MINIATURISED THERAGNOSTIC IRRADIATOR WITH MOTION FEEDBACK

Morgan Wheatley

Bachelor of Engineering
Mechatronic and Electronics Majors

Department of Electronic Engineering
Macquarie University

November 7, 2016

Supervisor: Professor Yves De Deene

ACKNOWLEDGMENTS

I would like to thank my family and friends for helping me throughout my education.

Thank you to Prof. Yves De Deene for the help throughout this project and to Mr David Baer and Peter at Meadowbank TAFE for their help with the design and manufacture of parts for the system.

STATEMENT OF CANDIDATE

I, Morgan Wheatley, declare that this report, submitted as part of the requirement for the award of Bachelor of Engineering in the Department of Engineering, Macquarie University, is entirely my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualification or assessment at any academic institution.

Student's Name: Morgan Wheatley

Student's Signature: Morgan Wheatley (electronic)

Date: 07/11/2016

ABSTRACT

Radiotherapy can help treat cancer in patients but it can also cause damage to healthy surrounding tissue. Tracking tumours within the body and accurately targeting them can result in less damage to patients. This project examines how a Multileaf Collimator system could be built to accurately track and target tumours whilst minimising damage to surrounding tissue, and to construct an functioning system. In this project, we developed a low cost novel solution to minimising Multileaf Collimator systems for radiotherapy. Whilst the system was not wholly constructed, the progress does demonstrate how a miniaturised system can be built at a low cost from retail components. Tracking and object recognition, motor positioning, physical construction of the system, and system communications were all demonstrated effectively. Calculations have been provided for determining an adequate motor and lead screw to drive the system, as well as future work for the project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cancer is one of the leading causes of death, both here in Australia and worldwide. Radiation therapies are one of the primary modalities used to treat cancers. Whilst radiation therapies can be used to destroy cancer cells, it will also damage healthy tissues that are irradiated along the beam path. The organs of the body are also in constant motion due to the beating of the heart, the inhalation and exhalation of the lungs, and organ motility. As such, radiation therapies become a balancing act between effectively irradiating the target volume whilst minimising the radiation dosage to surrounding tissues. One such way of minimising the radiation dose to the surrounding healthy tissue is to use real-time tracking of the target volume to detect the target volume and its boundaries as they move and to shape the radiation beam to minimise the radiation treatment outside of this volume. This has been shown to improve survival rates in clinical trials for various forms of cancers (liver, lung, pancreas) as opposed to non-real-time tracking methods [6]

1.1 Project Objective

This project aims to develop a miniaturised system that can track a target volume in two dimensions and to use a Multi-Leaf Collimator (MLC) to direct the radiation onto the target area to minimise damage to the area surrounding the target volume. Such a system could be used in radiotherapy to treat cancers.
The system will be evaluated in-house with the use of a radiation-sensitive hydrogel which changes colour due to radiation. A model will be formed from the hydrogel and a section of the gel will be marked to distinguish the target volume within the hydrogel. A slight amount of movement will be introduced to move the hydrogel model about and the system will track and irradiate the hydrogel model. The model will then be imaged and analysed to determine the amount radiation received by the model both within the target volume and the surrounding model volume. This will be used to measure the effectiveness of the system in both delivering an adequate amount of radiation to a tumour whilst minimising damage to the surrounding tissue.

Chapter 2

Literature Review

2.1 MLC

Multi-Leaf Collimators (MLC), as shown in Fig. 2.1 are used as a field-shaping technique in radiation oncology to direct irradiation beams onto a tumour. As tumours can vary in size and shape, a MLC can be used to form many different complex shapes of varying sizes. These systems consist of multiple thin plates of material that is able to largely attenuate the transmission of the irradiation source to its destination, effectively shielding an object from the radiation. These sheets, known as the leaves, are independently controlled in one dimension of movement. These leaves can be moved to allow an irradiation source to pass between the leaves to trace out the pattern of complex shapes of varying sizes, as shown in Fig. 2.2.

2.1.1 Movement of the Leaves

As the leaves of the MLC are independently actuated, they can be moved throughout the treatment to attain a new field shape, as may occur as the tumour moves within the body, as the patient moves, if different sections require a higher dosage, or if the system is rotated to irradiate the volume from another angle. When the system does need to actuate the leaves to another position to then irradiate the volume from this new position, there are two different methods which can be employed; Step-And-Shoot (SAS) and the dynamic method (DMLC) [2]. SAS involves turning off the irradiation source whilst the leaves are moved into their new position and then the source is turned back on to irradiate the target volume from this new position. DMLC keeps the irradiation source on as the leaves are repositioned, irradiating the target volume as the leaves are repositioned. DMLC has been found to have better outcomes than SAS in terms of both spatial and intensity resolutions as the leaves are able to better match the desired irradiation profile over the target volume [2].

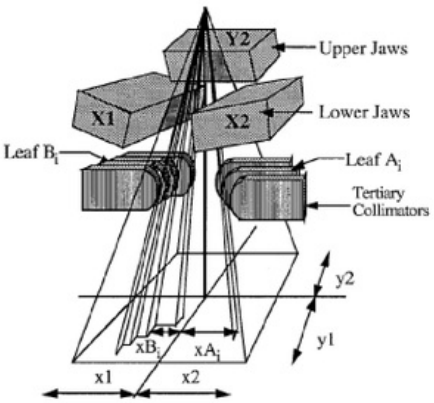

Figure 2.1: Representation of a MLC system [1].

2.1.2 Placement of Leaves

As the leaves of the MLC have a finite width, the MLC is not capable of shaping the irradiation beams to be in continuous contact with the boundary of the target volume. As such, there exists multiple strategies to place the leaves in relation to the boundary of the target volume; out-of-field, in-field, and cross-boundary placement, as shown in Fig. 2.4. Out-of-field placement places the leaves along the outside of the target volume such that no part of the leave will cover over part of the target volume, prioritising the target volume being wholly irradiated over healthy surrounding tissue being irradiated. In-field placement is much the opposite of in-field placement; the leaves are placed so that the minimal amount of the tumour is covered whilst making sure that none of the surrounding healthy tissue is irradiated. It prioritises not irradiating healthy tissue over wholly irradiating the target volume. Cross-boundary placement is seen as a compromise between the two aforementioned strategies, with the exact method of how the leaves are placed being dependent on how the strategy prioritises the trade-off between irradiating all of the target volume and minimising damage to surrounding healthy tissue (e.g. equal areas irradiated about the boundary, bisect each leaf edge with the boundary).

2.2 Image Tracking

Typically, images of the tumour would be obtained through the use of MRI or CT scans. Through these imaging techniques, and the associated image processing, three-dimensional images may be obtained of the structure within a being. These three-dimensional images may then be taken as two-dimensional object perpendicular to the irradiation beam to determine what area the irradiation beam will irradiate.

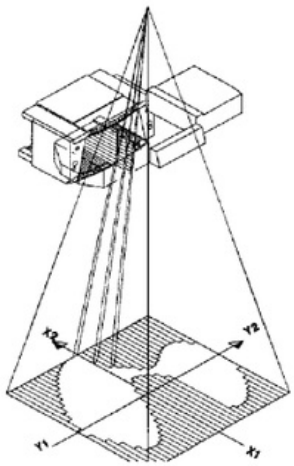

Figure 2.2: MLC system showing a complex shape being irradiated [1].

Image tracking can be broadly split into two groups; region-based tracking and feature-based tracking [4]. Region-based tracking methods, such as mean-shift tracking [3], are not as well suited towards determining the geometry of an object, unlike feature-based tracking, and so they are not used as widely in medical imaging as feature-based tracking methods [4]. Edge detection is one such feature-based tracking method that may be used to define the outline of objects in an image [9] as shown in Fig. 2.5. Sobel edge detection is an algorithm that overlays many discrete differentiation matrices all over the image to determine if there is a large rate of change in the image intensities in a region. If a region does have a very large rate of change in its image intensity, that will be determined as an edge and so objects outlines can be distinguished from a background, provided there is a significant difference in the intensity of the object and the surrounding image. For MRI

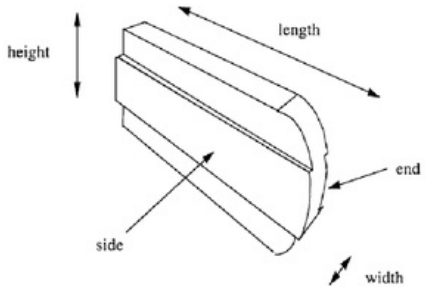

Figure 2.3: Leaf dimensions [1]

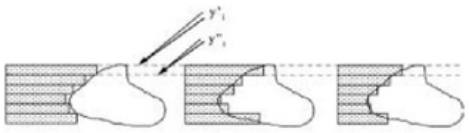

Figure 2.4: Out-of-field, in-field, and cross-boundary leaf positioning [1].

images, the difference between a tumour and the surrounding tissue is sufficient enough that edge detection would be applicable. However, edge detection may detect many different edges in an image and so a human operator would still be required to determine the tumour in the image.

Three volumes about the tumour are classified; the Gross Tumour Volume (GTV), the Clinical Target Volume (CTV), and the Planning Target Volume (PTV) [8]. The GTV is the volume of the detectable tumour. The CTV includes the GTV and is also made of the tissue that surrounds the GTV that includes the volume of subclinical tumour spread. Finally, the PTV includes the GTV and CTV volumes and incorporates a factor of safety into the irradiated volume by irradiating surrounding healthy tissue about the tumour to ensure that the entire tumour has been irradiated. Through the use of MLCs, the volume of the PTV can be reduced as the MLC is able to closely follow the outline of objects. With the use of motion feedback too, the PTV can be further reduced. Dilation is an image processing technique that can be used to expand the area surrounding an object. This can easily be applied to an identified tumour to expand its boundaries to incorporate the PTV area of an irradiation treatment.

2.3 Leaf Position Detection

The system must be aware of where the leaves are throughout the treatment to ensure that the system is accurately irradiating the target volume correctly. There exist many different ways of identifying the location of the leaves.
Limit switches may be used to 'zero' the position of the leaves to some datum point

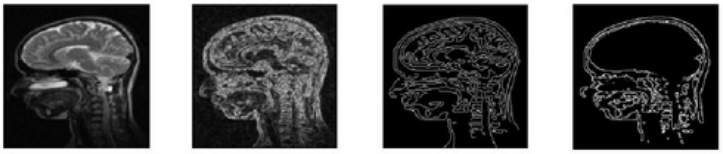

Figure 2.5: Differing edge detection of a sagittal plane MRI of a head (original MRI on the left) [7].

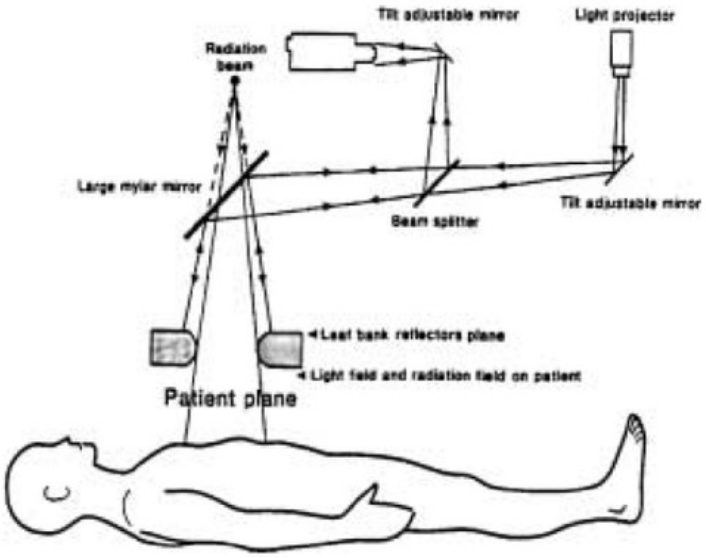

Figure 2.6: Use of a video-optical system to determine leaf positions [1].

from which all positions may then be based off of. Linear encoders may also be used to determine the position of the leaves over the possible range of movement. Video-optical systems may also be used to view the location of all of the leaves and to then work out the position of the leaves [1]. By using a beam splitter and mirror system, the position of the leaves may be determined as shown in Fig. 2.6.

Chapter 3

Experimental Procedures

3.1 Introduction

To evaluate the system, a moving test bed is to be built with a radiation-sensitive hydrogel phantom placed on top. The test bed will be able to move in two-dimensions perpendicular to the irradiation beam. The movement of the bed will be used to simulate the movement of the tumour that may be experienced by the patient's organ motility, heart beat, and breathing. A small amount of random motion will be introduced with a velocity of less than 5mm per second.

Different methods of irradiating the target volume will also be tested for its effectiveness, with combinations of SAS and DMLC radiation triggers, as well as the placement of the leaves about the target volumes boundaries. Varying combinations of target volume velocity, shape, and size may also be tested.

3.2 System Components and Design Overview

The current construction and design of the project consists of the items as listed:

- 1 x Besta PT-500 PSU
- 40 x Sanyo 5V Micro Stepper Motor and Lead Screw
- 40 x DRV8834 Low-Voltage Stepper Motor Driver Carrier
- 1 x Xilinx Spartan-3 XC3S200-5FT256 FPGA
- 1 x Point Grey DragonFly 2 camera
- 1 x Point Grey DragonFly camera
- 1 x USB-serial port connector

ISE Project Navigator has been used to simulate and program the FPGA whilst Matlab has been used to develop the tracking software. The physical designs of the system and the printed parts of the system were produced using Autodesk Inventor.

The system uses an IEEE 1394 Firewire PCI controller card to input a monochrome 1024 x 768 pixel image at 30.00 Hz with a Point Grey Dragonfly 2 camera and the PC's IEEE 1394 Firewire port to control the other camera inputting a monochrome 1024 x 768 pixel image at 15.00 Hz with a Point Grey Dragonfly camera. These images are input into the Matlab software, where they are converted to a binary image before an opening morphological operation is used to remove noise from the image. This image is then used to find the edges of the target volume and determine the position that the leaves of the MLC must be placed. The software then sends serial output from a USB port on the PC to an USB-to-RS-232 adapter which is then connected to the FPGA serial port. The FPGA has been programmed to accept the RS-232 standard and using a 115200/8-N-1 signal: 115200 is the baud rate at which the PC and FPGA have both been programmed to operate at, 8 data bits are sent per package, N for no parity bits, and 1 stop bit is used, giving a total of 10 bits sent per package which includes the one start bit. After the FPGA receives each of the eight data bits, they are sent off to a register that is assigned to a motor. Each motor has an individual register which corresponds to the position to which the motors need to move the leaves. As each eight data bits are received, the FPGA will then send these bits to the next motor in series. The FPGA operates as a Finite State Machine (FSM), sending off each subsequent byte of information to the next motor register in the series. The FPGA can receive a maximum of 144 system commands per second, as shown in (3.1).

$$11520\frac{\text{commands}}{\text{s}} = 115200\frac{\text{baud}}{\text{s}}/10\frac{\text{bit}}{\text{package}} \tag{3.1a}$$

$$5760\frac{\text{motor commands}}{\text{s}} = 11520\frac{\text{commands}}{\text{s}}/2\frac{\text{commands}}{\text{motor commands}} \tag{3.1b}$$

$$144\frac{\text{system commands}}{\text{s}} = 5760\frac{\text{motor commands}}{\text{s}}/40\frac{\text{motors}}{\text{system commands}} \tag{3.1c}$$

The register for each motor is used to compare against a counter value that counts the number of steps sent to each of the stepper motor drivers, this counter tells the current position of the motor. The direction of the stepper motor and the number of steps are then adjusted such that the two registers for each motor then are equal in value. The stepper motor driver takes the step, and direction bit signals from the FPGA, and then drives the stepper motor accordingly.
A camera above the MLC is used to calibrate the original positions of the leaves so that they can all be set to a reference position, or 'zero position'.

The leaves have been designed for a maximum travel of 25mm to cover the size of the aperture through which the target volume will be irradiated. As 20 leaves are required on either side of a 20 mm x 20 mm aperature, 20 1 mm sheets have been designed. Due to the small thicknesses of the leaves and a need to actuate the leaves, protrusions from the rear end of the leaves have been constructed. These protrusions become much thicker to accept a lead screw to move the leaves.

Chapter 4

Construction

The construction of the system consists of multiple discrete components; Matlab programming, FPGA programming, electronics, manufacture of the physical system, and the stepper motors used for the actuation of the leaves. Each of these sections will be discussed separately with the interactions between them also being discussed.

4.1 Matlab Programming

4.1.1 Imaging and Tracking

Many different methods of tracking objects are possible within the two broad groups of region-based tracking and feature-based tracking. Region-based tracking was determined to be ill-suited for this sort of project, as was discussed in Section 2.2. Whilst edge detection may be better suited towards determining objects in medical images such as MRI scans [4], it was determined that for this project edge detection via a method such as using a Sobel operator or other morphological operation may be unnecessary and may require much more complicated post-filtering to determine the target volume. It is believed that because demarcating the target volume from the background in our proposed test hydrogel models can be performed much more easily and with a sharper gradient than a tumour within a medical image of the human body, edge detection could be performed but a much simpler method of thresholding the a greyscale image and then detecting the edges of the object from the boundaries of the image.

Greyscale images are taken from the Dragonfly camera mounted beneath the target volume. These images are then converted to binary images, filtering out intensities in the greyscale image either above, below, or between threshold values. These values are set by the operator of the program and a display of the filtered image is displayed to aid the operator in correctly filtering out everything but the target volume.

4.1.2 Determination of Leaves Positions

As was discussed before, there exists three different strategies for determining leaf positions. The out-of-field and in-field placement strategies are similar to one another and both are based off of my cross-boundary leaf positioning strategy.

The cross-boundary strategy places the midpoint of the leaf on the boundary of the target volume. The strategy starts from the top-left corner of the determined maximum irradiation area, which is the aperture size of the MLC. The midpoint of the first leaf is determined and the program will check points in the binary image at discrete jumps along the travel of the leafs axis of movement. When it first comes across a binary cell that represents the object, the position will be recorded and the system will then move across one leaf distance and start at the top of the maximum irradiation area again to start trying to find the next outline of the object. This process is repeated for all 40 leaves on both sides of the object. In this way, the edges of the object can be found quickly for the relevant points.

The out-of-field and in-field strategies are similar to the above cross-boundary method. However in these cases, the program will first check for the first instance of the object along the path of one of the edges of the leaf instead of the midpoint. Once the edge is found, the program will check the pixel next to the current edge of the leaf. For out-of-field placement, if this pixel and the one before it along the path of leaf travel are both part of the object, the program will update the position of the leaf to this position. This will keep occurring until the program comes across a pixel along the leafs path back towards the starting edge of the screen that is not part of the object. It will then move to the next pixel and continue in this manner. This way the program will find the first pixel along the path of the leaf's travel and ensure that the entirety of the leaf is positioned outside of the leaf.

The in-field strategy is much the same but opposite in that it searches for the maximal position along the leaf's path of travel away from the edge of the image that last detects a pixel that is not part of the object. This way it ensures that the end of the leaf is positioned just inside the objects outline wholly whilst still maximising the parts of the object that will not be covered by the leaves.

4.1.3 Matlab Graphical User Interface (GUI)

This system is not believed to be fully automated; an operator will be required to correctly determine the target volume in the image, which will involve changing many different variables in the image, which will require the visual inspection of the operator. As such the GUI would require a display of the image and relevant information about the system to the operator, both when it is being initialised by the operator and when the system is

operating for the purposes of debugging as well as quality assurance.

4.2 FPGA Programming

4.2.1 FPGA Communication with Matlab Program

The FPGA takes information for the desired locations of the leaves along a serial port from the Matlab program. This information comes into a serial port that exists on the FPGA. The VHDL code must set the baud rate of the FPGAs serial port to the same baud rate that will be used from the Matlab program on the serial port so that they are both communicating at the same rate, otherwise incorrect information will be received. They must also agree on the same communication protocols, in this case 8-N-1, for the same reasons as before.
Depending on the resolution of the system and the number of steps that the stepper motor may be required to take to traverse the 20 mm aperture, more than one byte may need to be sent per motor. As such, the protocol could be changed to increase the amount of information sent in one transmission or registers within the FPGA could be updated after they received the multiple bytes that could be transmitted to increase the distance travelled by the leaves.

4.2.2 FPGA Finite State Machine

The FPGA acts as a FSM; it consists of 40 states each with two registers; one to hold the current position of the motor and one to determine the position the motor should be moved to. Most commands sent in a state will set a new value for the intended position register directly, but there exist a special set of commands that set the value of the current register to a maximum value or zero, go to the last state in the sequence, or return to the first value. These special commands are mostly used to initialise the system by zeroing the positions of the leaves one at a time. The complete list of commands is shown in the table below, Table 4.1.

A separate state exists for each different motor. When most bytes are transmitted to the FPGA, apart from the special commands listed in Table 4.1, the byte will be used to update the register which contains the position to which the current motor in sequence should move to. The state will then move onto the next motor in sequence and await the next command. This automatic movement to the next state for the next motor in sequence will reduce the bandwidth of sending multiple commands when the system is operating to set all of the leaves many times a second, as opposed to having to send information that also codes for which motor the information should be sent to. The special commands mostly exist to initialise the system when it is first turned on. One command is currently used to proceed to the previous motor in sequence which is useful in moving

Current State	Input	Next State	Intended Position
X	11111111	0	M(0, Zero Motor Position)
n	11111110	n - 1	M(n - 1, Unchanged)
n	11111101	n	M(n - 1, Maximum Motor Position)
n	11111100	n	M(n - 1, Zero Motor Position)
n	X	n + 1	M(n, X)
n = 0, 1, ..., 39			M(x, y) = Motor x, intended position y

Table 4.1: Finite State Machine table.

one motor at a time whilst moving the leaves and setting their zero position. Another two commands set the value of the registers of the current state to the maximum value or the minimum value of the leaves. This is also used to set the placement of the leaves into the zero position as the position of the leaf should be zero when it is in its zero position and the value should be set to the maximum when the leaves are in front of the zero position and need to be set to the maximum position value so that the leaves can be moved in the negative direction from this position. Other commands are intended to be added into the project, such as one to switch the UV lamp on and off when needed, and another to enable and disable the motors from the Matlab program.

Switches, LEDs and a seven-segment LED display have been used on the FPGA to ease in the use of debugging as well as enabling and disabling all of the motors. The LED display allows the user to determine which states value is currently being displayed on the LEDs. Another switch determines whether the display and lights are showing the current state and leaf position that is active on the FPGA or will allow the user to move the other switches to select a certain state, which will be shown on the display, with the LEDs showing the value of the register with the current motors position.

4.2.3 FPGA Positioning of Leaves

The FPGA receives the desired positioning of the leaves from the Matlab program and also keeps track of its current positioning of the leaves. The current position of the leaves is kept through the use of a counter that counts how many steps the motor is away from the zeroed position. When the new position information is received, the new position information is compared against the counter of the current position and the current position counter is incremented or decremented to set the current position counter to the same value as the new position information. The current position counter is incremented or decremented at the same frequency as the stepper motor step frequency and each time the counter is incremented or decremented, the Step signal for the motor is triggered high and then set low again. The Direction pin will also be set to the correct direction such that the motor will either spin clockwise or counter-clockwise dependant on whether the

position is being decremented or incremented.

4.2.4 FPGA Communication with Stepper Motor Drivers

There are two individual outputs to each of the stepper motor drivers from the FPGA as well as one shared output to all of the stepper motor drivers, giving a total of 81 pins for all of the 40 stepper motor drivers. The two individual outputs for each driver consist of the Step and Direction pins which is necessary to position each of the motors independently of one another. The shared output is the Enable pin used to enable and disable all of the stepper motor drivers to decrease power usage when none of the motors are in use. This will also decrease the heat in both the drivers as well as the stepper motors as they both draw maximum current when the stepper motors are not rotating.

4.3 Electronics

A PC's power supply is to be used to supply power to the stepper motors and their drivers. This is due to the high amperage that is needed to run stepper motors and that PSUs can supply large amounts of current on their 5 V and 3.3 V rails. Each rail can supply upwards of 25 A on each rail which should supply up to 0.5 A to each motor.

The stepper motor drivers will require three connections to the FPGA; a Step signal, a Direction signal, and an Enable signal. Whilst the drivers do allow microstepping which increases the resolution of the system, microstepping can drastically reduce the torque of the system and decreases the accuracy of the system. As such, most other pins on the stepper motor drivers will not need to be changed during operation and all can be hardwired to predetermined voltages as they will not need to change and will reduce the total number of I/O pins required on the FPGA.

4.4 Manufacture of the Physical System

4.4.1 Construction of Leaves

Multiple different configurations and sizes of leaves would likely have to be manufactured to test their suitability to the project. As such, 3D printing was chosen to construct the leaves due to the speed at which prototypes could be manufactured and then tested with incremental changes being performed on each subsequent design until a suitable design could be found. 3D printing was likely to result in a dimensional accuracy that would likely be unsuitable for this project so further processing of the leaves would be required

in the form of filing and abrasion.

4.4.2 Construction of Camera Housings, UV Lamp Mount, and MLC Housings

These sections of the project were designed around other critical components of the system, such as the cameras, the UV lamp, and the width and length of the leafs and the heads of the leaves. The UV lamp mount was 3D printed to interface with the three locking channels at the base of the UV lamp and to then interface with the topmost camera mount. The interface with the topmost camera mount allows the UV lamp to be removed when the topmost camera is placed into the mount to initialise the positions of the leaves.

The camera mounts hold the cameras with a very tight clearance fit so that the cameras are less likely to be knocked out of position whilst the system is running. One camera mount is placed beneath the test bed and the MLC system, with the other camera mount held above the MLC. The bottom mount is only used to hold the bottom camera which will be facing up at the target volume on the test bed. The bottom camera will be used for tracking the target volume whilst the system is operating and can also be used to initialise the positions of the leaves. The top camera mount will be able to hold both the UV lamp and a top camera. It will be used to both irradiate the target volume when the system is operating and to also use the camera to initialise the positions of the leaves and to record the positions of the leaves when the system is tracking an object.

The MLC housing has been constructed in five separate parts: One central section to support the leaves and hold them in position on either side of the central aperture, two end sections which are able to accommodate the wider area needed for the heads of the leaves as they are moved during the operation of the system, and another two motor housings, one on either end, to hold the 40 motors. All of the parts have either dowel pins, holes, or both that allow each of the parts to join together with a light interference fit. The five assembled sections of the MLC are shown in Fig.4.1.

The motor housings are each able to hold 20 motors with 10 motors mounted on the front and 10 on the back. The rear mounted motors have holes drilled through the motor housing to allow the lead screws to be able to pass through the motor housing, decreasing the necessary height of the system to accommodate all of the motors. One of the motor housings is shown in Fig.4.2 and Fig.4.3.

4.4.3 Construction of the Moving Test Bed

The test bed was to support the target volume above the bottom camera and beneath the aperture of the MLC that the UV light would pass through. The test bed was to be

Figure 4.1: The five assembled sections of the MLC housing with the leaves also in place.

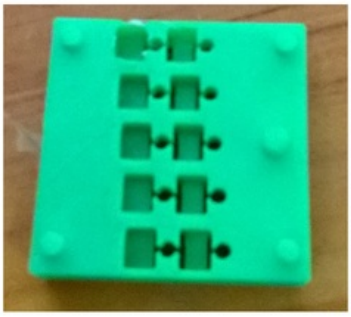

Figure 4.2: Front of motor housing.

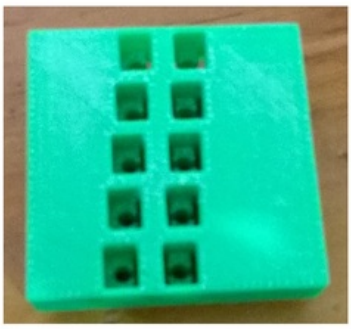

Figure 4.3: Rear of motor housing.

able to move independently in two dimensions, both perpendicular to the axis that the cameras are facing the target volume.

The test bed was constructed with two perpendicular stepper motors driving lead screws that push the bed back and forth in both of the two dimensions. A perspex plate is attached to the topmost lead screw onto which the target volume is placed. Transparent perspex was used so that the camera would be able to detect the target volume through the perspex plate from below. A picture of the constructed system is shown in Fig.4.4.

The lead screws used in the test bed are a 4-start, 8 mm lead, 8 mm diameter lead screw. The motors are NEMA 17 stepper motors and are capable of running step frequencies of 200 Hz with 200 steps per revolution. This allows for a maximum velocity of the bed along one of the axes at 8 mm/s. A maximal velocity of the bed would be when

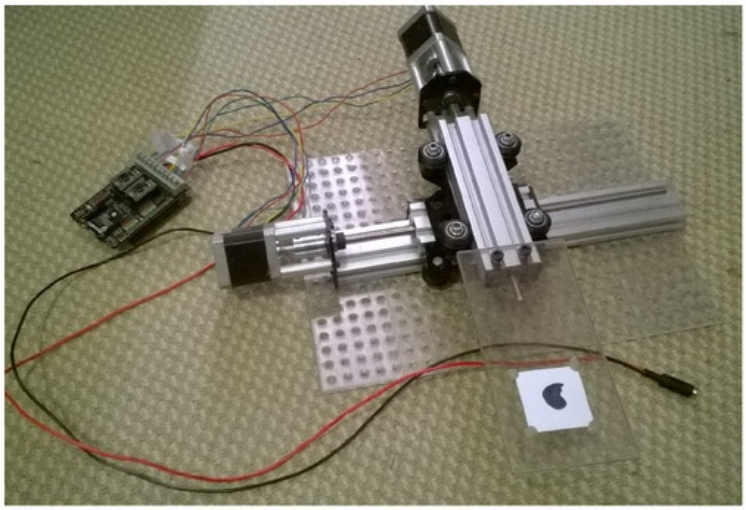

Figure 4.4: Test bed for movement of the target volume.

both motors are running at their maximum frequency and this would give a velocity of over 11.3 mm/s at 45 degrees between the axes of both lead screws. These velocities were determined as adequate for this project as they met the earlier established criterion of being able to travel up to 5 mm/s. These velocities should be sufficient to simulate the velocities of moving organs within the human body at a smaller scale.

4.5 Stepper Motors

Due to the size of the MLC, the motors used to drive the leaves had to be contained in a very small package. Stepper motors and DC motors with rotary encoders were both considered for use in this project. The biggest factors to consider in selecting a motor came down to size, torque/velocity profile, and cost. Unfortunately no motors that were come across in the design and planning stages of this project could meet all three of the requirements. Furthermore, there exist very few suppliers of miniature motors who provide datasheets for their products in the price range required.

The stepper motors that were selected for the project were selected due to their size, cost, and the ability to operate them as an open-loop system by recording how many step signals had been sent to the motors to track their position. No datasheet was provided or could be found. The motors appeared to be ones that would be used in DVD drives to move the head of the DVDs laser. A few details about the motor were available, such as the dimensions of the motor and the attached lead screw, and the rated voltage and

current, so other design characteristics of the project, such as mounts for the motors and the electronics to drive the motors, could be determined. However, when the motors did arrive, it was discovered that the dimensions of the motor and lead screw were significantly different from those advertised, but the design could be changed accordingly to accommodate the differences.

Lead screws were selected due to the small cross-sectional area they present to the end of the leaf, and the mechanical advantage the lead screw would provide to amplify the force available to move the leaves. The equations to move a square-threaded lead screw against a force and in the same direction as an applied force is shown in (4.1a) and (4.1b), respectively.

$$T_{raise} = \frac{Fd_m}{2}\left(\frac{l + \pi\mu d_m}{\pi d_m - \mu l}\right) \tag{4.1a}$$

$$T_{lower} = \frac{Fd_m}{2}\left(\frac{\pi\mu d_m - l}{\pi d_m + \mu l}\right) \tag{4.1b}$$

$$T_{raise} = \frac{Fd_m}{2}\left(\frac{l + \pi\mu d_m \sec\alpha}{\pi d_m - \mu l \sec\alpha}\right) \tag{4.1c}$$

$$T_{lower} = \frac{Fd_m}{2}\left(\frac{\pi\mu d_m \sec\alpha - l}{\pi d_m + \mu l \sec\alpha}\right) \tag{4.1d}$$

T = torque required for the motor to move the lead screw
F = load applied to the lead screw along its axis
d_m = mean diameter of the lead screw
μ = coefficient of friction between the lead screw and the nut
λ = lead of the screw thread
α = half of the thread angle

Equations (4.1c) and (4.1d) are used to determine the torque required to turn a lead screw as above but also taking into account the thread angle of threads, such as an ISO metric thread or and ACME thread.

Chapter 5

Results

5.1 Matlab Programming

The FPGA and Matlab software are able to communicate over the serial port at 115200 bits per second. Higher baud rates are capable in the Matlab software and the FPGA of up to 256000 bits per second but the PC that was used for this project was not capable of running the serial port at these baud rates.

The frame rate of the Matlab program was able to operate at around 15 Hz. This was the maximum frame rate of the camera and so this seems to be the limiting factor of the operating frequency of the tracking and object recognition code currently.

The tracking and image detection program was tested on many different images in varied scenarios before it was used for real-time imaging. Varied object sizes, positions, orientations, and multiple objects were all some of the different tests performed on the program. An example of one of the tested images, as well as the overlay to demonstrate the positioning of the leaves, has been shown in Fig.5.1. These images were visually inspected to check that the software was correctly identifying the correct placement of the leaves on the objects outline. They were also inspected to check that leaves that were not placed on the objects outline, such as in an area where there was no outline in the path of the leaves, would be placed at the midpoint between the nearest adjacent leaves on the outline of the object such that these leaves would be quick to respond to the movement of the tracked object were it to move into the path of the adjacent leaves. An example of this scenario can be seen in Fig.5.2.

A GUI has been constructed for users of the MLC system. A snapshot of the GUI has been included, as shown in Fig.5.3, which shows an irregularly shaped target volume that is being tracked on the left part of the GUI. The outermost lines drawn about the object which form a square, make up the area of the aperture of the MLC. The other lines about the volume show the leaf positioning of the MLC about the tumour. The two leftmost leaves can be seen to be placed end-to-end as there is no target volume in the path of

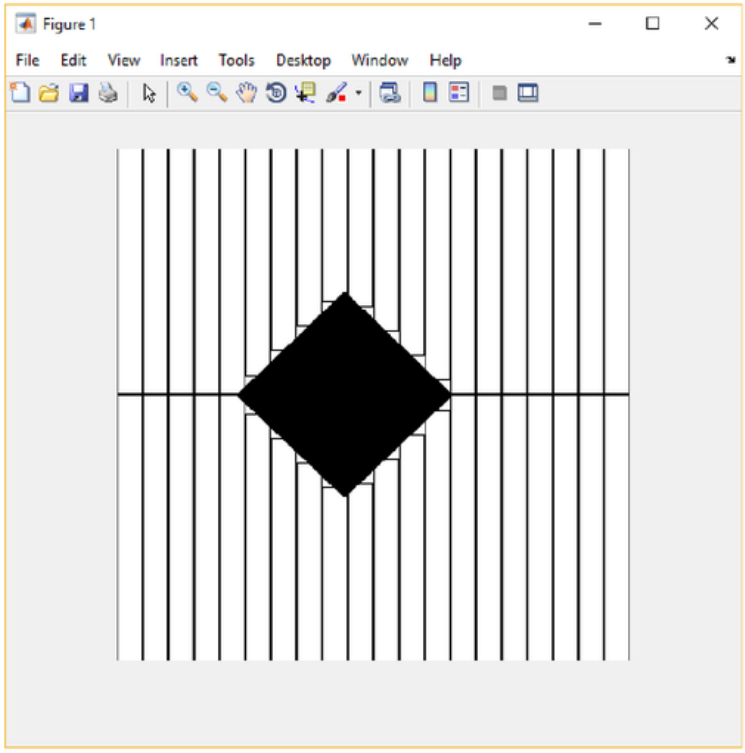

Figure 5.1: Detection of the edge of a diamond using the tracking and object detection program.

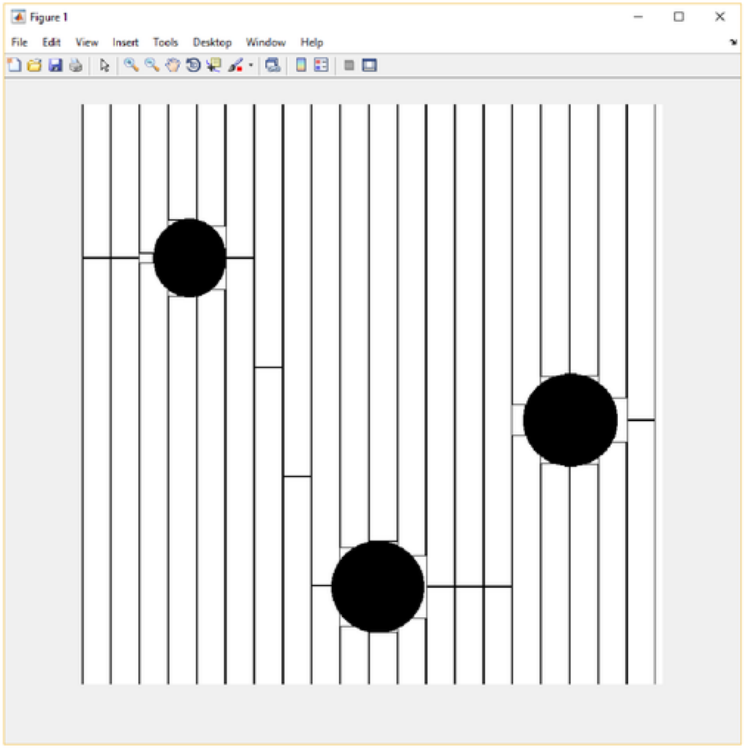

Figure 5.2: Multiple distinct objects demonstrating the positioning of adjacent leaves with no objects in some leaves paths.

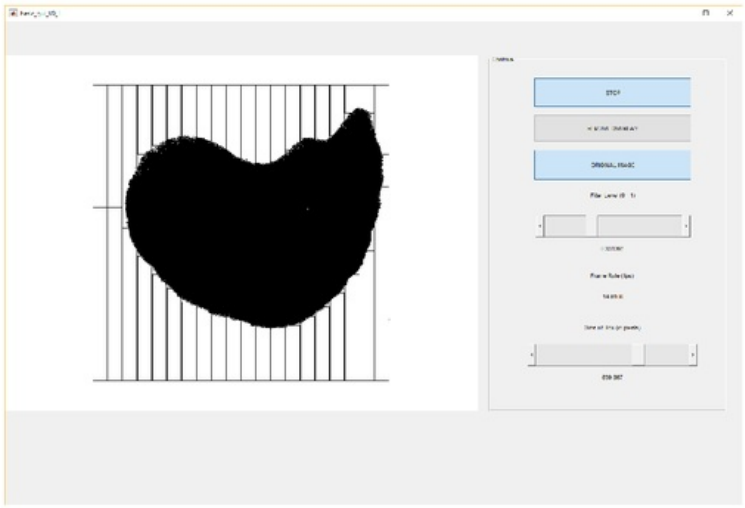

Figure 5.3: GUI layout and tracked object.

these opposing leaves. They are then placed between the next adjacent leaves that do have part of the target volume in their path of travel.

The controls on the rightmost edge of the screen affect multiple parts of the tracking and image recognition variables. The topmost button starts and stops image acquisition from the camera. The second button from the top turns on and off the overlay of the leaves and aperture sizing on the image. The third button switches between the filtered image used for tracking the target volume, as shown in Fig.5.3, and the original greyscale image captured by the camera which is used primarily to align the camera with the aperture of the MLC, as shown in Fig.5.4.

5.2 FPGA Programming

The FPGA was tested on the stepper motors and was able to drive them at up to 200 steps per second. They were also able to drive the motors to the correct number of steps. This was measured by both running the motors with no load at a low frequency (<4 Hz) and counting how many steps the motor made by touching the motor, and by running it at high frequencies (200 Hz) to one full revolution multiple times with a delay of 1 second between the end of one revolution and the start of another. These would be run 100 times and at each 10 revolutions the position of the motor was measured from the starting position. The motor would be set to run in two different scenarios; one scenario where the motor was run continuously in the one direction for each revolution and the

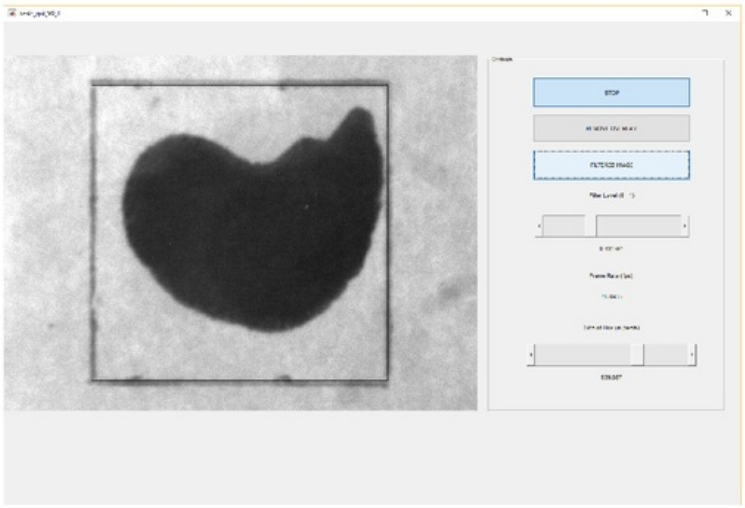

Figure 5.4: GUI layout and unfiltered object used to orientate the camera to the MLC aperture.

other scenario had the motor alternating the direction after every revolution. Neither method resulted in any recorded slip in the motor from its intended position.

Serial commands were also sent to the FPGA and tested for any errors in sending the correct information. The system was tested by sending both individual commands and multiple commands at once and reading the values of the registers off of the FPGA seven-segment LED display. The individual commands were run 80 times and no errors were detected. The multiple commands were sent in batches of 40 at a time and this was performed 10 times. No errors were detected. It should be noted that these tests could have been far more automated or rigorous, but it was felt at the time that the system seemed to be performing adequately to proceed with other work.

5.3 Electronics

All 40 of the stepper motor drivers were tested to ensure that they were operating correctly in their full-stepping mode. All were tested to ensure that they were able to drive the motor both clockwise and anticlockwise. The individual stepper motor drivers were not tested to see if they were all capable of driving the motors the correct number of steps but it was believed that this was not necessary.

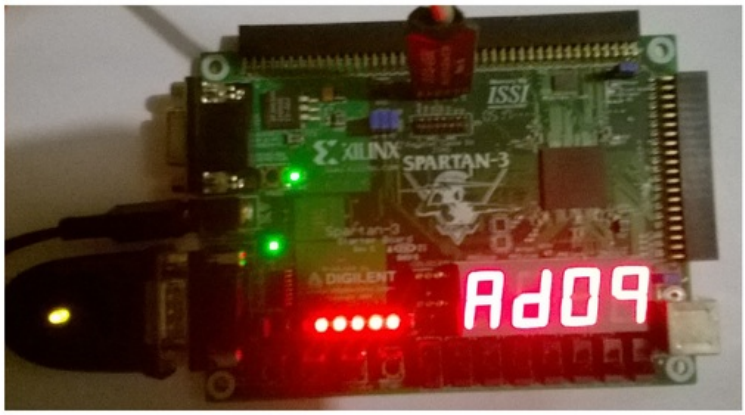

Figure 5.5: FPGA in its 9th motor state with the motors disabled and a motor position of 62.

5.4 Manufacture of the Physical System

The leaves were constructed using 3D printing of ABS plastic. This allowed for rapid prototyping of the leaves to test different designs, printing orientations, and low cost. The parts would be printed in large batches and then filing would be used to take the rough print to an accurately dimensioned leaf with a smooth surface finish. Measurements were taken with a micrometer to ensure that the thicknesses of each leaf were <0.95 mm over 14 points on the leaf.

Printing of the leaves resulted in poor dimensional accuracy and a low quality of surface finish. It was necessary to file the leaves to the correct dimensions, and in doing so, giving the leaves a superior surface finish. The heads were tapped correctly with very little friction and the backlash was too small to be measured accurately but was smaller than 0.5 mm of travel.

The leaves' heads also had to be drilled and tapped to interface with the motors lead screw. Unfortunately the lead screw of the motors was not a standard thread and, due to the 2.5 mm major diameter of the thread being so small, it was seen as too complex to manufacture a tap for the thread. However, taps were constructed from extra lead screws that were from unused motors. The end of the tap was grinded to produce two cutting edges. Through repeated uses of these constructed taps, the head of the leaves would eventually interface with the lead screw with a small amount of friction and a small amount of backlash (<0.5 mm).

Half of all of the leaves' heads had to be of different lengths to the other half due to the offset mounting of the motors within the motor housing. This was to allow the

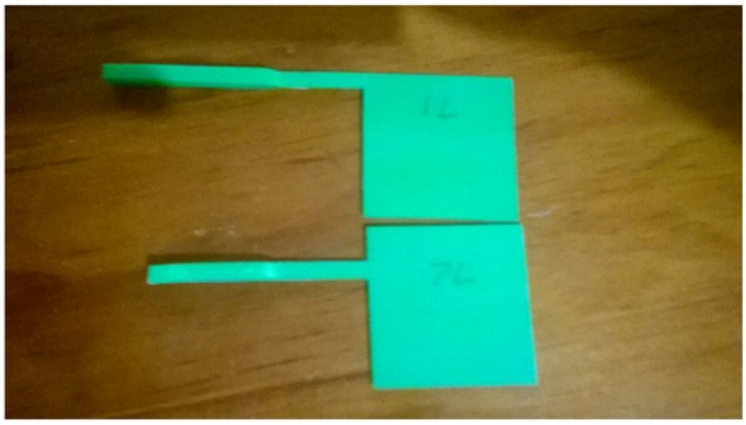

Figure 5.6: Two leaves: a long leaf (top) and a short leaf (bottom) showing the necks and the different sized heads.

Leaf No.	1	2	3	4	5	6	7	8	9	10
Left	1.9	1.6	0.9	1	0.4	0.4	1	0.8	1.5	1.1
Right	0.7	1.2	N/A	0.9	2.2	1.4	1.4	1.3	N/A	N/A
Leaf No.	11	12	13	14	15	16	17	18	19	20
Left	0.9	1.2	1.3	0.4	1.3	0.6	0.8	0.3	0.7	0.6
Right	N/A	1	2.2	N/A	N/A	N/A	2	2.3	2	1.9

Table 5.1: Leaf Force Measurements.

motors that were mounted in the back of the motor housing to still have the entire length of travel needed to move the leaf across the length of the aperture. A photo of both a short and long leaf can be seen in Fig.5.6.

Measurements were taken of the average force needed to move each of the leaves within the MLC housing. The measurements were taken with a 5 N spring scale. The results of the measurements are shown in Table 5.1. Some of the leaves necks were broken at the time of recording and so the leaves and their adjacent leaves were unable to be measured.

From these results, it can be seen that the left side leaves require less force to move each leaf than the right side. This is likely due to the difference in the average thickness of leaves on either side as well as the size of the housing on either side. Whilst all leaves were filed until they were less than 0.95 mm, the left side was machined down to less than 0.90 mm on each leaf to test a thinner leaf and the corresponding force required. The left leaves are still able to block all of the light through the aperture between each leaf whilst requiring less force to move each leaf and so 0.90 mm should be the thickness that each leaf should be machined to.

5.5 Stepper Motors

The stepper motors used for driving the leaves could operate up to 200 Hz. With 20 steps per revolution and a lead on the lead screws of 1.6 mm, the motors would be capable of driving the leaves at a maximal velocity of 16 mm/s which is well above the criterion of 5 mm/s that was proposed at the beginning of the project.

The torque calculated to drive the leaves using the motors lead screws was calculated as follows. A force of 1.5 N to drive the leaves was assumed as most of the values of force required to move the leaves, as shown in Table 5.1, are beneath this value, and it is believed that further filing of the leaves would be able to further reduce this value. The mean diameter of the lead screws was determined by measuring the lead screw with a profile projector. The major diameter was measured at 2.5 mm and a minor diameter of 1.6 mm, to give a mean diameter of 2.05 mm. The lead of the screw thread was 1.6 mm. A low value coefficient of friction was assumed for the system of 0.1. Equation (4.1a) was used due to the square threads of the lead screw, giving a value of 0.549 mN.m. This value is larger than torque values of similarly-sized motors at a much higher price point, such as Faulhaber's FDM0620 stepper motors which can only supply up to 0.25 mN.m [5].

The motors for the actuation of the leaves were eventually found to be insufficient to operate the system. The operating voltages and currents of the stepper motors were too weak to even drive the motors to thread a single unsupported leaf, let alone when the leaves were packed next to each other and would require more force to move due to the friction between the sheets. Driving the motors at close to two times the rated voltage and current (9.5 V and 0.65 A, respectively), produced a bit more torque and some motors were able to actuate a single leaf. However, these results were sporadic and the heat generated would either blow the motor within a few seconds or melt the surrounding motor housing, as can be seen in the top-leftmost motor position in Fig.4.2. Also, likely due to removing the housing about the motors which worked to hold the lead screw and attached rotor magnet in place within the motor, some motors would not rotate properly and the rotor magnets would take a few steps before hitting the stator magnets and seeming to find a spot from which the friction and irregular magnetic field strength, due to their offset position, would cause the motor to seize and stop rotating. For all of these reasons, it was decided that these motors would not work for this system and that other motors would have to be found. Two motors are shown in Fig.5.7, the top one shows the housing that supports the lead screw, and the bottom one shows a motor after it had the housing removed and wires soldered so that it could be used in the project.

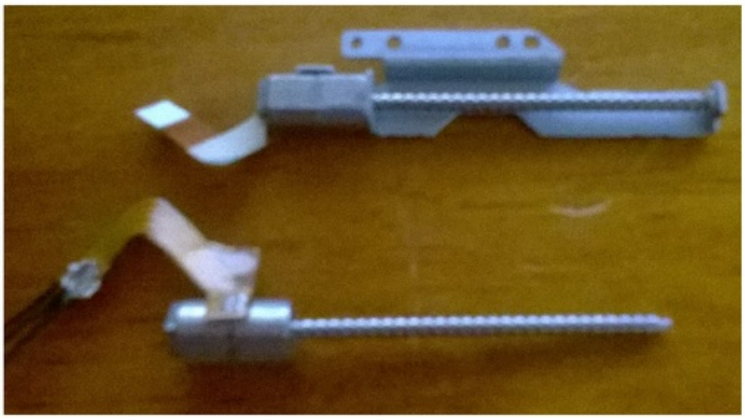

Figure 5.7: A motor before being being modified for use in the project (above) and one after having its housing removed and wiring soldered on (below).

Chapter 6

Discussion

Whilst the work on this project was not completed, it is believed that the project is possible if adequate motors are sourced for actuating the leaves of the MLC. The software is able to track the outline of an object based upon the intensity of the object compared to the rest of the image. It is able to then send this information via a serial port to the FPGA which then moves the motors from their current location to the location sent from the tracking and object detection program. These signals are sent to the stepper motor drivers which then are able to move a set of 40 stepper motors up to 144 times per second. It is believed that this forms the basis that this project is feasible and work should be continued into it.

Originally it was planned to use small photointerrupters to detect the positions of the leaves to set a zero position of the leaves. Unfortunately it was found that, even though the dimensions of the photointerrupters only measure 4.2 mm x 4.2 mm x 5.2 mm, they were still too large to be incorporated into the current design unless the total length of the system was extended by another 200 mm, which would make it much harder to assemble, manufacture, and to try not to break the fragile leaves. Using the cameras to detect the positions of the leaves and then set the positions of the leaves when the system is initialised was then decided upon. It is unknown at this time whether using the cameras to set the positions of the leaves will give accurate results.

It was found that the current motors were unable to move the leaves due to their low torque and their high lead. Looking at a similarly sized motor from Faulhaber and a M2.5 x 0.25 mm lead screw and using (4.1c), with a coefficient of friction of 0.1, half thread angle of thirty degrees, mean diameter of 2.375 mm, force of 1.5 N, and a lead of 0.25 mm, gives a torque value of 0.266 mN.m [5]. However, these motors can have gear heads attached to the motor to increase the mechanical advantage of the motor up to a torque value that is able to drive the system and with a high enough velocity ratio to still actuate the leaves at 5 mm/s.

Whilst the high temperatures experienced by the stepper motors controlling the leaves

was likely mostly attributable due to running the motors close to twice the rated voltage and current, it has raised the problem of heat dissipation from the motors. Fans are likely needed in the future designs of the MLC to actively cool the motors. Due to the close density of the motors, dissipating the heat with active air flow may not be effective. Using the SAS method to target the target volume by switching the UV lamp and the motors off and on for periods of time may be another method to better dissipate and avoid melting the surrounding motor housing. Finally, the motor housing could be built of a better heat-conducting material, most likely metal, to act as a large heat sink which could be made to better dissipate the heat over the entire surface of the motor mount and have a built in fin for increased heat exchange.

Chapter 7

Conclusions

Unfortunately the project was unable to be finished in time to meet the requirements of the system. It is believed that the majority of work on the system has been performed and that the largest barrier to progress in completing the report is the selection of appropriate motors.

However, the work done into producing a miniaturised theragnostic irradiator with motion feedback is promising; tracking software was made that was able to accurately detect an object based upon its intensity and track the object as it moved in real-time, motor commands could be sent to 40 motors over 100 times per second, and position the motors from their current position to their intended position, and the majority of the physical parts of the MLC have been constructed at a scale much smaller than currently existing MLC systems [1].

Whilst these results can not be expected to be directly comparable to the outcomes of existing MLC systems which would outperform this system in many ways, the price and ease of manufacture of the system produced would be suited to researchers involved in projects involving targeted radiation therapies at a smaller scale where they may not have access to very expensive medical-grade MLC systems.

Chapter 8

Future Work

Unfortunately the project was unable to be finished in time to meet the requirements of the system. It is believed that the majority of work on the system has been performed and that the largest barrier to progress in completing the report is the selection of appropriate motors.

The majority of the work on the FPGA has been done. Testing of correct timing of signals has been performed and work adequately. However, the current FPGA that has been used in the project has had some problems with not all of the pins working correctly. This may be due to broken pins on the board. However, it is currently believed that a second FPGA could be added to the project to split the number of drivers that each FPGA is controlling into two. This would effectively double the number of pins that are available in the project and is not believed to introduce any delays or errors into the system.

Currently the stepper motor drivers are capable of operating at voltages between 2.5 V - 10.8 V and can supply up to 1.5 A per phase continuously (maybe put in reference here). These seem sufficient to drive motors in this size range, which typically use quite low voltages, and so are not likely to need to be replaced in the future. A PCB (Printed Circuit Board) should also be manufactured for the project. To wire up the project currently takes up hundreds of individual wires and a very large surface area. This is problematic when wires come loose from their sockets and to correctly wire everything. A PCB would help to reduce a lot of these problems and give a much more portable design than currently afforded by wiring up the system with multiple large breadboards.

More work may need to be carried out for the Matlab tracking and object detection program. Currently the program does not detect the outline of the aperture of the MLC and align the leaf positioning region to it and instead requires the operator of the system to align the aperture of the MLC to the leaf positioning region shown on the GUI. This is much more likely to introduce errors into the system when it does undergo testing. As such, it is proposed that future work should work to position reference markers about the

aperture such that the program will be able to detect these markers and correctly orientate and size the leaf positioning region in the software to match the size and orientation of the aperture.

More work should also be performed in trying to optimise the code such that the run-time of each cycle of the tracking and object detection code can be decreased, increasing the frame-rate. Currently the object detection code decreases as the area of the target volume is decreased. Another method may need to be introduced that can decrease the run-time with small target volumes whilst keeping the system able to detect the outline of the smaller objects reliably.

Further electronics and programming needs to be incorporated for controlling the UV lamp. Currently no work was performed in this area as the UV lamp did not see use in the testing of the project. Whilst the lamp does use mains power, it could be switched on and off using a SSR (Solid-State Relay) controlled by the FPGA and connected to the mains.

This project is but one part of a larger project into constructing a miniaturised theragnostic irradiator system and so future work should involve incorporating this part of the system into the rest of the system.

The leaves and the effect of the penumbra upon the outcome of the irradiation also need to be investigated. Different leaf end shapes may need to be investigated to determine which end shape of the leaves will give a better result in irradiating the target volume. Whilst the system can not be used to currently shape the beam onto a moving system, the leaves could be moved by hand to form a shape and then a phantom hydrogel irradiated whilst stationary.

Chapter 9

Abbreviations

CTV	Clinical Target Volume
DMLC	Dynamic Method
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
GTV	Gross Tumour Volume
I/O	Input Output
MLC	Multi-Leaf Collimator
PSU	Power Supply Unit
PTV	Planning Target Volume
SAS	Step-And-Shoot
SSR	Solid-State Relay
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

Appendix A

Physical Designs and Objects

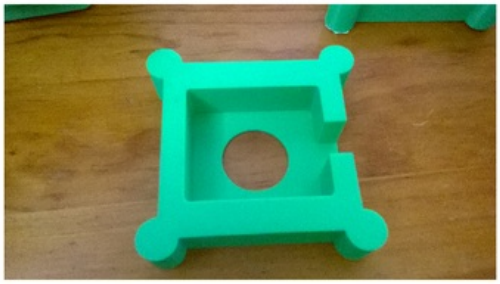

Figure A.1: Top camera mount.

Figure A.2: UV lamp mount.

Figure A.3: Bottom camera mount with the mounted camera.

Figure A.4: UV lamp.

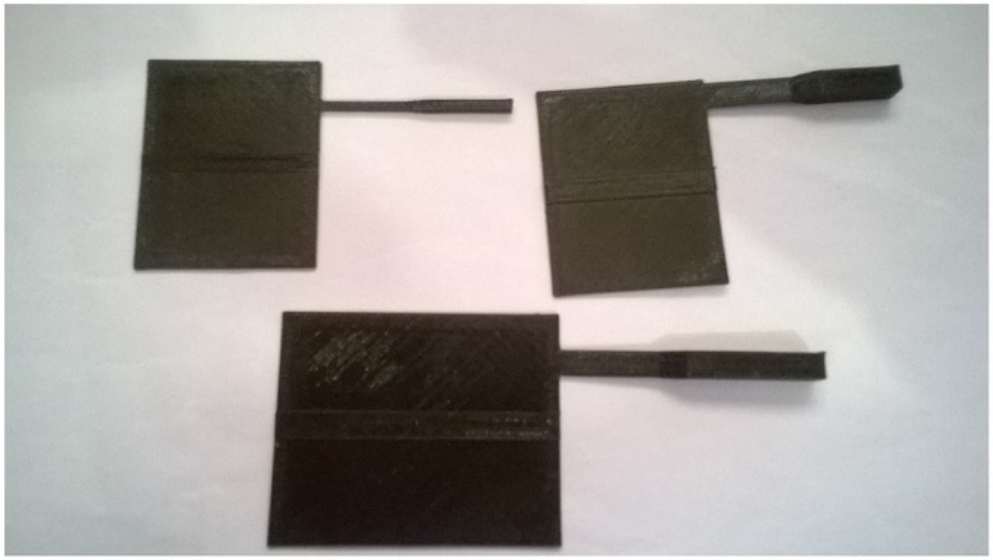

Figure A.5: Early Leaf Prototypes

Figure A.6: 20 mounted motors (some lead screws are loose due to the heat of operation melting the surrounding plastic).

Appendix B

Code

B.1 GUI, and Tracking and Object Recognition Code

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    12:12:29 11/02/2016
-- Design Name:
-- Module Name:    PositionDrive40M - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity PositionDrive40M is
Port ( CLK_50MHZ : in  STD_LOGIC;
SWITCHES : in STD_LOGIC_VECTOR (7 downto 0);
RXD : in  STD_LOGIC;
TXD : out  STD_LOGIC;
STEP : out  STD_LOGIC_VECTOR (39 downto 0);
DIR : out  STD_LOGIC_VECTOR (39 downto 0);
ENABLE : out  STD_LOGIC;
```

```
D : out  STD_LOGIC_VECTOR (7 downto 0);
D_ON : out  STD_LOGIC_VECTOR (3 downto 0);
MOTOR_NO_LIGHTS : out  STD_LOGIC_VECTOR (7 downto 0));
end PositionDrive40M;

architecture Behavioral of PositionDrive40M is

constant clk_50MHZ_freq : integer := 50000000;
constant baudrate       : integer := 115200;
-- Must be set to the same value as the Matlab program's value.
constant countlimit_x16 : integer := (clk_50MHZ_freq / 16) / baudrate;
-- Used to sample the serial port 16 times a second to detect new information.
constant step_freq      : integer := 10;
-- Motor steps per second.
constant period : integer := (clk_50MHZ_freq / 2) / step_freq;
-- Period of motor stepping.
constant max_step_pos   : std_logic_vector := "11001000";
constant zero_step_pos  : std_logic_vector := "00000000";

signal clk_period       : std_logic := '0';
signal count_period     : integer := 0;

signal clk_x16  : std_logic := '0';
signal count_x16        : integer := 0;

signal count_16 : integer := 0;

signal motor_light      : integer := 0;

signal in_byte  : std_logic_vector(7 downto 0) := "00000000";

type forty_output is array (39 downto 0) of std_logic_vector(7 downto 0);
signal out_byte : forty_output := ((others => (others => '0')));

signal cur_out_byte     : std_logic_vector(7 downto 0) := "00000000";

type forty_counters is array (39 downto 0) of std_logic_vector(7 downto 0);
signal count_to_step    : forty_counters := ((others => (others => '0')));

signal step_signal      : std_logic_vector(39 downto 0) := (others => '0');

signal dir_signal       : std_logic_vector(39 downto 0) := (others => '0');

signal zero_signal      : std_logic_vector(39 downto 0) := (others => '0');
signal max_signal       : std_logic_vector(39 downto 0) := (others => '0');

signal ready    : std_logic := '0';

type bit_order IS (bit_start, bit_0, bit_1, bit_2, bit_3, bit_4, bit_5, bit_6, bit_7, bit_stop);
signal bit_pos : bit_order;

TYPE byte_order IS (byte_0, byte_1, byte_2, byte_3, byte_4, byte_5, byte_6, byte_7, byte_8, byte_9,
byte_10, byte_11, byte_12, byte_13, byte_14, byte_15, byte_16, byte_17, byte_18, byte_19,
byte_20, byte_21, byte_22, byte_23, byte_24, byte_25, byte_26, byte_27, byte_28, byte_29,
byte_30, byte_31, byte_32, byte_33, byte_34, byte_35, byte_36, byte_37, byte_38, byte_39);
signal byte_pos : byte_order := byte_0;



signal eight    : std_logic_vector(2 downto 0) := "000";

signal delay    : std_logic_vector(7 downto 0) := (others => '0');

signal clk2 : std_logic;
```

```
signal D3       : std_logic_vector(7 downto 0);
signal D2       : std_logic_vector(7 downto 0);
signal D1       : std_logic_vector(7 downto 0);
signal D0       : std_logic_vector(7 downto 0);

signal motor_num        : std_logic_vector(7 downto 0);

signal number   : integer;

signal switch_m : std_logic_vector(7 downto 0) := "00000000";
signal switch_e : std_logic_vector(7 downto 0) := "00000000";

begin
process(CLK_50MHZ)
begin
if CLK_50MHZ = '1' and CLK_50MHZ'event then
if count_period < period then
count_period <= count_period + 1;
clk_period <= '0';
else
count_period <= 0;
clk_period <= '1';
end if;
end if;
end process;

process(clk_period)
begin
if clk_period = '1' and clk_period'event then
for i in 0 to 39 loop
if zero_signal(i) = '1' then
count_to_step(i) <= zero_step_pos;
elsif max_signal(i) = '1' then
count_to_step(i) <= max_step_pos;
elsif count_to_step(i) /= out_byte(i) then
if count_to_step(i) < out_byte(i) and dir_signal(i) = '1' then
dir_signal(i) <= '0';
elsif count_to_step(i) > out_byte(i) and dir_signal(i) = '0' then
dir_signal(i) <= '1';
else
step_signal(i) <= not step_signal(i);
if step_signal(i) = '1' and dir_signal(i) = '0' then
count_to_step(i) <= count_to_step(i) + 1;
elsif step_signal(i) = '1' and dir_signal(i) = '1' then
count_to_step(i) <= count_to_step(i) - 1;
end if;
end if;
end if;
end loop;
end if;
end process;

process(CLK_50MHZ)
begin
if CLK_50MHZ = '1' and CLK_50MHZ'event then
if count_x16 < countlimit_x16 then
count_x16 <= count_x16 + 1;
clk_x16 <= '0';
else
count_x16 <= 0;
clk_x16 <= '1';
end if;
end if;
end process;
```

```
process(ready)
begin
if ready = '0' and ready'event then
zero_signal <= "00000000000000000000000000000000000000000";
max_signal <= "00000000000000000000000000000000000000000";
if cur_out_byte = "11111111" then
motor_light <= 0;
byte_pos <= byte_0;
else
case byte_pos is
when byte_0 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 39;
byte_pos <= byte_39;
when "11111101" =>
max_signal(39) <= '1';
out_byte(39) <= max_step_pos;
when "11111100" =>
zero_signal(39) <= '1';
out_byte(39) <= zero_step_pos;
when others =>
out_byte(0) <= cur_out_byte;
motor_light <= 0;
byte_pos <= byte_1;
end case;
when byte_1 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 0;
byte_pos <= byte_0;
when "11111101" =>
max_signal(0) <= '1';
out_byte(0) <= max_step_pos;
when "11111100" =>
zero_signal(0) <= '1';
out_byte(0) <= zero_step_pos;
when others =>
out_byte(1) <= cur_out_byte;
motor_light <= 1;
byte_pos <= byte_2;
end case;
when byte_2 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 1;
byte_pos <= byte_1;
when "11111101" =>
max_signal(1) <= '1';
out_byte(1) <= max_step_pos;
when "11111100" =>
zero_signal(1) <= '1';
out_byte(1) <= zero_step_pos;
when others =>
out_byte(2) <= cur_out_byte;
motor_light <= 2;
byte_pos <= byte_3;
end case;
when byte_3 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 2;
byte_pos <= byte_2;
when "11111101" =>
max_signal(2) <= '1';
```

```
out_byte(2) <= max_step_pos;
when "11111100" =>
zero_signal(2) <= '1';
out_byte(2) <= zero_step_pos;
when others =>
out_byte(3) <= cur_out_byte;
motor_light <= 3;
byte_pos <= byte_4;
end case;
when byte_4 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 3;
byte_pos <= byte_3;
when "11111101" =>
max_signal(3) <= '1';
out_byte(3) <= max_step_pos;
when "11111100" =>
zero_signal(3) <= '1';
out_byte(3) <= zero_step_pos;
when others =>
out_byte(4) <= cur_out_byte;
motor_light <= 4;
byte_pos <= byte_5;
end case;
when byte_5 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 4;
byte_pos <= byte_4;
when "11111101" =>
max_signal(4) <= '1';
out_byte(4) <= max_step_pos;
when "11111100" =>
zero_signal(4) <= '1';
out_byte(4) <= zero_step_pos;
when others =>
out_byte(5) <= cur_out_byte;
motor_light <= 5;
byte_pos <= byte_6;
end case;
when byte_6 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 5;
byte_pos <= byte_5;
when "11111101" =>
max_signal(5) <= '1';
out_byte(5) <= max_step_pos;
when "11111100" =>
zero_signal(5) <= '1';
out_byte(5) <= zero_step_pos;
when others =>
out_byte(6) <= cur_out_byte;
motor_light <= 6;
byte_pos <= byte_7;
end case;
when byte_7 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 6;
byte_pos <= byte_6;
when "11111101" =>
max_signal(6) <= '1';
out_byte(6) <= max_step_pos;
```

```
when "11111100" =>
zero_signal(6) <= '1';
out_byte(6) <= zero_step_pos;
when others =>
out_byte(7) <= cur_out_byte;
motor_light <= 7;
byte_pos <= byte_8;
end case;
when byte_8 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 7;
byte_pos <= byte_7;
when "11111101" =>
max_signal(7) <= '1';
out_byte(7) <= max_step_pos;
when "11111100" =>
zero_signal(7) <= '1';
out_byte(7) <= zero_step_pos;
when others =>
out_byte(8) <= cur_out_byte;
motor_light <= 8;
byte_pos <= byte_9;
end case;
when byte_9 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 8;
byte_pos <= byte_8;
when "11111101" =>
max_signal(8) <= '1';
out_byte(8) <= max_step_pos;
when "11111100" =>
zero_signal(8) <= '1';
out_byte(8) <= zero_step_pos;
when others =>
out_byte(9) <= cur_out_byte;
motor_light <= 9;
byte_pos <= byte_10;
end case;
when byte_10 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 9;
byte_pos <= byte_9;
when "11111101" =>
max_signal(9) <= '1';
out_byte(9) <= max_step_pos;
when "11111100" =>
zero_signal(9) <= '1';
out_byte(9) <= zero_step_pos;
when others =>
out_byte(10) <= cur_out_byte;
motor_light <= 10;
byte_pos <= byte_11;
end case;
when byte_11 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 10;
byte_pos <= byte_10;
when "11111101" =>
max_signal(10) <= '1';
out_byte(10) <= max_step_pos;
when "11111100" =>
```

```
zero_signal(10) <= '1';
out_byte(10) <= zero_step_pos;
when others =>
out_byte(11) <= cur_out_byte;
motor_light <= 11;
byte_pos <= byte_12;
end case;
when byte_12 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 11;
byte_pos <= byte_11;
when "11111101" =>
max_signal(11) <= '1';
out_byte(11) <= max_step_pos;
when "11111100" =>
zero_signal(11) <= '1';
out_byte(11) <= zero_step_pos;
when others =>
out_byte(12) <= cur_out_byte;
motor_light <= 12;
byte_pos <= byte_13;
end case;
when byte_13 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 12;
byte_pos <= byte_12;
when "11111101" =>
max_signal(12) <= '1';
out_byte(12) <= max_step_pos;
when "11111100" =>
zero_signal(12) <= '1';
out_byte(12) <= zero_step_pos;
when others =>
out_byte(13) <= cur_out_byte;
motor_light <= 13;
byte_pos <= byte_14;
end case;
when byte_14 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 13;
byte_pos <= byte_13;
when "11111101" =>
max_signal(13) <= '1';
out_byte(13) <= max_step_pos;
when "11111100" =>
zero_signal(13) <= '1';
out_byte(13) <= zero_step_pos;
when others =>
out_byte(14) <= cur_out_byte;
motor_light <= 14;
byte_pos <= byte_15;
end case;
when byte_15 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 14;
byte_pos <= byte_14;
when "11111101" =>
max_signal(14) <= '1';
out_byte(14) <= max_step_pos;
when "11111100" =>
zero_signal(14) <= '1';
```

```
out_byte(14) <= zero_step_pos;
when others =>
out_byte(15) <= cur_out_byte;
motor_light <= 15;
byte_pos <= byte_16;
end case;
when byte_16 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 15;
byte_pos <= byte_15;
when "11111101" =>
max_signal(15) <= '1';
out_byte(15) <= max_step_pos;
when "11111100" =>
zero_signal(15) <= '1';
out_byte(15) <= zero_step_pos;
when others =>
out_byte(16) <= cur_out_byte;
motor_light <= 16;
byte_pos <= byte_17;
end case;
when byte_17 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 16;
byte_pos <= byte_16;
when "11111101" =>
max_signal(16) <= '1';
out_byte(16) <= max_step_pos;
when "11111100" =>
zero_signal(16) <= '1';
out_byte(16) <= zero_step_pos;
when others =>
out_byte(17) <= cur_out_byte;
motor_light <= 17;
byte_pos <= byte_18;
end case;
when byte_18 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 17;
byte_pos <= byte_17;
when "11111101" =>
max_signal(17) <= '1';
out_byte(17) <= max_step_pos;
when "11111100" =>
zero_signal(17) <= '1';
out_byte(17) <= zero_step_pos;
when others =>
out_byte(18) <= cur_out_byte;
motor_light <= 18;
byte_pos <= byte_19;
end case;
when byte_19 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 18;
byte_pos <= byte_18;
when "11111101" =>
max_signal(18) <= '1';
out_byte(18) <= max_step_pos;
when "11111100" =>
zero_signal(18) <= '1';
out_byte(18) <= zero_step_pos;
```

```
when others =>
out_byte(19) <= cur_out_byte;
motor_light <= 19;
byte_pos <= byte_20;
end case;
when byte_20 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 19;
byte_pos <= byte_19;
when "11111101" =>
max_signal(19) <= '1';
out_byte(19) <= max_step_pos;
when "11111100" =>
zero_signal(19) <= '1';
out_byte(19) <= zero_step_pos;
when others =>
out_byte(20) <= cur_out_byte;
motor_light <= 20;
byte_pos <= byte_21;
end case;
when byte_21 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 20;
byte_pos <= byte_20;
when "11111101" =>
max_signal(20) <= '1';
out_byte(20) <= max_step_pos;
when "11111100" =>
zero_signal(20) <= '1';
out_byte(20) <= zero_step_pos;
when others =>
out_byte(21) <= cur_out_byte;
motor_light <= 21;
byte_pos <= byte_22;
end case;
when byte_22 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 21;
byte_pos <= byte_21;
when "11111101" =>
max_signal(21) <= '1';
out_byte(21) <= max_step_pos;
when "11111100" =>
zero_signal(21) <= '1';
out_byte(21) <= zero_step_pos;
when others =>
out_byte(22) <= cur_out_byte;
motor_light <= 22;
byte_pos <= byte_23;
end case;
when byte_23 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 22;
byte_pos <= byte_22;
when "11111101" =>
max_signal(22) <= '1';
out_byte(22) <= max_step_pos;
when "11111100" =>
zero_signal(22) <= '1';
out_byte(22) <= zero_step_pos;
when others =>
```

```
out_byte(23) <= cur_out_byte;
motor_light <= 23;
byte_pos <= byte_24;
end case;
when byte_24 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 23;
byte_pos <= byte_23;
when "11111101" =>
max_signal(23) <= '1';
out_byte(23) <= max_step_pos;
when "11111100" =>
zero_signal(23) <= '1';
out_byte(23) <= zero_step_pos;
when others =>
out_byte(24) <= cur_out_byte;
motor_light <= 24;
byte_pos <= byte_25;
end case;
when byte_25 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 24;
byte_pos <= byte_24;
when "11111101" =>
max_signal(24) <= '1';
out_byte(24) <= max_step_pos;
when "11111100" =>
zero_signal(24) <= '1';
out_byte(24) <= zero_step_pos;
when others =>
out_byte(25) <= cur_out_byte;
motor_light <= 25;
byte_pos <= byte_26;
end case;
when byte_26 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 25;
byte_pos <= byte_25;
when "11111101" =>
max_signal(25) <= '1';
out_byte(25) <= max_step_pos;
when "11111100" =>
zero_signal(25) <= '1';
out_byte(25) <= zero_step_pos;
when others =>
out_byte(26) <= cur_out_byte;
motor_light <= 26;
byte_pos <= byte_27;
end case;
when byte_27 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 26;
byte_pos <= byte_26;
when "11111101" =>
max_signal(26) <= '1';
out_byte(26) <= max_step_pos;
when "11111100" =>
zero_signal(26) <= '1';
out_byte(26) <= zero_step_pos;
when others =>
out_byte(27) <= cur_out_byte;
```

```
motor_light <= 27;
byte_pos <= byte_28;
end case;
when byte_28 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 27;
byte_pos <= byte_27;
when "11111101" =>
max_signal(27) <= '1';
out_byte(27) <= max_step_pos;
when "11111100" =>
zero_signal(27) <= '1';
out_byte(27) <= zero_step_pos;
when others =>
out_byte(28) <= cur_out_byte;
motor_light <= 28;
byte_pos <= byte_29;
end case;
when byte_29 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 28;
byte_pos <= byte_28;
when "11111101" =>
max_signal(28) <= '1';
out_byte(28) <= max_step_pos;
when "11111100" =>
zero_signal(28) <= '1';
out_byte(28) <= zero_step_pos;
when others =>
out_byte(29) <= cur_out_byte;
motor_light <= 29;
byte_pos <= byte_30;
end case;
when byte_30 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 29;
byte_pos <= byte_29;
when "11111101" =>
max_signal(29) <= '1';
out_byte(29) <= max_step_pos;
when "11111100" =>
zero_signal(29) <= '1';
out_byte(29) <= zero_step_pos;
when others =>
out_byte(30) <= cur_out_byte;
motor_light <= 30;
byte_pos <= byte_31;
end case;
when byte_31 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 30;
byte_pos <= byte_30;
when "11111101" =>
max_signal(30) <= '1';
out_byte(30) <= max_step_pos;
when "11111100" =>
zero_signal(30) <= '1';
out_byte(30) <= zero_step_pos;
when others =>
out_byte(31) <= cur_out_byte;
motor_light <= 31;
```

```
byte_pos <= byte_32;
end case;
when byte_32 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 31;
byte_pos <= byte_31;
when "11111101" =>
max_signal(31) <= '1';
out_byte(31) <= max_step_pos;
when "11111100" =>
zero_signal(31) <= '1';
out_byte(31) <= zero_step_pos;
when others =>
out_byte(32) <= cur_out_byte;
motor_light <= 32;
byte_pos <= byte_33;
end case;
when byte_33 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 32;
byte_pos <= byte_32;
when "11111101" =>
max_signal(32) <= '1';
out_byte(32) <= max_step_pos;
when "11111100" =>
zero_signal(32) <= '1';
out_byte(32) <= zero_step_pos;
when others =>
out_byte(33) <= cur_out_byte;
motor_light <= 33;
byte_pos <= byte_34;
end case;
when byte_34 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 33;
byte_pos <= byte_33;
when "11111101" =>
max_signal(33) <= '1';
out_byte(33) <= max_step_pos;
when "11111100" =>
zero_signal(33) <= '1';
out_byte(33) <= zero_step_pos;
when others =>
out_byte(34) <= cur_out_byte;
motor_light <= 34;
byte_pos <= byte_35;
end case;
when byte_35 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 34;
byte_pos <= byte_34;
when "11111101" =>
max_signal(34) <= '1';
out_byte(34) <= max_step_pos;
when "11111100" =>
zero_signal(34) <= '1';
out_byte(34) <= zero_step_pos;
when others =>
out_byte(35) <= cur_out_byte;
motor_light <= 35;
byte_pos <= byte_36;
```

```
end case;
when byte_36 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 35;
byte_pos <= byte_35;
when "11111101" =>
max_signal(35) <= '1';
out_byte(35) <= max_step_pos;
when "11111100" =>
zero_signal(35) <= '1';
out_byte(35) <= zero_step_pos;
when others =>
out_byte(36) <= cur_out_byte;
motor_light <= 36;
byte_pos <= byte_37;
end case;
when byte_37 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 36;
byte_pos <= byte_36;
when "11111101" =>
max_signal(36) <= '1';
out_byte(36) <= max_step_pos;
when "11111100" =>
zero_signal(36) <= '1';
out_byte(36) <= zero_step_pos;
when others =>
out_byte(37) <= cur_out_byte;
motor_light <= 37;
byte_pos <= byte_38;
end case;
when byte_38 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 37;
byte_pos <= byte_37;
when "11111101" =>
max_signal(37) <= '1';
out_byte(37) <= max_step_pos;
when "11111100" =>
zero_signal(37) <= '1';
out_byte(37) <= zero_step_pos;
when others =>
out_byte(38) <= cur_out_byte;
motor_light <= 38;
byte_pos <= byte_39;
end case;
when byte_39 =>
case cur_out_byte is
when "11111110" =>
motor_light <= 38;
byte_pos <= byte_38;
when "11111101" =>
max_signal(38) <= '1';
out_byte(38) <= max_step_pos;
when "11111100" =>
zero_signal(38) <= '1';
out_byte(38) <= zero_step_pos;
when others =>
out_byte(39) <= cur_out_byte;
motor_light <= 39;
byte_pos <= byte_0;
end case;
```

```
end case;
end if;
end if;
end process;

process(clk_x16)
begin
if clk_x16 = '1' and clk_x16'event then
case bit_pos is
when bit_start =>
if RXD = '0' then
in_byte(0) <= RXD;
count_16 <= 0;
ready <= '0';
bit_pos <= bit_0;
else
ready <= '0';
bit_pos <= bit_start;
end if;
when bit_0 =>
if count_16 >= 16 then
count_16 <= 0;
in_byte(0) <= RXD;
bit_pos <= bit_1;
else
count_16 <= count_16 + 1;
bit_pos <= bit_0;
end if;
when bit_1 =>
if count_16 >= 16 then
count_16 <= 0;
in_byte(1) <= RXD;
bit_pos <= bit_2;
else
count_16 <= count_16 + 1;
bit_pos <= bit_1;
end if;
when bit_2 =>
if count_16 >= 16 then
count_16 <= 0;
in_byte(2) <= RXD;
bit_pos <= bit_3;
else
count_16 <= count_16 + 1;
bit_pos <= bit_2;
end if;
when bit_3 =>
if count_16 >= 16 then
count_16 <= 0;
in_byte(3) <= RXD;
bit_pos <= bit_4;
else
count_16 <= count_16 + 1;
bit_pos <= bit_3;
end if;
when bit_4 =>
if count_16 >= 16 then
count_16 <= 0;
in_byte(4) <= RXD;
bit_pos <= bit_5;
else
count_16 <= count_16 + 1;
bit_pos <= bit_4;
end if;
when bit_5 =>
```

```
if count_16 >= 16 then
count_16 <= 0;
in_byte(5) <= RXD;
bit_pos <= bit_6;
else
count_16 <= count_16 + 1;
bit_pos <= bit_5;
end if;
when bit_6 =>
if count_16 >= 16 then
count_16 <= 0;
in_byte(6) <= RXD;
bit_pos <= bit_7;
else
count_16 <= count_16 + 1;
bit_pos <= bit_6;
end if;
when bit_7 =>
if count_16 >= 16 then
count_16 <= 0;
in_byte(7) <= RXD;
bit_pos <= bit_stop;
else
count_16 <= count_16 + 1;
bit_pos <= bit_7;
end if;
when bit_stop =>
if count_16 >= 16 then
count_16 <= 0;
cur_out_byte <= in_byte;
bit_pos <= bit_start;
else
count_16 <= count_16 + 1;
ready <= '1';
bit_pos <= bit_stop;
end if;
end case;
end if;

end process;


process(CLK_50MHZ)
begin
if CLK_50MHZ = '1' and CLK_50MHZ'event then
delay <= delay + 1;
if delay = "11111111" then
clk2 <= '1';
else
clk2 <= '0';
end if;
end if;
end process;

process(clk2)
begin
if clk2 = '1' and clk2'event then
if switch_m = "00000000" then
case motor_light is
when 0 =>
D0 <= "00010001";
D1 <= "00010001";
when 1 =>
D0 <= "11011011";
D1 <= "00010001";
```

```
when 2 =>
D0 <= "01000101";
D1 <= "00010001";
when 3 =>
D0 <= "01001001";
D1 <= "00010001";
when 4 =>
D0 <= "10001011";
D1 <= "00010001";
when 5 =>
D0 <= "00101001";
D1 <= "00010001";
when 6 =>
D0 <= "00100001";
D1 <= "00010001";
when 7 =>
D0 <= "01011011";
D1 <= "00010001";
when 8 =>
D0 <= "00000001";
D1 <= "00010001";
when 9 =>
D0 <= "00001011";
D1 <= "00010001";
when 10 =>
D0 <= "00010001";
D1 <= "11011011";
when 11 =>
D0 <= "11011011";
D1 <= "11011011";
when 12 =>
D0 <= "01000101";
D1 <= "11011011";
when 13 =>
D0 <= "01001001";
D1 <= "11011011";
when 14 =>
D0 <= "10001011";
D1 <= "11011011";
when 15 =>
D0 <= "00101001";
D1 <= "11011011";
when 16 =>
D0 <= "00100001";
D1 <= "11011011";
when 17 =>
D0 <= "01011011";
D1 <= "11011011";
when 18 =>
D0 <= "00000001";
D1 <= "11011011";
when 19 =>
D0 <= "00001011";
D1 <= "11011011";
when 20 =>
D0 <= "00010001";
D1 <= "01000101";
when 21 =>
D0 <= "11011011";
D1 <= "01000101";
when 22 =>
D0 <= "01000101";
D1 <= "01000101";
when 23 =>
D0 <= "01001001";
```

```
D1 <= "01000101";
when 24 =>
D0 <= "10001011";
D1 <= "01000101";
when 25 =>
D0 <= "00101001";
D1 <= "01000101";
when 26 =>
D0 <= "00100001";
D1 <= "01000101";
when 27 =>
D0 <= "01011011";
D1 <= "01000101";
when 28 =>
D0 <= "00000001";
D1 <= "01000101";
when 29 =>
D0 <= "00001011";
D1 <= "01000101";
when 30 =>
D0 <= "00010001";
D1 <= "01001001";
when 31 =>
D0 <= "11011011";
D1 <= "01001001";
when 32 =>
D0 <= "01000101";
D1 <= "01001001";
when 33 =>
D0 <= "01001001";
D1 <= "01001001";
when 34 =>
D0 <= "10001011";
D1 <= "01001001";
when 35 =>
D0 <= "00101001";
D1 <= "01001001";
when 36 =>
D0 <= "00100001";
D1 <= "01001001";
when 37 =>
D0 <= "01011011";
D1 <= "01001001";
when 38 =>
D0 <= "00000001";
D1 <= "01001001";
when 39 =>
D0 <= "00001011";
D1 <= "01001001";
when others =>
D0 <= "11101111";
D1 <= "11101111";
end case;
else
case SWITCHES(3 downto 0) is
when "0000" =>
D0 <= "00010001";
when "0001" =>
D0 <= "11011011";
when "0010" =>
D0 <= "01000101";
when "0011" =>
D0 <= "01001001";
when "0100" =>
D0 <= "10001011";
```

```
when "0101" =>
D0 <= "00101001";
when "0110" =>
D0 <= "00100001";
when "0111" =>
D0 <= "01011011";
when "1000" =>
D0 <= "00000001";
when "1001" =>
D0 <= "00001011";
when others =>
D0 <= "11101111";
end case;

case SWITCHES(5 downto 4) is
when "00" =>
D1 <= "00010001";
when "01" =>
D1 <= "11011011";
when "10" =>
D1 <= "01000101";
when "11" =>
D1 <= "01001001";
when others =>
D1 <= "11101111";
end case;

if SWITCHES(3 downto 0) < "1010" then
number <= conv_integer(unsigned(SWITCHES(5 downto 4))) *
10 + conv_integer(unsigned(SWITCHES(3 downto 0)));
motor_num <= std_logic_vector(to_unsigned(number, 8));
end if;

end if;


if eight = "000" then
D_ON <= "1110";
D <= D0;
elsif eight = "010" then
D_ON <= "1101";
D <= D1;
elsif eight = "100" then
D_ON <= "1011";
D <= D2;
elsif eight = "110" then
D_ON <= "0111";
D <= D3;
else
D_ON <= "1111";
end if;

eight <= eight + 1;
end if;
end process;

D2 <= ("11000001" AND NOT switch_e) OR ("00100101" AND switch_e);
D3 <= ("00000011" AND NOT switch_m) OR ("10100001" AND switch_m);

STEP <= step_signal;
DIR <= dir_signal;
ENABLE <= NOT SWITCHES(6);

switch_m <= SWITCHES(7) & SWITCHES(7) & SWITCHES(7) & SWITCHES(7)
        & SWITCHES(7) & SWITCHES(7) & SWITCHES(7) & SWITCHES(7);
```

```
switch_e <= SWITCHES(6) & SWITCHES(6) & SWITCHES(6) & SWITCHES(6)
        & SWITCHES(6) & SWITCHES(6) & SWITCHES(6) & SWITCHES(6);

MOTOR_NO_LIGHTS <= (count_to_step(conv_integer(unsigned(motor_num))) AND switch_m) OR (co

TXD <= '0';

end Behavioral;
```

B.2 GUI, and Tracking and Object Recognition Code

```
function varargout = basic_gui_V0_1(varargin)
% Last Modified by GUIDE v2.5 06-Nov-2016 12:49:01

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @basic_gui_V0_1_OpeningFcn, ...
'gui_OutputFcn',  @basic_gui_V0_1_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
if nargin && ischar(varargin{1})
gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
[varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before basic_gui_V0_1 is made visible.
function basic_gui_V0_1_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;
handles.vid = videoinput('pointgrey', 1, 'Mono8_1024x768');
src = getselectedsource(handles.vid);
handles.vid.FramesPerTrigger = inf;
handles.ispushed2 = 0;
handles.ispushed3 = 1;
start(handles.vid)
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = basic_gui_V0_1_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

% --- Executes on button press in RunButton.
function RunButton_Callback(hObject, eventdata, handles)

tic;
timerRunNum = 0;
timerAccum = 0;
numOfL = 20;
ispushed1 = get(hObject, 'Value');
handles.numOfL = uint16(20);
if ispushed1
```

```
set(hObject, 'String', 'STOP');
else
set(hObject, 'String', 'RUN');
end;
axes(handles.Cam1);
while get(hObject, 'Value')
I = getsnapshot(handles.vid);

[Y, X, ~] = size(I);
Y = uint16(Y);
X = uint16(X);
Y_p = uint16(get(handles.slider2, 'Value'));
X_p = uint16(get(handles.slider2, 'Value'));
gapY = (Y - Y_p) / 2;
gapX = (X - X_p) / 2;
leafWidth = uint16(round((X_p) / numOfL));
I_bw = im2bw(I,get(handles.filterslider1, 'Value'));
%     I_bw = bwareaopen(I_bw, 20);
%     I_bw = imcomplement(I_bw);
%     I_bw = bwareaopen(I_bw, 20);
%     I_bw = imcomplement(I_bw);
FirstRPos = uint16(ones(1, numOfL));
SecondRPos = uint16(ones(1, numOfL));
Sorted = uint16(zeros(1, numOfL));
c = uint16(0);
for x = gapX + round(leafWidth/2):leafWidth:X - gapX
c = c + 1;
yFirst = 1 + gapY;
while yFirst < Y - gapY && I_bw(yFirst, x) ~= 0
yFirst = yFirst + 10;                    % the 'jump'
% significantly negatively impacts the speed as more
% 'empty space' is on the screen
end
ySecond = Y - gapY;
while (ySecond > 1 + gapY || ySecond >= yFirst + 1) && I_bw(ySecond, x) ~= 0
ySecond = ySecond - 10;                  % jump
end
if yFirst > ySecond
Sorted(c) = 1;
yFirst = Y/2 - 1;
ySecond = Y/2;
end
FirstRPos(c) = yFirst;
SecondRPos(c) = ySecond;
end

CurSort = true;

for k = 1:numOfL
if Sorted(k) == 1 && CurSort == true
CurSort = false;
fPos = k - 1;
```

```
elseif Sorted(k) == 0 && CurSort == false
CurSort = true;
num = k - fPos;
if fPos == 0
pos2 = (SecondRPos(k) + FirstRPos(k)) / 2;
for n = 1 : k - 1
FirstRPos(n) = pos2 - 1;
SecondRPos(n) = pos2;
end
elseif num == 1
SecondRPos(fPos + 1) = (SecondRPos(fpos) + FirstRPos(fPos)) / 2;
FirstRPos(fPos + 1) = SecondRPos(fPos + 1) - 1;
elseif num == 2
SecondRPos(fPos + 1) = (SecondRPos(fPos) + FirstRPos(fPos)) / 2;
FirstRPos(fPos + 1) = SecondRPos(fPos + 1) - 1;

SecondRPos(k - 1) = (SecondRPos(k) + FirstRPos(k)) / 2;
FirstRPos(k - 1) = SecondRPos(k - 1) - 1;
else
pos1 = (SecondRPos(fPos) + FirstRPos(fPos)) / 2;
pos2 = (SecondRPos(k) + FirstRPos(k)) / 2;
g = (pos2 - pos1) / (num - 2);
for n = 0 : num - 2
SecondRPos(fPos + 1 + n) = pos1 + g * n;
FirstRPos(fPos + 1 + n) = SecondRPos(fPos + 1 + n) - 1;
end
end
end
end

if CurSort == false && fPos ~= 0
pos1 = (SecondRPos(fPos) + FirstRPos(fPos)) / 2;
for n = fPos + 1 : numOfL
FirstRPos(n) = pos1 - 1;
SecondRPos(n) = pos1;
end
CurSort = true;
end

if get(handles.OverlayButton, 'Value') == 0
for k = 1:numOfL
if FirstRPos(k) <= gapY
FirstRPos(k) = gapY + 1;
end
if SecondRPos(k) <= FirstRPos(k)
SecondRPos(k) = FirstRPos(k) + 1;
end
I_bw(1 + gapY : FirstRPos(k) , gapX + k * leafWidth ...
: gapX + k * leafWidth) = 0;
I_bw(1 + gapY : FirstRPos(k) , gapX + (k - 1) * leafWidth + 1 ...
: gapX + (k - 1) * leafWidth + 1) = 0;
I_bw(FirstRPos(k) : FirstRPos(k) , gapX + (k - 1) * leafWidth + 1 ...
```

```
: gapX + k * leafWidth) = 0;
I_bw(1 + gapY : 1 + gapY , gapX + (k - 1) * leafWidth + 1 ...
: gapX + k * leafWidth) = 0;

I_bw(SecondRPos(k) : Y - gapY , gapX + k * leafWidth ...
: gapX + k * leafWidth) = 0;
I_bw(SecondRPos(k) : Y - gapY , gapX + (k - 1) * leafWidth + 1 ...
: gapX + (k - 1) * leafWidth + 1) = 0;
I_bw(SecondRPos(k) : SecondRPos(k) , gapX + (k - 1) * leafWidth + 1 ...
: gapX + k * leafWidth) = 0;
I_bw(Y - gapY : Y - gapY , gapX + (k - 1) * leafWidth + 1 ...
: gapX + k * leafWidth) = 0;
end
end

if get(handles.ImageButton, 'Value') == 0
if get(handles.OverlayButton, 'Value') == 0
I(1 + gapY : Y - gapY , gapX : gapX) = 0;
I(1 + gapY : Y - gapY , X - gapX : X - gapX) = 0;
I(1 + gapY : 1 + gapY , gapX + 1 : X - gapX) = 0;
I(Y - gapY : Y - gapY , gapX + 1 : X - gapX) = 0;
end
imshow(I)
else
imshow(I_bw)
end

timerVal = 1 / toc;
tic;
timerAccum = timerAccum + timerVal;
timerRunNum = timerRunNum + 1;
if timerRunNum >= 5
set(handles.editFrameRate, 'String', timerAccum / 5);
timerAccum = 0;
timerRunNum = 0;
end
set(handles.SizeValue, 'String', get(handles.slider2, 'Value'));
set(handles.FilterText, 'String', get(handles.filterslider1, 'Value'));
guidata(hObject, handles);
end
toc;

% --- Executes on button press in OverlayButton.
function OverlayButton_Callback(hObject, eventdata, handles)

ispushed2 = get(hObject, 'Value');
handles.ispushed2 = get(hObject, 'Value');
guidata(hObject, handles);
if ispushed2
set(hObject, 'String', 'PLACE OVERLAY');
else
set(hObject, 'String', 'REMOVE OVERLAY');
```

```
end;
disp(ispushed2);

% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% Stops camera when closing down.
stop(handles.vid);
flushdata(handles.vid);
delete(hObject);


% --- Executes on slider movement.
function filterslider1_Callback(hObject, eventdata, handles)

set(handles.FilterText, 'String', get(handles.filterslider1, 'Value'));


% --- Executes during object creation, after setting all properties.
function filterslider1_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'), ...
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on button press in ImageButton.
function ImageButton_Callback(hObject, eventdata, handles)

ispushed3 = get(hObject, 'Value');
handles.ispushed3 = get(hObject, 'Value');
guidata(hObject, handles);
if ispushed3
set(hObject, 'String', 'ORIGINAL IMAGE');
else
set(hObject, 'String', 'FILTERED IMAGE');
end;
disp(ispushed3);

% --- Executes on slider movement.
function slider2_Callback(hObject, eventdata, handles)

set(handles.SizeValue, 'String', get(handles.slider2, 'Value'));


% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'), ...
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

Bibliography

[1] A. Boyer, P. Biggs, J. Galvin, E. Klein, T. LoSasso, D. Low, K. Mah, and C. Yu, "Basic applications of multileaf collimators," 2001.

[2] C.-S. Chui, M. F. Chan, and C. C. Ling, "Delivery of intensity-modulated radiation therapy with a multileaf collimator: comparison of step-and-shoot and dynamic leaf motion methods," in *Engineering in Medicine and Biology Society, 2000. Proceedings of the 22nd Annual International Conference of the IEEE*, vol. 1, 2000, pp. 460–462 vol.1.

[3] D. Comaniciu, V. Ramesh, and P. Meer, "Real-time tracking of non-rigid objects using mean shift," in *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, vol. 2, 2000, pp. 142–149 vol.2.

[4] M. Dewan, "Image-based tracking methods - applications to improved motion compensation in cardiac mr and image-guided surgery," Ph.D. dissertation, Baltimore, MD, USA, 2008, aAI3288449.

[5] *FDM0620 Datasheet*, Faulhaber, Feb 2016.

[6] M. Riboldi, R. Orecchia, and G. Baroni, "Real-time tumour tracking in particle therapy: technological developments and future perspectives," *The Lancet Oncology*, vol. 13, no. 9, pp. e383 – e391, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1470204512702437

[7] K. Somasundaram and K. Ezhilarasan, "Edge detection in mri of head scans using fuzzy logic," in *Advanced Communication Control and Computing Technologies (ICACCCT), 2012 IEEE International Conference on*, Aug 2012, pp. 131–135.

[8] J. Stewart and D. E. Davison, "Conformal radiotherapy cancer treatment with multileaf collimators: improving performance with real-time feedback," in *Proceedings of 2005 IEEE Conference on Control Applications, 2005. CCA 2005.*, Aug 2005, pp. 125–130.

[9] Y. y. Zheng, J. l. Rao, and L. Wu, "Edge detection methods in digital image processing," in *Computer Science and Education (ICCSE), 2010 5th International Conference on*, Aug 2010, pp. 471–473.