_settings.laser_power.low1 = eeprom_settings.laser_power.low1;
   }
   else
       i++;
```

```
   if (i2c_eeprom_check (EEPROM, 0x2c, 1) == OK)
   {
    eeprom_settings.laser_power.low2 = i2c_eeprom_read (EEPROM, 0x2c);
    system_settings.laser_power.low2 = eeprom_settings.laser_power.low2;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x2e, 1) == OK)
   {
        eeprom_settings.laser_power.test = i2c_eeprom_read (EEPROM, 0x2e);
        system_settings.laser_power.test = eeprom_settings.laser_power.test;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x30, 1) == OK)
   {
    eeprom_settings.laser_power.tickle = i2c_eeprom_read (EEPROM, 0x30);
    system_settings.laser_power.tickle = eeprom_settings.laser_power.tickle;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x32, 1) == OK)
   {
    eeprom_settings.laser_power.frequency = i2c_eeprom_read (EEPROM, 0x32);
    system_settings.laser_power.frequency = eeprom_settings.laser_power.frequency;
   }
   else
       i++;

   if (i2c_eeprom_check (EEPROM, 0x34, 3) == OK)
       eggs_cleaned = i2c_eeprom_count_read (EEPROM, 0x34);      // get current
count
   else
   {
       i++;
       eggs_cleaned = 0;
   }

   if (i2c_eeprom_check (EEPROM, 0x38, 3) == OK)
       rows_cleaned = i2c_eeprom_count_read (EEPROM, 0x38);      // get current
count
   else
   {
       i++;
       rows_cleaned = 0;
   }

   // check and show error if eeprom not read correctly
   clear_lcd ();

   if (i != 0)                                          // error reading eeprom
   {
       print_lcd ("EEPROM read:   X-   ", LINE_1, 0);
       strcpy (temp_str, itoa (i, BASE_10));
       print_lcd (temp_str, LINE_1, 17);                // display error no.
       print_lcd (lcd_text[2], LINE_2, 0);              // print line on lcd
       print_lcd ("Some system settings", LINE_3, 0);
       print_lcd ("returned to default.", LINE_4, 0);

       delay_10ms (250);
       print_lcd (lcd_text[0], LINE_2, 0);              // clear other lines
       print_lcd (lcd_text[0], LINE_3, 0);
       print_lcd (lcd_text[0], LINE_4, 0);

       error = 1;                                       // error initialsing eeprom
   }
   else
       print_lcd ("EEPROM read:        ", LINE_1, 0);


   #ifndef DEV
```

```
    // set-up laser power settings
    i       =       i2c_e2pot               (DIGITAL_POT1,       LASER_CLOCK_FREQ,
system_settings.laser_power.frequency);
    i        +=        2*(i2c_e2pot        (DIGITAL_POT1,        LASER_HIGH_POWER,
system_settings.laser_power.high));
    i        +=        4*(i2c_e2pot        (DIGITAL_POT1,        LASER_LOW_POWER1,
system_settings.laser_power.low1));
    i        +=        8*(i2c_e2pot        (DIGITAL_POT2,        LASER_LOW_POWER2,
system_settings.laser_power.low2));
    i        +=       16*(i2c_e2pot        (DIGITAL_POT2,        LASER_TEST_POWER,
system_settings.laser_power.test));
    i       +=       32*(i2c_e2pot       (DIGITAL_POT2,       LASER_TICKLE_POWER,
system_settings.laser_power.tickle));

   #else
       i = 1;
   #endif

   if (i != 0)
   {
       print_lcd ("Laser init.:   X-   ", LINE_2, 0);
       strcpy (temp_str, itoa (i, BASE_10));
       print_lcd (temp_str, LINE_2, 17);       // display error no.
       error += 2;         // error initialising lasers
   }
   else
       print_lcd ("Laser init.:   ¯    ", LINE_2, 0);


   #ifndef DEV

   // set-up left resonant scanner settings
   i = i2c_dac  (LEFT_SCANNER, SCANNER_AMPLITUDE, 0xff);   // kick start scanner
   delay_10ms (100);                                       // wait to settle
   i         +=        2*(i2c_dac         (LEFT_SCANNER,         SCANNER_AMPLITUDE,
system_settings.left_scanner.amplitude));
   i          +=         4*(i2c_dac          (LEFT_SCANNER,          SCANNER_ALARM,
system_settings.left_scanner.low_alarm));
   i          +=         8*(i2c_dac          (LEFT_SCANNER,          POS_THRESHOLD,
system_settings.left_scanner.upper_threshold));
   i          +=        16*(i2c_dac          (LEFT_SCANNER,          NEG_THRESHOLD,
system_settings.left_scanner.lower_threshold));

   #else
       i = 1;
   #endif

   if (i != 0)
   {
       print_lcd ("LHS scan init: X-   ", LINE_3, 0);
       strcpy (temp_str, itoa (i, BASE_10));
       print_lcd (temp_str, LINE_3, 17);               // display error no.
       error += 4;         // error initialising LHS scanner
   }
   else
      print_lcd  ("LHS scan init: ¯    ", LINE_3, 0);


   #ifndef DEV

   // set-up right resonant scanner settings
   i = i2c_dac  (RIGHT_SCANNER, SCANNER_AMPLITUDE, 0xff);   // kick start scanner
   delay_10ms (100);                                       // wait to settle
   i         +=        2*(i2c_dac         (RIGHT_SCANNER,         SCANNER_AMPLITUDE,
system_settings.right_scanner.amplitude));
   i          +=         4*(i2c_dac          (RIGHT_SCANNER,          SCANNER_ALARM,
system_settings.right_scanner.low_alarm));
   i          +=         8*(i2c_dac          (RIGHT_SCANNER,          POS_THRESHOLD,
system_settings.right_scanner.upper_threshold));
   i          +=        16*(i2c_dac          (RIGHT_SCANNER,          NEG_THRESHOLD,
system_settings.right_scanner.lower_threshold));

   #else
       i = 1;
   #endif
```

```
   if (i != 0)
   {
       print_lcd ("RHS scan init: X-   ", LINE_4, 0);
       strcpy (temp_str, itoa (i, BASE_10));
       print_lcd (temp_str, LINE_4, 17);               // display error no.
       error += 8;         // error initialising RHS scanner
   }
   else
       print_lcd ("RHS scan init: ¯    ", LINE_4, 0);


   #ifndef DEV

   // set-up top resonant scanner settings
   i = i2c_dac  (TOP_SCANNER,   SCANNER_AMPLITUDE, 0xff);  // kick start scanner
   delay_10ms (100);                                       // wait to settle
   i       +=       2*(i2c_dac       (TOP_SCANNER,              SCANNER_AMPLITUDE,
system_settings.top_scanner.amplitude));
   i        +=        4*(i2c_dac        (TOP_SCANNER,               SCANNER_ALARM,
system_settings.top_scanner.low_alarm));
   i        +=        8*(i2c_dac        (TOP_SCANNER,               POS_THRESHOLD,
system_settings.top_scanner.upper_threshold));
   i        +=       16*(i2c_dac        (TOP_SCANNER,               NEG_THRESHOLD,
system_settings.top_scanner.lower_threshold));

   #else
       i = 1;
   #endif

   delay_10ms (100);           // wait
   clear_lcd ();               // clear ready for next screen set

   if (i != 0)
   {
       print_lcd ("Top scan init: X-   ", LINE_1, 0);
       strcpy (temp_str, itoa (i, BASE_10));
       print_lcd (temp_str, LINE_1, 17);               // display error no.
       error += 16;        // error initialising top scanner
   }
   else
       print_lcd ("Top scan init: ¯    ", LINE_1, 0);


   // set-up galvanometer settings
   set_dtoas (system_settings.galvo_offset);



   // check to see if all resonant scanners are ok
   print_lcd ("Top Scanner  : ", LINE_2, 0);
   print_lcd ("Left Scanner : ", LINE_3, 0);
   print_lcd ("Right Scanner: ", LINE_4, 0);

   if (TOP_SCANNER_ALARM == OFF)           // check top resonant scanner
       print_lcd ("¯", LINE_2, 15);
   else
   {
       print_lcd ("X", LINE_2, 15);
       error |= 32;                                // scanner not reached correct
amplitude
   }

   if (LEFT_SCANNER_ALARM == OFF)          // check left resonant scanner
       print_lcd ("¯", LINE_3, 15);
   else
   {
       print_lcd ("X", LINE_3, 15);
       error |= 32;                                // scanner not reached correct
amplitude
   }

   if (RIGHT_SCANNER_ALARM == OFF)         // check right resonant scanner
       print_lcd ("¯", LINE_4, 15);
   else
   {
       print_lcd ("X", LINE_4, 15);
```

```
        error |= 32;                                  // scanner not reached correct
amplitude
    }


    // check all 6 laser dc power supplies are ok
    delay_10ms (250);
    clear_lcd ();

    print_lcd (" LASER DC PSUs:     ", LINE_1, 0);
    print_lcd (" L1 L2 L3  R1 R2 R3 ", LINE_2, 0);
    print_lcd (lcd_text[2], LINE_3, 0);                 // print line on lcd

    #ifndef DEV
        i = i2c_io_read (I2C_PORT2);
    #else
        i = 0xff;
    #endif

    if ((i & 0x01) == 0x01)                 // check laser dc psu #1
        print_lcd ("X", LINE_4, 2);
    else
        print_lcd ("¯", LINE_4, 1);

    if ((i & 0x02) == 0x02)                 // check laser dc psu #2
        print_lcd ("X", LINE_4, 5);
    else
        print_lcd ("¯", LINE_4, 4);

    if ((i & 0x04) == 0x04)                 // check laser dc psu #3
        print_lcd ("X", LINE_4, 8);
    else
        print_lcd ("¯", LINE_4, 7);

    if ((i & 0x08) == 0x08)                 // check laser dc psu #4
        print_lcd ("X", LINE_4, 12);
    else
        print_lcd ("¯", LINE_4, 11);

    if ((i & 0x10) == 0x10)                 // check laser dc psu #5
        print_lcd ("X", LINE_4, 15);
    else
        print_lcd ("¯", LINE_4, 14);

    if ((i & 0x20) == 0x20)                 // check lasee dc psu #6
        print_lcd ("X", LINE_4, 18);
    else
        print_lcd ("¯", LINE_4, 17);

    if ((i & 0x3f) != 0)
        error += 64;                        // falult dc laser supply

    delay_10ms (250);                       // delay for 1 second
    clear_lcd ();
    print_lcd (lcd_text[10], LINE_4, 0);    // Line of separaters

    disarm_system ();                       // clear laser safety relay
    stop_firing_lasers ();                  // set laser enable gate low
    clear_test_fire1();                     // clear test fire 1 signal
    clear_test_fire2();                     // clear test fire 2 signal

    GREEN_LED = ON;                          // set green status led

    return (error);                         // return error status
}


/*************************************************************************/
/*      Name:           arm_system                                       */
/*      Description:    enables laser safety relay                       */
/*      Inputs:         -                                                */
/*      Return Value:   -                                                */
/*************************************************************************/
```

```
void arm_system (void)
{
    PORT_C |= 0x02;                     // laser enable relay closed - Relay 4

    RED_LED = ON;                       // show red - armed led on CPU board
    print_lcd (lcd_text[5], LINE_1, 0); // Status: ARMED
    print_lcd ("A", LINE_4, 8);         // show armed flag on LCD
}


/*************************************************************************/
/*      Name:           disarm_system                                    */
/*      Description:    diables laser safety relay                       */
/*      Input:          -                                                */
/*      Return Value:   -                                                */
/*************************************************************************/

void disarm_system (void)
{
    PORT_C &= 0xfd;                     // laser enable relay opened - Relay 4

    GREEN_LED = ON;                     // show green - safe led on CPU board
    print_lcd (lcd_text[6], LINE_1,0);  // status disarmed
    print_lcd ("D", LINE_4, 8);         // show disarmed flag on LCD

    clear_laser_ready();                // reset system ready flag to Staalkat
}


/*************************************************************************/
/*      Name:           combine_lasers                                   */
/*      Description:    moves mirrors out for the return stroke          */
/*      Input:          -                                                */
/*      Return Value:   -                                                */
/*************************************************************************/

void combine_lasers (void)
{
    PORT_C |= 0x04;                 // mirror relay closed      - Relay 3

    A1_PIO2 = 1;
    A0_PIO2 = 0;
    PIO2 |= 0x80;                   // mirror move signal to laser controller

    mirror_state = COMBINED;
    print_lcd ("C", LINE_4, 16);    // show laser combined flag on LCD

    send_rs232_data ("E", 1);
}


/*************************************************************************/
/*      Name:           separate_lasers                                  */
/*      Description:    moves the mirrors to the home position           */
/*      Input:          -                                                */
/*      Return Value:                                                    */
/*************************************************************************/

void separate_lasers (void)
{
    PORT_C &= 0xfb;                 // mirror relay opened      - Relay 3

    A1_PIO2 = 1;
    A0_PIO2 = 0;
    PIO2 &= 0x7f;                   // mirror move signal to laser controller

    mirror_state = SEPARATED;
    print_lcd ("S", LINE_4, 16);    // show laser separated flag on LCD

    send_rs232_data ("F", 1);
}
```

```
/*************************************************************************/
/*      Name:           fire_lasers                                      */
/*      Description:    sets laser gate pulse high to fire lasers        */
/*      Input:          -                                                */
/*      Return Value:   unsigned char                                    */
/*                      0 = OK                                           */
/*************************************************************************/

unsigned char fire_lasers (void)
{
   if (check_faults() != 0)
       return (1);

   if ((PORT_B & U_D) == DOWN)         // check carraige in up position
       return (2);

   if ((PORT_B & E_STOP) == E_STOP)    // check e-stop
       return (3);

   if ((PORT_B & STAALKAT_READY) != STAALKAT_READY) // check staalkat machine OK
       return (4);

   A1_PIO2 = 1;
   A0_PIO2 = 0;
   PIO2 |= 0x40;

   print_lcd (lcd_text[7], LINE_1, 0); // Status: FIRING
   print_lcd ("F", LINE_4, 14);        // show laser firing flag on LCD

   send_rs232_data ("G", 1);           // tell host PC lasers are firing

   return (OK);
}


/*************************************************************************/
/*      Name:           stop_firing_lasers                               */
/*      Description:    clears laser gate signal to low to stop firing   */
/*      Input:          -                                                */
/*      Return Value:   -                                                */
/*************************************************************************/

void stop_firing_lasers (void)
{
   A1_PIO2 = 1;
   A0_PIO2 = 0;
   PIO2 &= 0xBf;

   print_lcd (lcd_text[5], LINE_1, 0); // Status: ARMED
   print_lcd (".", LINE_4, 14);        // show laser not firing flag on LCD

   marking_egg = OFF;

   send_rs232_data ("H", 1);           // tell host pc lasers stopped firing
}


/*************************************************************************/
/*      Name:           encoder_int                                      */
/*      Description:    encoder interrupt routine                        */
/*      Input:          -                                                */
/*      Return Value:   -                                                */
/*************************************************************************/

void encoder_int(void) interrupt 0
{
   unsigned char j = 0;

   position++;                         // increase position counter

   if (marking_egg == ON && system_ok == OK)   // if lasers firing, track egg
       set_galvos (galvo_count++);
```

```
   if ((PORT_B & FW_BW) == FORWARDS)
   {
       for (j = 0; j < EGGS_PER_ROW; j++)
       {
            if ((egg_position[j] < EGG_CLEANED) && (egg_position[j] > 0))
                egg_position[j]--;
       }
    }
}


/*************************************************************************/
/*      Name:           decode_rs232_data                                */
/*      Description:    decode rs232 data received                       */
/*      Input:          -                                                */
/*      Return Value:   -                                                */
/*************************************************************************/

void decode_rs232_data (void)
{
    unsigned char rsdata[10];
    xdata unsigned long l;
    xdata unsigned char temp[30];
    unsigned char count1;
    unsigned char count2;

    // if first byte is not valid start of header or end byte not end of
    // transmission - ignore
    if (rs232_in_buffer[0] != SOH || rs232_in_buffer[in_buff_len-1] != EOT)
    {
        buffer_full = 0;            // reset buffer flag
        in_buff_len = 0;            // reset length of byffer counter
        rs232_in_buffer[0] = 0;     // clear rs232 input buffer
        return;                     // exit routine
    }

    // extract substitution bytes
    for (count1 = 1, count2 = 0; count1 < in_buff_len; count1++)
    {
        // count1 = 1, miss off 1st byte which is start of header
        // count2 - new string position pointer

        if (rs232_in_buffer[count1] != SUB)     // normal byte found
        {
            if (rs232_in_buffer[count1] != EOT) // and not end of transmission
                rsdata[count2++]  =  rs232_in_buffer[count1];  //  save  into  new
string
        }
        else                                    // substitution byte found
        {
            count1++;                           // strip out substitution byte
            rsdata[count2++]  =  rs232_in_buffer[count1]  ^  0x20;  //  reconstitute
byte
        }
    }
    rsdata[count2] = 0;             // terminate end of new data

    buffer_full = 0;                // reset buffer flag
    in_buff_len = 0;                // reset length of input buffer counter
    rs232_in_buffer[0] = 0;         // clear rs232 input buffer

    switch (rsdata[0])
    {
        // arm system via laser enble realy
        case 'A':   arm_system ();
                    break;

        // disarm system via clearing laser enable relay
        case 'B':   disarm_system ();
                    break;

        // set laser ready relay to signal to Staalkat machine
        case 'C':   set_laser_ready();
                    break;
```

```
// clear laser ready relay to signal that laser system is not ready
case 'D':   clear_laser_ready();
            break;

// combine two lasers for return stroke, clears move mirrors relay
case 'E':   combine_lasers ();
            break;

// separates laser for forward stroke, sets move mirrors relay
case 'F':   separate_lasers ();
            break;

// fire lasers, sets laser fire pulse high
case 'G':   fire_lasers ();
            break;

// stop firing lasers, sets laser fire pulse low
case 'H':   stop_firing_lasers ();
            break;

// set system into automatic mode for normal operation
case 'I':   system_mode = AUTOMATIC_MODE;
            send_rs232_data ("I", 1);
            break;

// set system into manual mode for testing
case 'J':   system_mode = MANUAL_MODE;
            send_rs232_data ("J", 1);
            break;

// reset counters
case 'K':  count1 = i2c_eeprom_count_write (EEPROM, 0x34, 0);
           if (count1 == 0)
               eggs_cleaned = 0;

           count2 = i2c_eeprom_count_write (EEPROM, 0x38, 0);
           if (count2 == 0)
               rows_cleaned = 0;

           if (PORT_C & 0x01 == 0x01)  // laser system is armed and ready
           {
               strcpy (temp, itoa (eggs_cleaned, BASE_10));
               print_lcd (lcd_text[9], LINE_2, 0);
               print_lcd (temp, LINE_2, 8);   // display egg count

               strcpy (temp, itoa (rows_cleaned, BASE_10));
               print_lcd (lcd_text[11], LINE_3, 0);
               print_lcd (temp, LINE_3, 8);   // display row count
           }

           temp[0] = 'K';
           temp[1] = count1;
           temp[2] = count2;
           temp[3] = 0;
           send_rs232_data (temp, 3);
           break;

// set galvonometer scan patterns
case 'L':   system_settings.galvo_scan_profile = rsdata[1];

            temp[0] = 'L';
            temp[1] = system_settings.galvo_scan_profile;
            temp[2] = 0;
            send_rs232_data (temp, 2);
            break;

// set maximum galvonometer amplitudes for both sides
case 'M':   system_settings.galvo_max_amp = rsdata[1];

            temp[0] = 'M';
            temp[1] = system_settings.galvo_max_amp;
            temp[2] = 0;
            send_rs232_data (temp, 2);
            break;
```

```
        // set galvonometer offsets for both sides
        case 'N':   system_settings.galvo_offset = rsdata[1];

                    temp[0] = 'N';
                    temp[1] = system_settings.galvo_offset;
                    temp[2] = 0;
                    send_rs232_data (temp, 2);
                    break;

        // set amplitude of LHS resonant scanner
        case 'O':   system_settings.left_scanner.amplitude = rsdata[1];
                    count1    =    i2c_dac    (LEFT_SCANNER,    SCANNER_AMPLITUDE,
system_settings.left_scanner.amplitude);

                    if (count1 == 0)            // command received and ok
                    {
                       temp[0] = 'O';
                       temp[1] = system_settings.left_scanner.amplitude;
                       temp[2] = 0;
                       send_rs232_data (temp, 2);
                    }
                    break;

        // set amplitude of RHS resonant scanner
        case 'P':   system_settings.right_scanner.amplitude = rsdata[1];
                    count1    =    i2c_dac    (RIGHT_SCANNER,    SCANNER_AMPLITUDE,
system_settings.right_scanner.amplitude);

                    if (count1 == 0)            // command received and ok
                    {
                       temp[0] = 'P';
                       temp[1] = system_settings.right_scanner.amplitude;
                       temp[2] = 0;
                       send_rs232_data (temp, 2);
                    }
                    break;

        // set amplitude of top resonant scanner
        case 'Q':   system_settings.top_scanner.amplitude = rsdata[1];
                    count1    =    i2c_dac    (TOP_SCANNER,    SCANNER_AMPLITUDE,
system_settings.top_scanner.amplitude);

                    if (count1 == 0)            // command received and ok
                    {
                       temp[0] = 'Q';
                       temp[1] = system_settings.top_scanner.amplitude;
                       temp[2] = 0;
                       send_rs232_data (temp, 2);
                    }
                    break;

        // set high laser power
        case 'R':   system_settings.laser_power.high = rsdata[1];
                    count1    =    i2c_e2pot    (DIGITAL_POT1,    LASER_HIGH_POWER,
system_settings.laser_power.high);

                    if (count1 == 0)            // command received and ok
                    {
                       temp[0] = 'R';
                       temp[1] = system_settings.laser_power.high;
                       temp[2] = 0;
                       send_rs232_data (temp, 2);
                    }
                    break;

        // set low 1 laser power
        case 'S':   system_settings.laser_power.low1 = rsdata[1];
                    count1    =    i2c_e2pot    (DIGITAL_POT1,    LASER_LOW_POWER1,
system_settings.laser_power.low1);

                    if (count1 == 0)            // command received and ok
                    {
                       temp[0] = 'S';
                       temp[1] = system_settings.laser_power.low1;
                       temp[2] = 0;
                       send_rs232_data (temp, 2);
                    }
```

```
                    break;

        // set low 2 laser power
        case 'T':   system_settings.laser_power.low2 = rsdata[1];
                    count1    =    i2c_e2pot    (DIGITAL_POT2,    LASER_LOW_POWER2,
system_settings.laser_power.low2);

                    if (count1 == 0)            // command received and ok
                    {
                       temp[0] = 'T';
                       temp[1] = system_settings.laser_power.low2;
                       temp[2] = 0;
                       send_rs232_data (temp, 2);
                    }
                    break;

        // set test laser power
        case 'U':   system_settings.laser_power.test = rsdata[1];
                    count1    =    i2c_e2pot    (DIGITAL_POT2,    LASER_TEST_POWER,
system_settings.laser_power.test);

                    if (count1 == 0)            // command received and ok
                    {
                       temp[0] = 'U';
                       temp[1] = system_settings.laser_power.test;
                       temp[2] = 0;
                       send_rs232_data (temp, 2);
                    }
                    break;

        // set tickle laser power
        case 'V':   system_settings.laser_power.tickle = rsdata[1];
                    count1    =    i2c_e2pot    (DIGITAL_POT2,    LASER_TICKLE_POWER,
system_settings.laser_power.tickle);

                    if (count1 == 0)            // command received and ok
                    {
                       temp[0] = 'V';
                       temp[1] = system_settings.laser_power.tickle;
                       temp[2] = 0;
                       send_rs232_data (temp, 2);
                    }
                    break;

        // set low alarm point for LHS resonant scanner
        case 'W':   system_settings.left_scanner.low_alarm = rsdata[1];
                    count1     =     i2c_dac     (LEFT_SCANNER,     SCANNER_ALARM,
system_settings.left_scanner.low_alarm);

                    if (count1 == 0)            // command received and ok
                    {
                       temp[0] = 'W';
                       temp[1] = system_settings.left_scanner.low_alarm;
                       temp[2] = 0;
                       send_rs232_data (temp, 2);
                    }
                    break;

        // set low alarm point for RHS resonant scanner
        case 'X':   system_settings.right_scanner.low_alarm = rsdata[1];
                    count1     =     i2c_dac     (RIGHT_SCANNER,     SCANNER_ALARM,
system_settings.right_scanner.low_alarm);

                    if (count1 == 0)            // command received and ok
                    {
                       temp[0] = 'X';
                       temp[1] = system_settings.right_scanner.low_alarm;
                       temp[2] = 0;
                       send_rs232_data (temp, 2);
                    }
                    break;

        // set low alarm point for top resonant scanner
        case 'Y':   system_settings.top_scanner.low_alarm = rsdata[1];
                    count1     =     i2c_dac     (TOP_SCANNER,     SCANNER_ALARM,
system_settings.top_scanner.low_alarm);
```

```
                    if (count1 == 0)            // command received and ok
                    {
                       temp[0] = 'Y';
                       temp[1] = system_settings.top_scanner.low_alarm;
                       temp[2] = 0;
                       send_rs232_data (temp, 2);
                    }
                    break;

        // request from host PC to send current status bytes
        case 'Z':   A1_PIO2 = 1;
                    A0_PIO2 = 0;                // set to read PIO2 Port C

                    temp[0] = 'Z';
                    temp[1] = PORT_B;           // get PIO1, Port B data
                    temp[2] = PIO2;             // get PIO2, Port C data
                    temp[2] &= 0x0f;            // mask off top nibble

                    if (LEFT_SCANNER_ALARM == OFF)
                        temp[2] += 0x10;

                    if (RIGHT_SCANNER_ALARM == OFF)
                        temp[2] += 0x20;

                    if (TOP_SCANNER_ALARM == OFF)
                        temp[2] += 0x40;

                    #ifndef DEV
                        temp[3] = i2c_io_read (I2C_PORT2);  // get dc psu status
                        temp[4] = i2c_io_read (I2C_PORT1);  // get laser status
                    #else
                       temp[3] = 0x00;
                       temp[4] = 0x00;
                    #endif

                    temp[5] = 0;                // end of temp string
                    send_rs232_data (temp, 5);  // send data to rs232 port
                    break;

        // set test fire 1 relay
        case '#':   set_test_fire1();
                    send_rs232_data ("#", 1);
                    break;

        // set test fire 2 relay
        case '?':   set_test_fire2();
                    send_rs232_data ("?", 1);
                    break;

        case '$':   clear_test_fire1();         /* clear test fire 1 relay  */
                    clear_test_fire2();         /* clear test fire 2 relay  */
                    send_rs232_data ("$", 1);
                    break;

        case '&':   clear_test_fire1();         // clear test fire 1 relay
                    clear_test_fire2();         // clear test fire 2 relay
                    send_rs232_data ("&", 1);
                    break;

        // set frequency for laser systems
        case '+':   system_settings.laser_power.frequency = rsdata[1];
                    count1    =    i2c_e2pot    (DIGITAL_POT1,    LASER_CLOCK_FREQ,
system_settings.laser_power.frequency);

                    if (count1 == 0)            // command received and ok
                    {
                       temp[0] = '+';
                       temp[1] = system_settings.laser_power.frequency;
                       temp[2] = 0;
                       send_rs232_data (temp, 2);
                    }
                    break;
```

```
        // left scanner upper set point
        case 'a':   system_settings.left_scanner.upper_threshold = rsdata[1];
                    i2c_dac             (LEFT_SCANNER,              POS_THRESHOLD,
system_settings.left_scanner.upper_threshold);
                    break;

        // right scanner upper set point
        case 'b':   system_settings.right_scanner.upper_threshold = rsdata[1];
                    i2c_dac             (RIGHT_SCANNER,             POS_THRESHOLD,
system_settings.right_scanner.upper_threshold);
                    break;

        // top scanner upper set point
        case 'c':   system_settings.top_scanner.upper_threshold = rsdata[1];
                    i2c_dac              (TOP_SCANNER,              POS_THRESHOLD,
system_settings.top_scanner.upper_threshold);
                    break;

        // left scanner lower set point
        case 'd':   system_settings.left_scanner.lower_threshold = rsdata[1];
                    i2c_dac             (LEFT_SCANNER,              NEG_THRESHOLD,
system_settings.left_scanner.lower_threshold);
                    break;

        // right scanner lower set point
        case 'e':system_settings.right_scanner.lower_threshold = rsdata[1];
                    i2c_dac             (RIGHT_SCANNER,             NEG_THRESHOLD,
system_settings.right_scanner.lower_threshold);
                    break;

        // top scanner lower set point
        case 'f':   system_settings.top_scanner.lower_threshold = rsdata[1];
                    i2c_dac              (TOP_SCANNER,              NEG_THRESHOLD,
system_settings.top_scanner.lower_threshold);
                    break;

        // set forward delay value
        case 'k':  system_settings.pulse_delay = rsdata[1];

                    temp[0] = 'k';
                    temp[1] = system_settings.pulse_delay;
                    temp[2] = 0;
                    send_rs232_data (temp, 2);
                    break;

        // set return start firing position
        case 'l':  system_settings.start_pos = rsdata[1];

                    temp[0] = 'l';
                    temp[1] = system_settings.start_pos;
                    temp[2] = 0;
                    send_rs232_data (temp, 2);
                    break;

       // set return end firing position
       case 'm':   system_settings.stop_pos = rsdata[1];

                    temp[0] = 'm';
                    temp[1] = system_settings.stop_pos;
                    temp[2] = 0;
                    send_rs232_data (temp, 2);
                    break;

       // set egg spacing between eggs in pulses
       case 'n':   system_settings.egg_spacing = rsdata[1];

                    temp[0] = 'n';
                    temp[1] = system_settings.egg_spacing;
                    temp[2] = 0;
                    send_rs232_data (temp, 2);
                    break;

       // set width of eggs in pulses
       case 'o':   system_settings.egg_width = rsdata[1];
```

```
                temp[0] = 'o';
                temp[1] = system_settings.egg_width;
                temp[2] = 0;
                send_rs232_data (temp, 2);
                break;

        // reset system to eprom default settings
        case 'p':   system_settings = system_defaults;
                    send_rs232_data ("p", 1);
                    break;

        // send current system settings to host PC
        case 'q':   temp[0] = 'q';

                    temp[1] = system_settings.pulse_delay;
                    temp[2] = system_settings.start_pos;
                    temp[3] = system_settings.stop_pos;
                    temp[4] = system_settings.egg_width;
                    temp[5] = system_settings.egg_spacing;

                    temp[6] = system_settings.galvo_max_amp;
                    temp[7] = system_settings.galvo_offset;
                    temp[8] = system_settings.galvo_scan_profile;

                    temp[9] = system_settings.laser_power.high;
                    temp[10] = system_settings.laser_power.low1;
                    temp[11] = system_settings.laser_power.low2;
                    temp[12] = system_settings.laser_power.test;
                    temp[13] = system_settings.laser_power.tickle;
                    temp[14] = system_settings.laser_power.frequency;

                    temp[15] = system_settings.top_scanner.amplitude;
                    temp[16] = system_settings.left_scanner.amplitude;
                    temp[17] = system_settings.right_scanner.amplitude;

                    temp[18] = system_settings.top_scanner.low_alarm;
                    temp[19] = system_settings.left_scanner.low_alarm;
                    temp[20] = system_settings.right_scanner.low_alarm;
                    l = eggs_cleaned;
                    temp[21] = l % 65536UL;
                    l /= 256UL;                             // knock off lo byte
                    temp[22] = l % 256UL;
                    temp[23] = l / 256UL;
                    l = rows_cleaned;
                    temp[24] = l % 65536UL;
                    l /= 256UL;                             // knock off lo byte
                    temp[25] = l % 256UL;
                    temp[26] = l / 256UL;
                    temp[27] = 0;
                    send_rs232_data (temp, 27);
                    break;

        // return system to eeprom defaults
        case 'r':   system_settings = eeprom_settings;
                    send_rs232_data ("r", 1);
                    break;

        // save all variables into eeprom
         case   's':          count1   =   i2c_eeprom_char_write   (EEPROM,   0x00,
system_settings.pulse_delay);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x02,
system_settings.start_pos);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x04,
system_settings.stop_pos);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x06,
system_settings.egg_width);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x08,
system_settings.egg_spacing);

                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x0a,
system_settings.galvo_max_amp);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x0c,
system_settings.galvo_offset);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x0e,
system_settings.galvo_scan_profile);
```

```
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x10,
system_settings.top_scanner.amplitude);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x12,
system_settings.top_scanner.low_alarm);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x14,
system_settings.top_scanner.upper_threshold);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x16,
system_settings.top_scanner.lower_threshold);

                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x18,
system_settings.left_scanner.amplitude);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x1a,
system_settings.left_scanner.low_alarm);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x1c,
system_settings.left_scanner.upper_threshold);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x1e,
system_settings.left_scanner.lower_threshold);

                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x20,
system_settings.right_scanner.amplitude);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x22,
system_settings.right_scanner.low_alarm);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x24,
system_settings.right_scanner.upper_threshold);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x26,
system_settings.right_scanner.lower_threshold);

                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x28,
system_settings.laser_power.high);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x2a,
system_settings.laser_power.low1);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x2c,
system_settings.laser_power.low2);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x2e,
system_settings.laser_power.test);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x30,
system_settings.laser_power.tickle);
                    count1     +=     i2c_eeprom_char_write     (EEPROM,     0x32,
system_settings.laser_power.frequency);

                    // return success command or not here?
                    if (count1 == 0)
                    {
                       eeprom_settings   =   system_settings;     //   update   eeprom
settings
                       send_rs232_data ("s", 1);
                    }
                    break;

       // Send eprom settings to host computer
       case 't':
                    break;

       // set left galvonometer position manually
       case 'u':   set_left_dtoa (rsdata[1]);
                   send_rs232_data ("u", 1);
                   break;

       // set right galvonometer position manually
       case 'v':   set_right_dtoa (rsdata[1]);
                   send_rs232_data ("v", 1);
                   break;

       default:    break;
    }
}


/*************************************************************************/
/*      Name:          itoa                                              */
/*      Description:   Converts an integer into its ascii representation  */
/*     Input:          value - the number to be converted                */
/*                     radix - the base for the conversion               */
/*     Return Value:   pointer to the end of the string                  */
/*************************************************************************/
```

```
char* itoa (unsigned long value, unsigned char radix)
{
   xdata char new_string[14];              // temporary charater string
   xdata char new_string2[14];             // temporary charater string
   xdata char temp_char;                   // character store used in conversion
   xdata char position = 0;                // current character position
   xdata unsigned long remainder;          // remainder from division

   if (value < 1)
       new_string[position++] = '0';
   else
   {
       for (position = 0; value > 0; )         // loop for all digits in number
       {
           remainder   = value % radix;
           value       = value / radix;

           if ((temp_char = (unsigned char) remainder + '0') > '9')
               temp_char += 'A'-'0'-10;

           new_string[position++] = temp_char; // store the character
       }
   }

   position--;                                 // back to end character

   // invert string
   for (temp_char = 0; position >= 0; temp_char++, position--)
       new_string2[temp_char] = new_string[position];

   new_string2[temp_char] = 0;                 // add string end null char.

   return (new_string2);                       // return ptr to end of string
}



/*************************************************************************/
/*      Name:           set_galvos                                       */
/*      Description:    send set value to left and right d/a converters  */
/*      Input:          unsigned char value                              */
/*      Return Value:   -                                                */
/*************************************************************************/

void set_galvos (unsigned char pulse_number)
{
    switch (system_settings.galvo_scan_profile)
    {
        // set to centre position
        case STATIC_PROFILE:        set_dtoas (127);
                                    break;

        // set to offset position plus incremental pulse position
        case   LINEAR_PROFILE:                      set_dtoas   (pulse_number   +
system_settings.galvo_offset);
                                    break;

        // set to offset position plus incremental pulse position
        case  COMPENSATED_PROFILE:      set_dtoas  (scan_profile[pulse_number]  +
system_settings.galvo_offset);
                                    // do this via algorithm ?
                                    break;
        default:                    break;
    }
    /*unsigned int scale;
    scale = value * system_settings.galvo_max_amp;
    scale = scale / 0xff;

    PIO2 = (unsigned char) scale;  */

    // do we need to scale galvos, add offset here?????
}

/*************************************************************************/
/* End Of Module                                                         */
/*************************************************************************/
```

8.4 Appendix 4: Research System Remote Control Program

The following listings detail the software developed for the trial machine's remote control program written in Microsoft's Visual Basic language (version 4).

Being of a visual / object oriented nature, much of the software code generated by the Visual Basic program pertains to the description of the visual objects (buttons, textboxes and images for example). As this code is quite voluminous and does little to add to the description of the operation of the program, these have been omitted from the listings below for the sake of brevity.

8.4.1 adas1.glo

```
DefInt A-Z

'--- MSComm event constants
Global Const MSCOMM_EV_SEND = 1
Global Const MSCOMM_EV_RECEIVE = 2
Global Const MSCOMM_EV_CTS = 3
Global Const MSCOMM_EV_DSR = 4
Global Const MSCOMM_EV_CD = 5
Global Const MSCOMM_EV_RING = 6
Global Const MSCOMM_EV_EOF = 7

'--- MSComm error code constants
Global Const MSCOMM_ER_BREAK = 1001
Global Const MSCOMM_ER_CTSTO = 1002
Global Const MSCOMM_ER_DSRTO = 1003
Global Const MSCOMM_ER_FRAME = 1004
Global Const MSCOMM_ER_OVERRUN = 1006
Global Const MSCOMM_ER_CDTO = 1007
Global Const MSCOMM_ER_RXOVER = 1008
Global Const MSCOMM_ER_RXPARITY = 1009
Global Const MSCOMM_ER_TXFULL = 1010

'--- Common Dialog constants
Global Const CDERR_CANCEL = 32755
Global Const CF_SCREENFONTS = &H1&

'--- Global variables
Global CancelSend           'Flag to stop sending

'--- RS232 protocol constants
Global Const SOH_DATA = 1
Global Const EOT_DATA = 4
Global Const SUB_DATA = 27
Global Const NUL_DATA = 0

Global q_sent
```

```
#If Win32 Then
    Declare Sub SetWindowPos Lib "user32" (ByVal hWnd As Long, ByVal
hWndInsertAfter As Long, ByVal X As Long, ByVal Y As Long, ByVal cx As Long, ByVal
cy As Long, ByVal wFlags As Long)
#Else
    Declare Sub SetWindowPos Lib "USER" (ByVal hWnd%, ByVal hWndInsertAfter%,
ByVal X%, ByVal Y%, ByVal cx%, ByVal cy%, ByVal wFlags%)
#End If
```

8.4.2 cansend.frm

```
'************************************************
' CANSEND.FRM is a dialog box that allows the user
' to cancel a "Transmit Text File" operation.  This
' is a modeless form that acts modal while allowing
' other processes to continue.
'************************************************

DefInt A-Z
Option Explicit

Const SWP_NOMOVE = &H2
Const SWP_NOSIZE = &H1


Private Sub Command1_Click()
   CancelSend = True
End Sub


Private Sub Form_Activate()
   ' Make this form a floating window that is always on top.
   SetWindowPos hWnd, -1, 0, 0, 0, 0, SWP_NOMOVE Or SWP_NOSIZE
End Sub


Private Sub Form_Deactivate()
   If Not CancelSend Then
      Form2.Show
   End If
End Sub
```

8.4.3 adas1.frm

```
Private Sub COM_PORT_SELECT_Click(Index As Integer)
    If Index = 1 Then
        COM_PORT_SELECT(1).Checked = True
        COM_PORT_SELECT(2).Checked = False
    Else
        COM_PORT_SELECT(1).Checked = False
        COM_PORT_SELECT(2).Checked = True
    End If
    NewPort = Index

    On Error Resume Next
```

```
    OldPort = MSComm1.CommPort
    If NewPort <> OldPort Then                    'if port num changing
        If MSComm1.PortOpen Then            'close the old port
           MSComm1.PortOpen = False
           ReOpen = True
        End If

        MSComm1.CommPort = NewPort          'set new port number

        If Err = 0 Then
           If ReOpen Then
              MSComm1.PortOpen = True
           End If
        End If

        If Err Then
            MsgBox Error$, 48
            MSComm1.CommPort = OldPort
            MSComm1.PortOpen = True
            COM_PORT_SELECT(OldPort).Checked = True
            COM_PORT_SELECT(NewPort).Checked = False
            Exit Sub
        End If
    End If
End Sub


Private Sub default_eeprom_Click(Index As Integer)
response = MsgBox("Are you sure you wish to reset to system EEPROM settings?", 36,
"Reload EEPROM")
    If response = 6 Then
        Send_data ("r")
    End If
End Sub


Private Sub default_eprom_Click(Index As Integer)
response = MsgBox("Are you sure you wish to reset to system default settings?",
36, "Reload Defaults")
    If response = 6 Then
        Send_data ("p")
    End If
End Sub


Private Sub download_settings_Click(Index As Integer)
    Send_data ("q")
End Sub


Private Sub egg_spacing_Change()
    num = Val(egg_spacing.Text)
    If num > 255 Then num = 255
    If num < 0 Then num = 0
    egg_spacing.Text = Str$(num)
    If q_sent = 0 Then
        Send_data ("n" + Chr$(num))
        egg_spacing.ForeColor = &HFF&
    End If
End Sub


Private Sub egg_width_Change()
    num = Val(egg_width.Text)
    If num > 255 Then num = 255
    If num < 0 Then num = 0
    egg_width.Text = Str$(num)
    If q_sent = 0 Then
        Send_data ("o" + Chr$(num))
        egg_width.ForeColor = &HFF&
    End If
End Sub
```

```
Private Sub Exit_button_Click()
    response = MsgBox("Are you sure?", 20, "Exit")

    If response = 6 Then
        Unload Me
        End
    End If

End Sub


Private Sub fire_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As
Single)
If laser_enable.Value = False Then
        MsgBox "Please ensure LASER is enabled.", 48, "Laser Not Enabled"
    Else
        Send_data ("G")
        fire_timer = Enabled
    End If
End Sub


Private Sub fire_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    'Mode.Caption = "Manual"
    'Mode.ForeColor = &HFF00&
    Send_data ("H")
End Sub


Private Sub fire_timer_Timer()
    fire_timer.Enabled = False
    Send_data ("H")
End Sub


Private Sub Form_Load()
    CRLF$ = Chr$(13) + Chr$(10)

    MSComm1.CommPort = 2
    MSComm1.Settings = "9600,N,8,1"
    MSComm1.Handshaking = False
    MSComm1.InputLen = 0
    MSComm1.RThreshold = 1

    On Error Resume Next
    MSComm1.PortOpen = True

    If Err Then
        MsgBox Error$, 48
        MSComm1.PortOpen = False
        COM_PORT_SELECT(1).Checked = False
        Exit Sub
    End If

    send_data_timer.Enabled = True

    galvo_profile.AddItem "(None)", 0
    galvo_profile.AddItem "Linear", 1
    galvo_profile.AddItem "+Ve Compensation", 2

    Send_data ("q")

End Sub


Private Sub Form_Unload(Cancel As Integer)
    If MSComm1.PortOpen Then
        MSComm1.PortOpen = False
    End If
End Sub
```

```
Private Sub forward_delay_Change()
    num = Val(forward_delay.Text)
    If num > 255 Then num = 255
    If num < 0 Then num = 0
    forward_delay.Text = Str$(num)
    If q_sent = 0 Then
        Send_data ("k" + Chr$(num))
        forward_delay.ForeColor = &HFF&
    End If
End Sub


Private Sub galvo_amplitude_Change()
    galvo_amplitude_text.Text = Str$(galvo_amplitude.Value)
    If q_sent = 0 Then
        Send_data ("M" + Chr$(galvo_amplitude.Value))
        galvo_amplitude_text.ForeColor = &HFF&
    End If
End Sub


Private Sub galvo_offset_Change()
    galvo_offset_text.Text = Str$(galvo_offset.Value)
    If q_sent = 0 Then
        Send_data ("N" + Chr$(galvo_offset.Value))
        galvo_offset_text.ForeColor = &HFF&
    End If
End Sub


Private Sub galvo_position_left_Change()
    galvo_position_left_text.Text = Str$(galvo_position_left.Value)
    Send_data ("u" + Chr$(galvo_position_left.Value))
End Sub


Private Sub galvo_position_right_Change()
    galvo_position_right_text.Text = Str$(galvo_position_right.Value)
    Send_data ("v" + Chr$(galvo_position_right.Value))
End Sub


Private Sub galvo_profile_Click()
    output_value = galvo_profile.ListIndex
    If q_sent = 0 Then
        Send_data ("L" + Chr$(output_value))
        galvo_profile.ForeColor = &HFF&
    End If
End Sub


Private Sub Help_Click()
    CMDialog1.HelpFile = "eggwash.hlp"
    CMDialog1.HelpCommand = HELP_INDEX
    CMDialog1.Action = 6
End Sub


Private Sub Help_menu_selection_Click(Index As Integer)
    If Index = 5 Then
        About.Show MODAL
    ElseIf Index = 1 Then
        CMDialog1.HelpCommand = &H4 'help on help
        CMDialog1.Action = 6
    ElseIf Index = 2 Then
        CMDialog1.HelpFile = "litewash.hlp"
        CMDialog1.HelpCommand = &H3  'contents
        CMDialog1.Action = 6
    ElseIf Index = 3 Then
        CMDialog1.HelpFile = "litewash.hlp"
        CMDialog1.HelpCommand = &H105  'search
        CMDialog1.Action = 6
    End If
End Sub
```

```
Private Sub laser_enable_Click(Value As Integer)
    If q_sent = 0 Then
        If laser_enable.Value = False Then
            Send_data ("B")
        Else
            Send_data ("A")
        End If
    End If
End Sub


Private Sub laser_frequency_Change()
    laser_frequency_text.Text = Str$(laser_frequency.Value)
    If q_sent = 0 Then
        Send_data ("+" + Chr$(laser_frequency.Value))
        laser_frequency_text.ForeColor = &HFF&
    End If
End Sub


Private Sub laser_power_high_Change()
    laser_power_high_text.Text = Str$(laser_power_high.Value)
    If q_sent = 0 Then
        Send_data ("R" + Chr$(laser_power_high.Value))
        laser_power_high_text.ForeColor = &HFF&
    End If
End Sub


Private Sub laser_power_low1_Change()
    laser_power_low1_text.Text = Str$(laser_power_low1.Value)
    If q_sent = 0 Then
        Send_data ("S" + Chr$(laser_power_low1.Value))
        laser_power_low1_text.ForeColor = &HFF&
    End If
End Sub


Private Sub laser_power_low2_Change()
    laser_power_low2_text.Text = Str$(laser_power_low2.Value)
    If q_sent = 0 Then
        Send_data ("T" + Chr$(laser_power_low2.Value))
        laser_power_low2_text.ForeColor = &HFF&
    End If
End Sub


Private Sub laser_power_test_Change()
    laser_power_test_text.Text = Str$(laser_power_test.Value)
    If q_sent = 0 Then
        Send_data ("U" + Chr$(laser_power_test.Value))
        laser_power_test_text.ForeColor = &HFF&
    End If
End Sub


Private Sub laser_power_tickle_Change()
    laser_power_tickle_text.Text = Str$(laser_power_tickle.Value)
    If q_sent = 0 Then
        Send_data ("V" + Chr$(laser_power_tickle.Value))
        laser_power_tickle_text.ForeColor = &HFF&
    End If
End Sub


Private Sub save_current_eeprom_Click(Index As Integer)
response = MsgBox("Are you sure you wish to save all current settings to the
EEPROM?", 36, "Save Current Settings")
    If response = 6 Then
        Send_data ("s")
    End If
End Sub
```

```
Private Sub scanner_left_alarm_Change()
    scanner_left_alarm_text.Text = Str$(scanner_left_alarm.Value)
    If q_sent = 0 Then
        Send_data ("W" + Chr$(scanner_left_alarm.Value))
        scanner_left_alarm_text.ForeColor = &HFF&
    End If
End Sub


Private Sub scanner_right_alarm_Change()
    scanner_right_alarm_text.Text = Str$(scanner_right_alarm.Value)
    If q_sent = 0 Then
        Send_data ("X" + Chr$(scanner_right_alarm.Value))
        scanner_right_alarm_text.ForeColor = &HFF&
    End If
End Sub


Private Sub scanner_top_alarm_Change()
    scanner_top_alarm_text.Text = Str$(scanner_top_alarm.Value)
    If q_sent = 0 Then
        Send_data ("Y" + Chr$(scanner_top_alarm.Value))
        scanner_top_alarm_text.ForeColor = &HFF&
    End If
End Sub


Private Sub test_fire_MouseDown(Button As Integer, Shift As Integer, X As Single,
Y As Single)
    If laser_enable.Value = False Then
        MsgBox "Please ensure LASER is enabled.", 48, "Laser Not Enabled"
    Else
        If spare1_relay.Value = True Then Send_data ("#") 'LASER 1 FIRE
        If spare2_relay.Value = True Then Send_data ("?") 'LASER 2 FIRE
    End If
End Sub


Private Sub test_fire_MouseUp(Button As Integer, Shift As Integer, X As Single, Y
As Single)
    Mode.Caption = "Manual"
    Mode.ForeColor = &HFF00&
    Send_data ("&") 'LASER 1 OFF
    Send_data ("$") 'LASER 2 OFF
End Sub


Private Sub laser_freq_change()
    Laser_freq_text = Str$(laser_freq.Value)

    Select Case laser_freq.Value
    Case 4: output_value = 63
    Case 5: output_value = 47
    Case 6: output_value = 33
    Case 7: output_value = 23
    Case 8: output_value = 16
    Case 9: output_value = 10
    Case 10: output_value = 5
    End Select

    Call laspow_high_change
    Call laspow_low1_change
    Call laspow_low2_change
    Call laspow_test_change
    Send_data ("+" + Chr$(output_value))
End Sub


Private Sub laser_ready_Click(Value As Integer)
    If q_sent = 0 Then
        If laser_ready.Value = False Then
            Send_data ("D")
        Else
            Send_data ("C")
        End If
    End If
End Sub
```

```
Private Sub laspow_high_change()
    Laspow_high_text = Str$(laspow_high.Value)

    output_value = laspow_high.Value
    If laspow_high.Value < output_value_min Then
        output_value = output_value_min / 2
    End If
    output_value = output_value * (output_value_max / 63)
    Send_data ("R" + Chr$(output_value))
End Sub


Private Sub laspow_low1_change()
    Laspow_low1_text = Str$(laspow_low1.Value)

    Select Case laser_freq.Value
    Case 4: output_value_max = 63
            output_value_min = 6
    Case 5: output_value_max = 55
            output_value_min = 6
    Case 6: output_value_max = 45
            output_value_min = 7
    Case 7: output_value_max = 38
            output_value_min = 9
    Case 8: output_value_max = 34
            output_value_min = 10
    Case 9: output_value_max = 30
            output_value_min = 11
    Case 10: output_value_max = 26
            output_value_min = 12
    End Select

    output_value = laspow_low1.Value / 2
    If laspow_low1.Value < output_value_min Then
        output_value = output_value_min / 2
    End If
    output_value = output_value * (output_value_max / 63)
    Send_data ("S" + Chr$(output_value))
End Sub


Private Sub laspow_low2_change()
    Laspow_low2_text = Str$(laspow_low2.Value)
    Select Case laser_freq.Value
    Case 4: output_value_max = 57
            output_value_min = 6
    Case 5: output_value_max = 47
            output_value_min = 7
    Case 6: output_value_max = 39
            output_value_min = 9
    Case 7: output_value_max = 33
            output_value_min = 10
    Case 8: output_value_max = 29
            output_value_min = 11
    Case 9: output_value_max = 26
            output_value_min = 13
    Case 10: output_value_max = 23
            output_value_min = 14
    End Select

    output_value = laspow_low2.Value / 2
    If laspow_low2.Value < output_value_min Then
        output_value = output_value_min / 2
    End If
    output_value = output_value * (output_value_max / 63)
    Send_data ("T" + Chr$(output_value))
End Sub
```

```
Private Sub laspow_test_change()
    Laspow_test_text = Str$(laspow_test.Value)

    Select Case laser_freq.Value
    Case 4: output_value_max = 58
            output_value_min = 4
    Case 5: output_value_max = 48
            output_value_min = 4
    Case 6: output_value_max = 40
            output_value_min = 5
    Case 7: output_value_max = 34
            output_value_min = 6
    Case 8: output_value_max = 30
            output_value_min = 7
    Case 9: output_value_max = 26
            output_value_min = 8
    Case 10: output_value_max = 23
            output_value_min = 9
    End Select

    output_value = laspow_test.Value / 2
    If laspow_test.Value < output_value_min Then
        output_value = output_value_min / 2
    End If
    output_value = output_value * (output_value_max / 63)
    Send_data ("U" + Chr$(output_value))
End Sub


Private Sub laspow_tickle_change()
    Laspow_tickle_text = Str$(laspow_tickle.Value)
    output_value = laspow_tickle.Value + 2
    Send_data ("V" + Chr$((laspow_tickle.Value * 6) - 3))
End Sub


Private Sub manual_click()
    If Mode.Caption = "Auto" Then

        Send_data ("J")

        Mode.Caption = "Manual"
        manual.Caption = "&Auto"

        mirror_1.Enabled = True
        mirror_2.Enabled = True
        laser_enable.Enabled = True
        laser_ready.Enabled = True
        spare1_relay.Enabled = True
        spare2_relay.Enabled = True
        Relay_Outputs.Enabled = True

        Label39.Enabled = True
        Label17.Enabled = True
        Label24.Enabled = True
        galvo_position_left.Enabled = True
        galvo_position_right.Enabled = True
        galvo_position_left_text.Enabled = True
        galvo_position_right_text.Enabled = True

        fire.Enabled = True
        test_fire.Enabled = True
    Else
        Beep
        response = MsgBox("Ensure machine is safe for automatic operation.", 49,
"Automatic Check")
        If response = 1 Then
            Send_data ("I")

            Mode.Caption = "Auto"
            manual.Caption = "&Manual"

            fire.Enabled = False
            test_fire.Enabled = False
            mirror_1.Enabled = False
            mirror_2.Enabled = False
```

```
            'laser_ready.Value = True
            'laser_enable.Value = True
            laser_enable.Enabled = False
            laser_ready.Enabled = False
            spare1_relay.Enabled = False
            spare2_relay.Enabled = False
            Relay_Outputs.Enabled = False

            Label39.Enabled = False
            Label17.Enabled = False
            Label24.Enabled = False
            galvo_position_left.Enabled = False
            galvo_position_right.Enabled = False
            galvo_position_left_text.Enabled = False
            galvo_position_right_text.Enabled = False
        End If
    End If
End Sub


Private Sub mirror_1_Click(Value As Integer)
    If q_sent = 0 Then Send_data ("E")
End Sub


Private Sub mirror_2_Click(Value As Integer)
    If q_sent = 0 Then Send_data ("F")
End Sub


Private Sub MSComm1_OnComm()

    Dim EVMsg$
    Dim ERMsg$

    '--- Branch according to the CommEvent Prop..
    Select Case MSComm1.CommEvent
        '--- Event messages
        Case MSCOMM_EV_RECEIVE
            instring2$ = ""

            ' get data from input buffer
            Do
                instring1$ = MSComm1.Input                    ' get next data if
available
                instring2$ = instring2$ + instring1$    ' add to main input buffer
            Loop Until Right$(instring1$, 1) = Chr$(EOT_DATA) ' stop when EOT
received

            str_length = Len(instring2$)                ' find main string length
            instring1$ = instring2$                         ' make a copy of input
string
            instring2$ = ""                               ' clear input string for
later use

            ' strip out substitution bytes
            For X = 2 To str_length                     ' ignore 1st byte (SOH)
                char_string$ = Mid$(instring1$, X, 1)   ' get character
                    If char_string$ <> Chr$(SUB_DATA) Then
                        If char_string$ <> Chr$(EOT_DATA) Then
                            instring2$ = instring2$ + char_string$    ' add if
normal valid byte
                        End If
                    Else                                ' substitution byte
                        X = X + 1                       ' point to data byte
                        char_string$ = Mid$(instring1$, X, 1) ' get data byte
                        char_data = Asc(char_string$)   ' convert to numeric data
                        char_data = char_data Xor 32       ' convert back from
subsitution
                        char_string$ = Chr$(char_data)     ' convert back to
character
                        instring2$ = instring2$ + char_string$   ' add to main
buffer
                    End If
            Next X
```

```
' ACT ON DATA FOUND.....................................>


Select Case Left$(instring2$, 1)

    Case "+"    ' update laser frequency
        q_sent = 1
        laser_frequency.Value = Asc(Mid$(instring2$, 2, 1))
        laser_frequency_text.ForeColor = &H0&
        q_sent = 0

    Case "E"    ' show combined lasers
        q_sent = 1
        mirror_1.Value = True
        q_sent = 0

    Case "F"    ' show separated lasers
        q_sent = 1
        mirror_2.Value = True
        q_sent = 0

    Case "G"    ' show laser firing
        Mode.Caption = "Laser Firing!"
        Mode.ForeColor = &HFF&

    Case "H"    ' show laser not firing - return to normal mode
        If manual.Caption = "&Manual" Then
            Mode.Caption = "Auto"
        Else
            Mode.Caption = "Manual"
        End If
        Mode.ForeColor = &HFFFF00

    Case "K"    ' reset counters
        If Asc(Mid$(instring2$, 2, 1)) = 0 Then
            egg_num.ForeColor = &HFFFF00
            egg_num.Text = " 0"
        End If
        If Asc(Mid$(instring2$, 3, 1)) = 0 Then
            row_num.ForeColor = &HFFFF00
            row_num.Text = " 0"
        End If

    Case "L"    ' update galvo profile
        q_sent = 1
        galvo_profile.ListIndex = Asc(Mid$(instring2$, 2, 1))
        galvo_profile.ForeColor = &H0&
        q_sent = 0

    Case "M"    ' update galvo amplitude
        q_sent = 1
        galvo_amplitude.Value = Asc(Mid$(instring2$, 2, 1))
        galvo_amplitude_text.ForeColor = &H0&
        q_sent = 0

    Case "N"    ' update galvo offset
        q_sent = 1
        galvo_offset.Value = Asc(Mid$(instring2$, 2, 1))
        galvo_offset_text.ForeColor = &H0&
        q_sent = 0

    Case "O"    ' update scanner left amplitude
        q_sent = 1
        scanner_left_amplitude.Value = Asc(Mid$(instring2$, 2, 1))
        scanner_left_amplitude_text.ForeColor = &H0&
        q_sent = 0

    Case "P"    ' update scanner right amplitude
        q_sent = 1
        scanner_right_amplitude.Value = Asc(Mid$(instring2$, 2, 1))
        scanner_right_amplitude_text.ForeColor = &H0&
        q_sent = 0
```

```
Case "Q"    ' update scanner top amplitude
    q_sent = 1
    scanner_top_amplitude.Value = Asc(Mid$(instring2$, 2, 1))
    scanner_top_amplitude_text.ForeColor = &H0&
    q_sent = 0


Case "R"    ' update laser high level
    q_sent = 1
    laser_power_high.Value = Asc(Mid$(instring2$, 2, 1))
    laser_power_high_text.ForeColor = &H0&
    q_sent = 0

Case "S"    ' update laser low1 level
    q_sent = 1
    laser_power_low1.Value = Asc(Mid$(instring2$, 2, 1))
    laser_power_low1_text.ForeColor = &H0&
    q_sent = 0

Case "T"    ' update laser low2 level
    q_sent = 1
    laser_power_low2.Value = Asc(Mid$(instring2$, 2, 1))
    laser_power_low2_text.ForeColor = &H0&
    q_sent = 0

Case "U"    ' update laser test level
    q_sent = 1
    laser_power_test.Value = Asc(Mid$(instring2$, 2, 1))
    laser_power_test_text.ForeColor = &H0&
    q_sent = 0

Case "V"    ' update laser tickle level
    q_sent = 1
    laser_power_tickle.Value = Asc(Mid$(instring2$, 2, 1))
    laser_power_tickle_text.ForeColor = &H0&
    q_sent = 0

Case "W"    ' update scanner left alarm
    q_sent = 1
    scanner_left_alarm.Value = Asc(Mid$(instring2$, 2, 1))
    scanner_left_alarm_text.ForeColor = &H0&
    q_sent = 0

Case "X"    ' update scanner right alarm
    q_sent = 1
    scanner_right_alarm.Value = Asc(Mid$(instring2$, 2, 1))
    scanner_right_alarm_text.ForeColor = &H0&
    q_sent = 0

Case "Y"    ' update scanner top alarm
    q_sent = 1
    scanner_top_alarm.Value = Asc(Mid$(instring2$, 2, 1))
    scanner_top_alarm_text.ForeColor = &H0&
    q_sent = 0

Case "k"    ' update forward delay
    q_sent = 1
    forward_delay.Text = Str$(Asc(Mid$(instring2$, 2, 1)))
    forward_delay.ForeColor = &H0&
    q_sent = 0

Case "l"    ' update return start
    q_sent = 1
    return_start.Text = Str$(Asc(Mid$(instring2$, 2, 1)))
    return_start.ForeColor = &H0&
    q_sent = 0

Case "m"    ' update return end
    q_sent = 1
    return_end.Text = Str$(Asc(Mid$(instring2$, 2, 1)))
    return_end.ForeColor = &H0&
    q_sent = 0
```

```
                Case "n"    ' update egg spacing
                    q_sent = 1
                    egg_spacing.Text = Str$(Asc(Mid$(instring2$, 2, 1)))
                    egg_spacing.ForeColor = &H0&
                    q_sent = 0

                Case "o"    ' update egg width
                    q_sent = 1
                    egg_width.Text = Str$(Asc(Mid$(instring2$, 2, 1)))
                    egg_width.ForeColor = &H0&
                    q_sent = 0

                Case "p"    ' system reset to default eprom settings
                    Beep
                    response = MsgBox("System has been reset to default EPROM
settings", 49, "Reset to Defaults")
                    Send_data ("q")

                Case "r"    ' system rest to eeprom settings
                    response = MsgBox("System has been reset to EEPROM settings",
49, "Reset to EEPROM settings")
                    Send_data ("q")

                Case "s"    ' system settings have been saved to eeprom
                    response = MsgBox("System settins have been saved to EEPROM",
49, "Settings saved to EEPROM")
                    Send_data ("q")

                Case "w"    ' increment egg counter
                    num = Val(egg_num.Text)
                    num = num + 1
                    egg_num.Text = Str$(num)

                Case "x"    ' update egg and row counters
                    num = Asc(Mid$(instring2$, 2, 1))
                    num = num + (256 * Asc(Mid$(instring2$, 3, 1)))
                    num = num + (65536 * Asc(Mid$(instring2$, 4, 1)))
                    egg_num.Text = Str$(num)

                    num = Asc(Mid$(instring2$, 5, 1))
                    num = num + (256 * Asc(Mid$(instring2$, 6, 1)))
                    num = num + (65536 * Asc(Mid$(instring2$, 7, 1)))
                    row_num.Text = Str$(num)

                Case "q"
                    ' response to request micros data settings
                    q_sent = 1

                    forward_delay.Text = Str$(Asc(Mid$(instring2$, 2, 1)))
                    return_start.Text = Str$(Asc(Mid$(instring2$, 3, 1)))
                    return_end.Text = Str$(Asc(Mid$(instring2$, 4, 1)))
                    egg_width.Text = Str$(Asc(Mid$(instring2$, 5, 1)))
                    egg_spacing.Text = Str$(Asc(Mid$(instring2$, 6, 1)))

                    galvo_amplitude.Value = Asc(Mid$(instring2$, 7, 1))
                    galvo_offset.Value = Asc(Mid$(instring2$, 8, 1))
                    galvo_profile.ListIndex = Asc(Mid$(instring2$, 9, 1))

                    laser_power_high.Value = Asc(Mid$(instring2$, 10, 1))
                    laser_power_low1.Value = Asc(Mid$(instring2$, 11, 1))
                    laser_power_low2.Value = Asc(Mid$(instring2$, 12, 1))
                    laser_power_test.Value = Asc(Mid$(instring2$, 13, 1))
                    laser_power_tickle.Value = Asc(Mid$(instring2$, 14, 1))
                    laser_frequency.Value = Asc(Mid$(instring2$, 15, 1))

                    scanner_top_amplitude.Value = Asc(Mid$(instring2$, 16, 1))
                    scanner_left_amplitude.Value = Asc(Mid$(instring2$, 17, 1))
                    scanner_right_amplitude.Value = Asc(Mid$(instring2$, 18, 1))
                    scanner_top_alarm.Value = Asc(Mid$(instring2$, 19, 1))
                    scanner_left_alarm.Value = Asc(Mid$(instring2$, 20, 1))
                    scanner_right_alarm.Value = Asc(Mid$(instring2$, 21, 1))

                    num = Asc(Mid$(instring2$, 22, 1))
                    num = num + (256 * Asc(Mid$(instring2$, 23, 1)))
                    num = num + (65536 * Asc(Mid$(instring2$, 24, 1)))
                    egg_num.Text = Str$(num)
```

```
    num = Asc(Mid$(instring2$, 25, 1))
    num = num + (256 * Asc(Mid$(instring2$, 26, 1)))
    num = num + (65536 * Asc(Mid$(instring2$, 27, 1)))
    row_num.Text = Str$(num)

    q_sent = 0

Case "Z"
    ' response to request control port settings
    data_input = Asc(Mid$(instring2$, 2, 1))

    If (data_input And 1) = 0 Then
        sys_ready_on.Visible = True
        sys_ready_off.Visible = False
    Else
        sys_ready_off.Visible = True
        sys_ready_on.Visible = False
    End If

    If (data_input And 2) = 0 Then
        fb_on.Visible = False
        fb_off.Visible = True
    Else
        fb_off.Visible = False
        fb_on.Visible = True
    End If

    If (data_input And 4) = 0 Then
        left_right_on.Visible = True
        left_right_off.Visible = False
    Else
        left_right_off.Visible = True
        left_right_on.Visible = False
    End If

    If (data_input And 8) = 0 Then
        egg_sense_on.Visible = True
        egg_sense_off.Visible = False
    Else
        egg_sense_off.Visible = True
        egg_sense_on.Visible = False
    End If

    If (data_input And 16) = 0 Then
        egg_pos_on.Visible = True
        egg_pos_off.Visible = False
    Else
        egg_pos_off.Visible = True
        egg_pos_on.Visible = False
    End If

    If (data_input And 32) = 32 Then
        e_stop_on.Visible = True
        e_stop_off.Visible = False
        'If mode.text = "Auto" Then
        '    MsgBox "Emergency Stop", 16, "E-Stop"
        'End If
    Else
        e_stop_off.Visible = True
        e_stop_on.Visible = False
    End If

    If (data_input And 64) = 0 Then
        spare1_on.Visible = True
        spare1_off.Visible = False
    Else
        spare1_off.Visible = True
        spare1_on.Visible = False
    End If

    If (data_input And 128) = 0 Then
        spare2_on.Visible = True
        spare2_off.Visible = False
    Else
        spare2_off.Visible = True
        spare2_on.Visible = False
    End If
```

```
' led status for moving mirrors and scanner alarms
data_input = Asc(Mid$(instring2$, 3, 1))

' show status of leds for moving mirror switches

If (data_input And 1) = 0 Then
    mirror1_in_on.Visible = True
    mirror1_in_off.Visible = False
Else
    mirror1_in_off.Visible = True
    mirror1_in_on.Visible = False
End If

If (data_input And 2) = 0 Then
    mirror1_out_on.Visible = True
    mirror1_out_off.Visible = False
Else
    mirror1_out_off.Visible = True
    mirror1_out_on.Visible = False
End If

If (data_input And 4) = 0 Then
    mirror2_in_on.Visible = True
    mirror2_in_off.Visible = False
Else
    mirror2_in_off.Visible = True
    mirror2_in_on.Visible = False
End If

If (data_input And 8) = 0 Then
    mirror2_out_on.Visible = True
    Mirror2_out_off.Visible = False
Else
    Mirror2_out_off.Visible = True
    mirror2_out_on.Visible = False
End If


' show led status for scanner alarms

If (data_input And 16) = 0 Then
    Leftscan_alarm_on.Visible = True
    Leftscan_alarm_off.Visible = False
Else
    Leftscan_alarm_off.Visible = True
    Leftscan_alarm_on.Visible = False
End If

If (data_input And 32) = 0 Then
    Rightscan_alarm_on.Visible = True
    Rightscan_alarm_off.Visible = False
Else
    Rightscan_alarm_off.Visible = True
    Rightscan_alarm_on.Visible = False
End If

If (data_input And 64) = 0 Then
    Topscan_alarm_on.Visible = True
    Topscan_alarm_off.Visible = False
Else
    Topscan_alarm_off.Visible = True
    Topscan_alarm_on.Visible = False
End If

'show dc power supply status
data_input = Asc(Mid$(instring2$, 4, 1))

If (data_input And 1) = 0 Then
    dc_psu1_left_on.Visible = True
    dc_psu1_left_off.Visible = False
Else
    dc_psu1_left_off.Visible = True
    dc_psu1_left_on.Visible = False
End If
```

```
If (data_input And 2) = 0 Then
    dc_psu2_left_on.Visible = True
    dc_psu2_left_off.Visible = False
Else
    dc_psu2_left_off.Visible = True
    dc_psu2_left_on.Visible = False
End If

If (data_input And 4) = 0 Then
    dc_psu3_left_on.Visible = True
    dc_psu3_left_off.Visible = False
Else
    dc_psu3_left_off.Visible = True
    dc_psu3_left_on.Visible = False
End If

If (data_input And 8) = 0 Then
    dc_psu1_right_on.Visible = True
    dc_psu1_right_off.Visible = False
Else
    dc_psu1_right_off.Visible = True
    dc_psu1_right_on.Visible = False
End If

If (data_input And 16) = 0 Then
    dc_psu2_right_on.Visible = True
    dc_psu2_right_off.Visible = False
Else
    dc_psu2_right_off.Visible = True
    dc_psu2_right_on.Visible = False
End If

If (data_input And 32) = 0 Then
    dc_psu3_right_on.Visible = True
    dc_psu3_right_off.Visible = False
Else
    dc_psu3_right_off.Visible = True
    dc_psu3_right_on.Visible = False
End If


' show laser staus on leds
data_input = Asc(Mid$(instring2$, 5, 1))

If (data_input And 1) = 0 Then
    ready_right_on.Visible = True
    ready_right_off.Visible = False
Else
    ready_right_off.Visible = True
    ready_right_on.Visible = False
End If

If (data_input And 2) = 0 Then
    key_right_on.Visible = True
    key_right_off.Visible = False
Else
    key_right_off.Visible = True
    key_right_on.Visible = False
End If

If (data_input And 4) = 0 Then
    water_right_on.Visible = True
    water_right_off.Visible = False
Else
    water_right_off.Visible = True
    water_right_on.Visible = False
End If

If (data_input And 16) = 0 Then
    ready_left_on.Visible = True
    ready_left_off.Visible = False
Else
    ready_left_off.Visible = True
    ready_left_on.Visible = False
End If
```

```
                    If (data_input And 32) = 0 Then
                        key_left_on.Visible = True
                        key_left_off.Visible = False
                    Else
                        key_left_off.Visible = True
                        key_left_on.Visible = False
                    End If

                    If (data_input And 64) = 0 Then
                        water_left_on.Visible = True
                        water_left_off.Visible = False
                    Else
                        water_left_off.Visible = True
                        water_left_on.Visible = False
                    End If

            End Select


        Case MSCOMM_EV_SEND

        Case MSCOMM_EV_CTS
            EVMsg$ = "Change in CTS Detected"
        Case MSCOMM_EV_DSR
            EVMsg$ = "Change in DSR Detected"
        Case MSCOMM_EV_CD
            EVMsg$ = "Change in CD Detected"
        Case MSCOMM_EV_RING
            EVMsg$ = "The Phone is Ringing"
        Case MSCOMM_EV_EOF
            EVMsg$ = "End of File Detected"

        '--- Error messages
        Case MSCOMM_ER_BREAK
            EVMsg$ = "Break Received"
        Case MSCOMM_ER_CTSTO
            ERMsg$ = "CTS Timeout"
        Case MSCOMM_ER_DSRTO
            ERMsg$ = "DSR Timeout"
        Case MSCOMM_ER_FRAME
            EVMsg$ = "Framing Error"
        Case MSCOMM_ER_OVERRUN
            ERMsg$ = "Overrun Error"
        Case MSCOMM_ER_CDTO
            ERMsg$ = "Carrier Detect Timeout"
        Case MSCOMM_ER_RXOVER
            ERMsg$ = "Receive Buffer Overflow"
        Case MSCOMM_ER_RXPARITY
            EVMsg$ = "Parity Error"
        Case MSCOMM_ER_TXFULL
            ERMsg$ = "Transmit Buffer Full"
        Case Else
            ERMsg$ = "Unknown error or event"
    End Select

    If Len(ERMsg$) Then
        '--- Display error messages in an alert
        '    message box.
        Beep
        Ret = MsgBox(ERMsg$, 1, "Press Cancel to Quit, Ok to ignore.")
        ERMsg$ = ""
        '--- If Cancel (2) was pressed
        If Ret = 2 Then
            MSComm1.PortOpen = 0    'Close the port and quit
        End If
    End If
End Sub

Private Sub reset_counts_Click()
    response = MsgBox("Are you sure you wish to reset the counters?", 36, "Reset
Counters")
    If response = 6 Then
        egg_num.ForeColor = &HFF&
        row_num.ForeColor = &HFF&
        Send_data ("K")
    End If
End Sub
```

```
Private Sub scanner_top_amplitude_Change()
    scanner_top_amplitude_text.Text = Str$(scanner_top_amplitude.Value)
    If q_sent = 0 Then
        Send_data ("Q" + Chr$(scanner_top_amplitude.Value))
        scanner_top_amplitude_text.ForeColor = &HFF&
    End If
End Sub


Private Sub return_end_Change()
    num = Val(return_end.Text)
    If num > 255 Then num = 255
    If num < 0 Then num = 0
    return_end.Text = Str$(num)
    If q_sent = 0 Then
        Send_data ("m" + Chr$(num))
        return_end.ForeColor = &HFF&
    End If
End Sub


Private Sub return_start_Change()
    num = Val(return_start.Text)
    If num > 255 Then num = 255
    If num < 0 Then num = 0
    return_start.Text = Str$(num)
    If q_sent = 0 Then
        Send_data ("l" + Chr$(num))
        return_start.ForeColor = &HFF&
    End If
End Sub


Private Sub scanner_right_amplitude_Change()
    scanner_right_amplitude_text.Text = Str$(scanner_right_amplitude.Value)
    If q_sent = 0 Then
        Send_data ("P" + Chr$(scanner_right_amplitude.Value))
        scanner_right_amplitude_text.ForeColor = &HFF&
    End If
End Sub


Private Sub scanner_left_amplitude_Change()
    scanner_left_amplitude_text.Text = Str$(scanner_left_amplitude.Value)
    If q_sent = 0 Then
        Send_data ("O" + Chr$(scanner_left_amplitude.Value))
        scanner_left_amplitude_text.ForeColor = &HFF&
    End If
End Sub


Private Sub Send_data(data_string$)

    Command1$ = Left$(data_string$, 1)      ' get 1st byte to send

    If Len(data_string$) > 1 Then           ' if multiple bytes to send
        data1$ = Mid$(data_string$, 2, 1)   ' get 2nd byte
        If data1$ = Chr$(SUB_DATA) Or data1$ = Chr$(EOT_DATA) Or data1$ = Chr$(0)
Then
            data_asc = Asc(data1$)          ' get byte to substitute
            data_asc = data_asc Xor 32      ' alter byte
            data1$ = Chr$(SUB_DATA) + Chr$(data_asc)    ' andd substitution byte
        End If
        ' send command for multiple bytes
        MSComm1.Output = Chr$(SOH_DATA) + Command1$ + data1$ + Chr$(EOT_DATA)
    Else
        ' send command for single byte
        MSComm1.Output = Chr$(SOH_DATA) + Command1$ + Chr$(EOT_DATA)
    End If

End Sub
```

```
Private Sub send_data_timer_Timer()
    Send_data ("Z")
End Sub


Private Sub upload_ram_Click()
   On Error Resume Next
   Dim hSend, BSize, LF&

    send_data_timer.Enabled = False
    MsgBox "Ensure PC COM port is connected to micro-controller. Then press the
micro reset button.", 48, "Connect COM Lead"

   ' Get the text filename from the user.
   OpenLog.DialogTitle = "Send Hex File"
   OpenLog.Filter = "Hex Files (*.HEX)|*.hex|All Files (*.*)|*.*"
   Do
      OpenLog.filename = ""
      OpenLog.ShowOpen
      If Err = cdlCancel Then Exit Sub
      Temp$ = OpenLog.filename

      ' If the file doesn't exist, go back.
      Ret = Len(Dir$(Temp$))
      If Err Then
         MsgBox Error$, 48
         MsgBox "Ensure PC COM port is re-connected to Light Wash Machine.", 48,
"Connect COM Lead"
         send_data_timer.Enabled = True
         Exit Sub
      End If
      If Ret Then
         Exit Do
      Else
         MsgBox Temp$ + " not found!", 48
      End If
   Loop

   ' Open the log file.
   hSend = FreeFile
   Open Temp$ For Binary Access Read As hSend
   If Err Then
      MsgBox Error$, 48
   Else
      ' Display the Cancel dialog box.
      CancelSend = False
      Form2.Label1.Caption = "Transmitting Hex File - " + Temp$
      Form2.Show
      Form2.progress.Value = 0
      bytes_sent = 0

      ' Read the file in blocks the size of the transmit buffer.
      BSize = MSComm1.OutBufferSize
      LF& = LOF(hSend)
      Do Until EOF(hSend) Or CancelSend
         ' Don't read too much at the end.
         If LF& - Loc(hSend) <= BSize Then
            BSize = LF& - Loc(hSend) + 1
         End If

         ' Read a block of data.
         Temp$ = Space$(BSize)
         Get hSend, , Temp$

         ' Transmit the block.
         MSComm1.Output = Temp$
         If Err Then
            MsgBox Error$, 48
            Exit Do
         End If
```

```
         ' Wait for all the data to be sent.
         Do
            Ret = DoEvents()
         Loop Until MSComm1.OutBufferCount = 0 Or CancelSend

         bytes_sent = bytes_sent + BSize
         Form2.progress.Value = (bytes_sent / LF&) * 100
      Loop
   End If

   Close hSend
   CancelSend = True
   Form2.Hide
   MsgBox "Ensure PC COM port is re-connected to Light Wash Machine.", 48,
"Connect COM Lead"
   send_data_timer.Enabled = True

End Sub
```