UNIVERSITY OF SOUTHAMPTON

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING

Electronics and Computer Science

A Stereoscopic Vision Robot

by

Henry S. Lovett

A project progress report submitted for the award of
MEng Electronic Engineering

Supervisor: Prof. Steve Gunn
Examiner: Prof. Mark Zwolinski

12th April, 2013

Turn off iNotes!

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
Electronics and Computer Science

A project report submitted for the award of MEng Electronic Engineering

A STEREOSCOPIC VISION ROBOT

by Henry S. Lovett

I don't like this. Consider changing...

This report describes the research, design, build and test of a stereoscopic robot using two cameras and an Atmel Microcontroller. A custom PCB was designed to fit a wheeled base. The report is in two main parts, hardware design and vision algorithms. The hardware design section describes the prototypes and design of the subsystems.

The vision algorithms section discusses range finding from stereo images, comparison algorithms between stereo pairs and the implementation and test of a two dimensional fast Fourier transform on an AVR.

The final product is a small mobile robot equipped with cameras and capability of image processing. The device has the ability to execute a predefined set of commands in an automatic stand alone way. The robot can also be used as a shell terminal to debug and remote control.

Contents

List of Figures

List of Tables

Listings

List of Symbols

I^2C	Inter-Integrated Circuit
TWI	Two Wire Interface
SCCB	Serial Camera Control Bus
SPI	Serial Peripheral Interface
kB	Kilobytes
ISR	Interrupt Service Routine
PCB	Printed Circuit Board
FIFO	First In - First Out
ADC	Analogue to Digital Converter
DAC	Digital to Analogue Converter
PWM	Pulse Width Modulation
GPIO	General Purpose Input Output
PLL	Phase Locked Loop
FPU	Floating Point Unit
SAD	Sum of Absolute Differences
SSD	Sum of Squared Differences
NCC	Normalised Cross Correlation
φ_0	Field of view of the camera
φ_1, φ_2	Angle from camera to the object
B	Separation distance of two cameras
D	Distance from camera to the object
i, j	Pixel index of an Image
x_0	Horizontal resolution of the image
x_1, x_2	Distance of object from the normal of the camera
δ	Distance to move
γ	Number of tabs on the wheel
A	Angle of rotation
C_w	Circumference of the wheel
r_b	Distance from centre of the robot to the wheel

C_b	$2\pi r_b$

Acknowledgements

I would like to thank Professor Steve Gunn whose help throughout the year was invaluable and for the weekly meetings no matter where he was in the world. I would also like to thank my friends and family for their support and help throughout my degree and with proof reading, to Tom for our years of being lab partners during our time here, and finally to Alice for asking the two golden questions that solved 90% of my problems whenever I had one; “Is it connected?” and “Is it switched on?”.

All work presented in this report is my own unless otherwise stated or referenced.

Chapter 1

Introduction

The original idea for the project was a stereoscopic mapping robot, similar to Goebel (2012). This would autonomously search an area and return an occupancy map (Thrun, 2003). However, due to time constraints, the image processing aspects remain prototyped in MATLAB, but not implemented in C. The end robot is able to capture stereo image pairs, move with reasonable accuracy compute Fourier Transforms. The robot can run a predefined set of commands automatically or act as a shell terminal.

Stereoscopy in computer vision is the ability to calculate the locations and depths using images from two or more cameras, which are used to triangulate and estimate distances (Saxena et al., 2007). By using two cameras on the same plane, separated by a set horizontal distance, the depth of the observed scene can be perceived by the system.

Stereo vision is a small section of computer vision which is widely used in many applications, including Microsoft's Xbox Kinect (Microsoft, 2012), where stereo vision is used to locate a game player in order to use their movements to control the game. Mrovlje and Vrančić (2008) uses stereo vision to be able to locate the distance to a marker.

Leap Motion?

The stereo vision robot discussed in this report is a low cost alternative to other robots which use laser range finders or high quality cameras (Se et al., 2002). The robot used the base seen in figure 1.1 and use two OmniVision OV7670 cameras delivering QVGA format images.

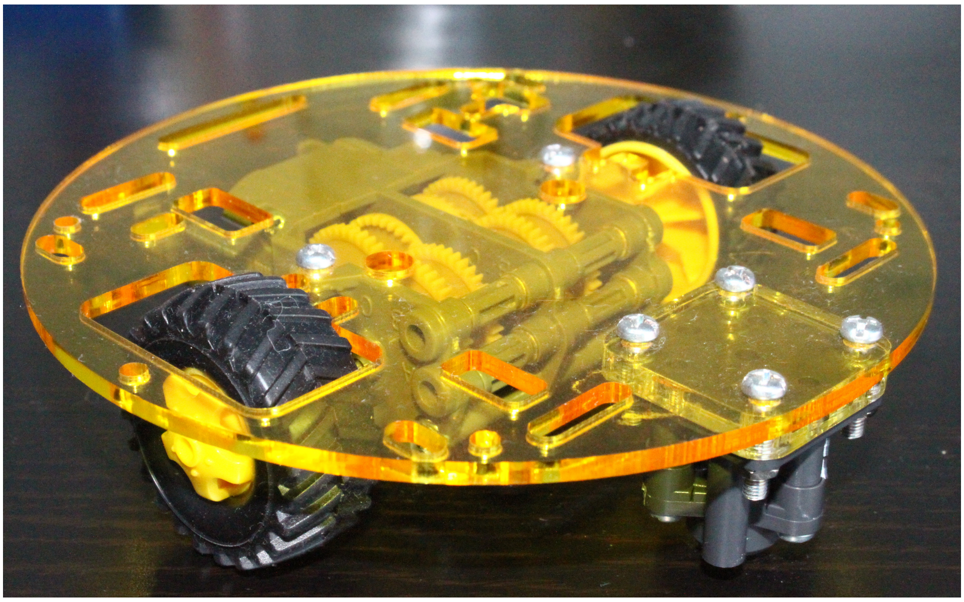


Figure 1.1: The base of the robot

The final robot designed could be used for a variety of applications. The robot can perceive depth of the captured area so can avoid obstacles and navigate. The robot could also be adapted to stream the camera data to a remote computer and be controlled by a user to explore unknown and potentially hostile areas safely.

1.1 Project Management

In order to reduce the risk within the project, all aspects of potential issues were looked at and are summarised in table 1.1. A Gantt chart of how time was planned to be spent can be seen in figure A.1.

The project was designed in stages - first, gaining operation of all the basic sections; movement, image capturing, image detection algorithms etc. These were then be brought together once tested to create the final product. This meant functionality was obtained of the basic aspects before addressing more complex issues.

Risk	Severity	Prevention
Components not arriving on time	High	Order parts as early as possible
Project not fulfilling specification	High	Develop in stages to obtain functionality in parts. Ensure enough time is allocated to the project.
PCB Design is incorrect	Medium	Check the design carefully and get second opinion.
Failure of personal computer causing data loss	Low	Keep back ups of all work on Devtrack Git repository and Dropbox.

Table 1.1: A list of risks and the prevention steps taken to reduce their impact

Chapter 2

Research

The research for this project was split into three sections:

1. Hardware
2. Software, broken down into:
   (a) Firmware, and
   (b) Image Processing

Hardware and firmware research will be discussed in this section. Image processing is looked at in detail in chapter 4.

2.1 Hardware Research

2.1.1 Microcontrollers

The budget for the design was £80 (not including PCB). The choice of microcontroller was an important one, as a compromise between cost, power and usability had to be made. There are two main brands of microcontrollers present in the consumer market: ARM and AVRs.

ARM is an architecture which is developed by ARM Holdings. ARM devices come in a many varieties: ARM9, ARM7, Strong ARM, ARM Cortex etc. Whilst ARM Holdings do not fabricate and sell the devices themselves, many companies, such as Texas Instruments, use the architecture and manufacture their own devices. ARM cores are based on a RISC Harvard architecture and tend to be 32-bit with a high clock speed. ARM microcontrollers have on chip support for SPI, I^2C , PWM, ADCs and can have Flash, SRAM and EEPROM memory built-in. For this comparison, the Stellaris by Texas Instruments will be examined.

Atmel have a variety of products in the microcontroller market, which have an AVR core that is a predominantly 8-bit, Harvard RISC architecture. They range from a low clock speed device for the hobbyist (ATMega and ATTiny series), to an improved 8-bit variant (XMega), and a 32-bit architecture (AT32UC3). XMegas and AT32s tend to have higher clock speeds than the ATMega. Atmel devices often have on board peripherals such as I^2C , SPI and ADCs, as well as a number of different memories: Flash, EEPROM and SRAM. An AT32UC3C0512C, ATXmega256A3BU and ATMega644P will be compared in this section.

Table 2.1 shows a brief summary of some common ARM and AVR microcontrollers. The Stellaris offers the most power with the largest DMIPS performance. However, due to the necessity of floating point operations, the AT32 clearly has a distinct advantage by having a built-in floating point unit. The XMega and ATMega do not offer enough power and are restricted by a small amount of SRAM and Flash. All devices looked at use 3.3V supply and have basic communication protocols (SPI, I^2C and USART). Overall, the AT32UC3 is the best choice with a high throughput, a floating point unit and a vast amount of GPIOs and communications. There is no EEPROM which may be desirable, but these can be added onto an SPI or I^2C bus. This device, although slightly more costly, is best suited to this application out of the selection researched.

2.2 Firmware

2.2.1 Camera

The camera used is the OV7670 by OmniVision. Steve Gunn provided source code for use on the Il Matto development board, which streamed video from the camera to a colour TFT screen. The camera is supplied on a small breakout board with a FIFO buffer. The camera operation is discussed in section 3.1. Many implementations of firmware for this camera exist across different devices.

Attribute	ARM Stellaris	AT32UC3C0512C	XMegaA3BU	ATMega644P
Clock Speed (MHz)	80	33 or 66	32	12
DMIPS	100	91	-	20 MIPS
Package	100 LQFP or 108 BGA	64, 100, 144TQFP	64 QFP or QFN	40 DIP, 44 TQFP, 44 QFN
Cost of 1 unit(£)	10.30	15.39	6.65	6.86
Flash Size(kB)	256	512	256	64
SDRAM Size (kB)	32	64	16	4
EEPROM Size(kB)	2	None internal	4	2
GPIO	64	45, 81 or 123	47	32
Operating Voltage (V)	3.3	5 or 3.3	1.6- 3.6[1]	2.7-5.5
Communication Interfaces	SPI, I^2C , SSI, MAC, CAN, EPI, USB, USART, I2S	SPI, TWI, EBI, USB, Ethernet, CAN, USART, I2S	USART, TWI, USB, SPI	SPI, TWI, USART
Floating Point	None	Built in FPU	None	None
ADCs	16	16	16	8
Timers	4	3 16-bit	7 16-bit, 8 8-bit	2 8-bit, 1 16-bit

Table 2.1: Comparison Table of some common microcontrollers. Data of microcontrollers taken from Atmel Corporation (2012a), Atmel Corporation (2012b), Atmel Corporation (2012d) and Texas Instruments (2012). Costings from Farnell (2012)

2.2.2 Atmel Software Framework

Atmel offer a software framework which contains basic code and device drivers for many of their XMega and AT32 devices (Atmel Corporation, 2009). There are also many AVR application notes which provide explanations and example code for protocols like I^2C , SPI and timers. These application notes are aimed at older devices like the ATTiny and ATMega. Both the ASF and the application notes are available online and are a good source of code for basic protocols.

Chapter 3

Hardware and Firmware Development

For initial development, the *'Il Matto'* board, designed by Steve Gunn, was used. The system has an ATMega644P, 12MHz clock and an on board SD card socket. This provided the ability to prototype circuits which were then used to create a PCB.

3.1 Camera

The camera used is an OV7670 by OmniVision. It is mounted onto a break out board and connected to a AL422B FIFO Buffer. The breakout board has all passive components and a 24MHz clock mounted. The schematic for the device can be seen in appendix B. The camera has a small hardware modification, due to a PCB fault. Pin 8 of the buffer is lifted and connected to pin 6 of the header, which is disconnected from the PCB (see figure 3.1). Original code for the camera operation was given by Steve Gunn which streamed continuous video to a TFT screen from the camera. The operation required was to take a single photo from the camera and store the data.

3.1.1 Single Camera Operation

The camera uses a SCCB Interface (OmniVision, 2007) created by OmniVision. This is almost identical to the I^2C Interface by Phillips and the two protocols

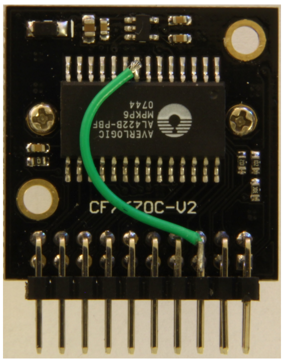


Figure 3.1: Reverse side of the OV7670 Camera showing the modification

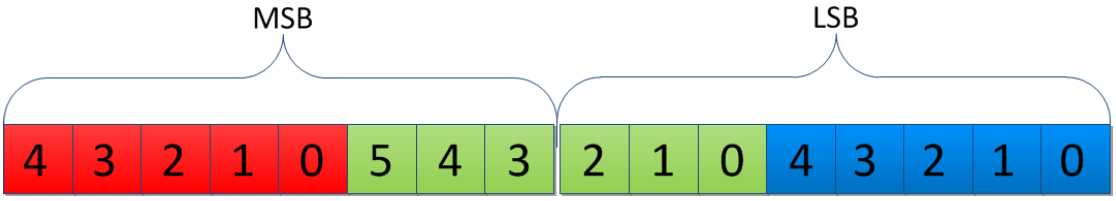


Figure 3.2: RGB565 pixel format

are compatible. The original code for the camera used a software driven SCCB interface which was very slow and used up processing time. This was changed to make used of the built-in interrupt-driven I^2C interface (named TWI in Atmel AVRs)[1]. This communication bus is used to write to the control registers of the camera to set up the signals, image format and image size. The set up procedure is used from the code given and sets the camera to function as explained below.

RGB565 is a 16 bit pixel representation where bits 0-4, 5-10 and 11-15, represent the blue, green and red intensity respectively (see figure 3.2). This is a compact way of storing data but only allows 65536 colours. Greys can also appear to be slightly green due to the inconsistent colour ratio of the green field. This representation was used as it is a compact format, easily converted to grey scale and is widely used.

The camera must use a high speed clock in order to ensure the pixels obtained are from the same time. This makes it difficult for an ATMega (typically clocked at 8-12MHz) to be able to respond to the camera quick enough. This highlights the necessity for a FIFO Buffer.

The OV7670 is set up so that the VSYNC pin goes low at the beginning of every frame of data, and HREF is high when the data being output is valid. The pixel

[1] I^2C , SCCB and TWI are all the same but are called differently due to Phillips owning the right to the name "I^2C"

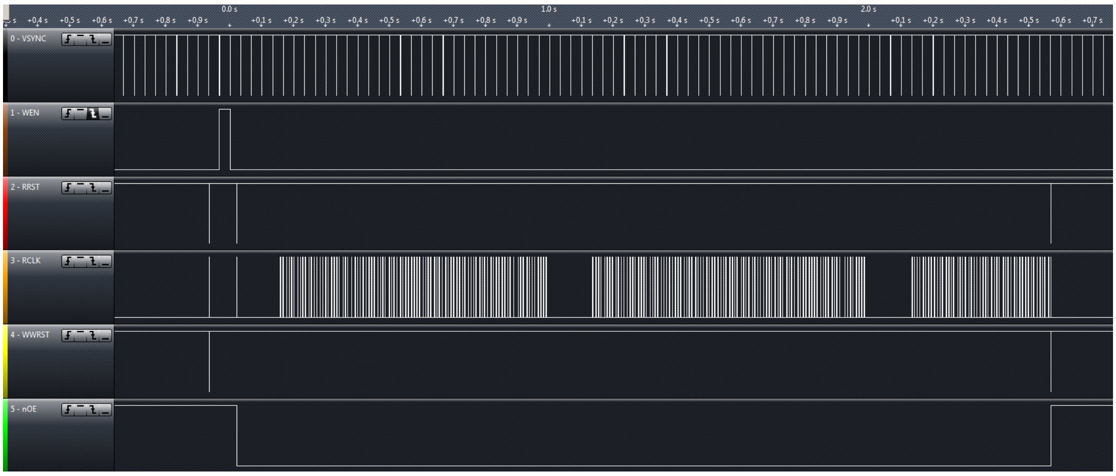


Figure 3.3: Signals generated to control the OV7670 capture and read

data is then clocked out on every rising edge of PCLK. To control the buffer, WEN (write enable) is NAND with the HREF signal. When both are high, the write enable to the buffer will be active and the data will be clocked in by PCLK.

In order to acquire a full image, WEN must be high between two consecutive VSYNC pulses. VSYNC is set up to interrupt the AVR and a small state machine is implemented to count VSYNC pulses and control WEN correctly. After the WEN signal is pulsed, the buffer will contain all the valid pixel data.

To obtain the data from the buffer, the AVR sets output enable low and pulses the read clock. Valid data is available on the data port while RCLK is high. All the data is then read half a pixel at a time (the endianness of the data can be set up on the camera).

After the data has been read, the buffer is reset by asserting the read and write reset signals (RRST and WRST) for at least one clock cycle of the relevant clock. The entire operation can be seen in figure 3.3.

Difficulties arose at this point with the storage of the data. The ATMega644P has 4kB of internal SRAM, but 153.6kB of memory is needed to store a single image at QVGA (320 by 240 pixels, 2 bytes per pixel) quality.

Firstly, data was sent to a desktop computer by USART. A simple program was written in C# to receive and store all the data as a Bitmap image. This method was slow, taking around 30 seconds to transmit one uncompressed image.

The second option was to use extra memory connected to the microcontroller. An SD card was used as FAT file system so the memory card is usable on a computer. Text log files were also written to aid debugging. This is discussed in section 3.2.

3.1.2 Dual Camera Operation

In order for stereovision to be successful, two cameras separated by a horizontal distance (B) will need to be driven at the same time to obtain photos within a small time frame of one another.

A major problem occurred with using the I^2C interface to set up both cameras. The camera has an I^2C address of 21_{16}, which cannot be changed. Multiple I^2C devices with exactly the same address cannot be used on the same bus. Two solutions to this are possible: driving one from I^2C and one from SCCB, or using an I^2C multiplexer. By using two different buses the cameras will be individually addressable. However, SCCB is slow and processor-hungry as it is software driven. This takes up program memory and is not reusable for other operations.

An I^2C multiplexer sits on one I^2C bus and has multiple output buses. The master can then address the multiplexer and select whether to pass the bus to channel 0, channel 1 or not allow the data to be transferred. This saves processor time, but means a write operation has to be done to select the camera bus before being able to write to the camera. This slows down the operation slightly, but not as much as using SCCB. The main disadvantage to the I^2C multiplexer is the extra hardware needed; firstly the multiplexer itself, but also 7 extra resistors to pull up the two extra buses and the three interrupt lines which aren't used, must be added.

Overall, the disadvantages posed by using a multiplexer are small, so it was used as opposed to the SCCB interface. A suitable multiplexer is the Phillips PCA9542A (Phillips, 2009).

The buffers have an output enable pin so the data bus can be shared by both cameras to the AVR. The ATMega644P offers three interrupt pins, two of which are used by the two VSYNC pins for the cameras to drive two individual ISRs.

Operation to read an image is identical to using one camera. The code was duplicated so that the two cameras were accessed individually. Care was taken to avoid any bus contention on the data bus, but no checking is explicitly done. When taking a photo, both frames are taken at a time period close together to capture the same scenario. The data can then be read in either order and stored.

	Bitmap	JPEG	PNG	GIF
Extension	*.bmp	*.jpg /*.jpeg	*.png	*.gif
Compression	No	Lossless and Lossy	Lossless ZIP	Lossy
File Size of 320 by 240 pixel Image (kB)	225	20	23	24
Bits per Pixel	8, 16, 24 or 32	24	24, 32 or 48	24, but only 256 Colours

Table 3.1: A table comparing different image formats available (Fulton, 2010)

3.2 SD Card

An SD card was chosen due to its small size, low cost and a large data storage. Smaller variants also exist, the mini and micro SD. The cards work using an SPI bus which can be used for other devices within the system as well. For the prototyping stage, the FATFS library (Electronic Lives Manufacturing, 2012) was used. This supplied all basic read and write operations for the SD card. For the AT32, the ASF supplied a more comprehensive FAT library and was used in the final product.

3.2.1 Storing Images

Many image formats are common, such as Joint Photographic Expert Group (JPEG), Portable Network Graphics (PNG), Bitmap (BMP) and Graphics Interchange Format (GIF). Table 3.1 shows a summary of these common image formats.

It is clear that the best choice for images would be either PNG or JPEG. However, these require more computational time to compress the image into the correct format. To avoid compression, and thereby save processing time, bitmap was chosen at the expense of using more memory. The data in a bitmap image is also stored in RGB format so can be read back easily when processing the image. Appendix F shows the make up of a Bitmap File that was used.

By writing the image in this format, they are then able to be opened on any operating system. This aids debugging and allows the prototyping of image algorithms

Figure 3.4: An Example Image taken using the OV7670 and stored as a Bitmap on the SD Card

in a more powerful environment. Figure 3.4 shows a photo taken by the OV7670 and stored on a SD card. The quality is not professional, but all features can be seen. The colour is accurate and large text can be read at a small distance. Though the images were able to be opened in most programs, MATLAB gave errors when reading the bitmaps. This could be avoided by opening and resaving the images in MS Paint.

3.3 The Prototypes

The first prototype made can be seen in figure 3.5. This obtained two images from both cameras, and stored them to the SD Card. The cameras shared the data bus and a I^2C multiplexer was used. An ATMega168 was added to the system as no pins remained for debugging on the *Il Matto* board. The pinout for the 644 can be seen in table 3.2. The ATMega168 used the already existing I^2C bus and acted as a port extender - button inputs were read and status LEDs were written to. This enabled a debug output for the system. The prototype worked well and proved the circuit worked.

A second prototype was made to stream the images to a TFT screen. This can be seen in figure 3.6. This provided the ability to check the cameras worked, and

	Port A	Port B	Port C	Port D
0	Data 0	SD Write Protect	I^2C - SCL	No Connection
1	Data 1	SD Card Detect	I^2C - SDA	No Connection
2	Data 2	USB Data Plus	Read Clock 1	VSync 0
3	Data 3	USB Data Minus	Read Reset 1	VSync 1
4	Data 4	SPI Chip Select	Write Enable 1	Read Clock 0
5	Data 5	SPI MOSI	Write Reset 1	Read Reset 0
6	Data 6	SPI MISO	Output Enable 0	Write Enable 0
7	Data 7	SPI Clock	Output Enable 1	Write Reset 0

Table 3.2: Pin Connections of the ATMega644P for Dual Camera Operation.

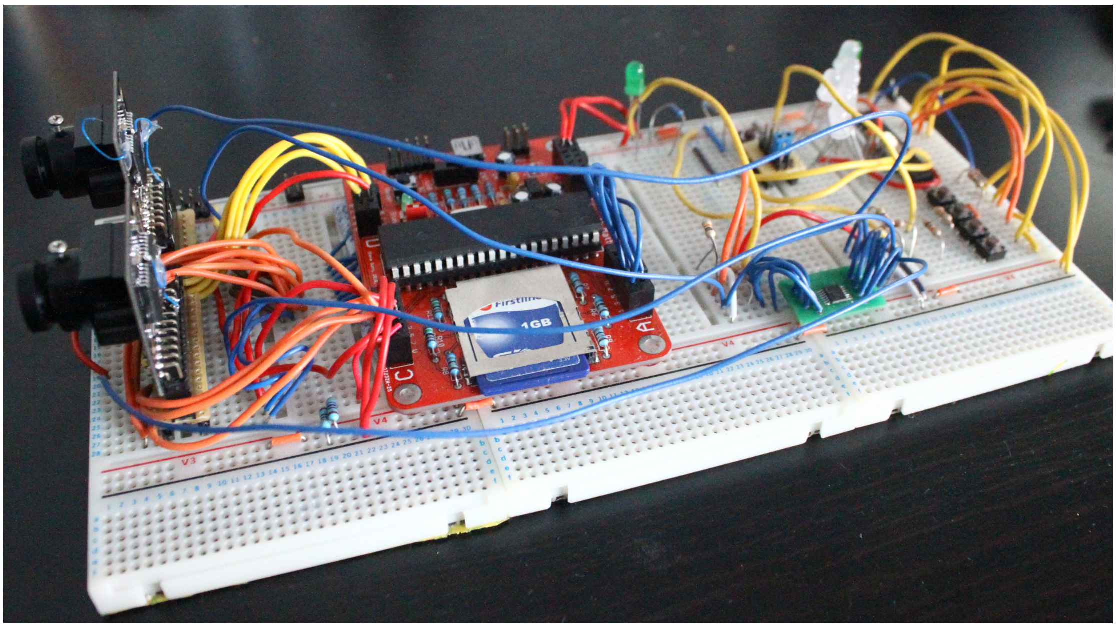


Figure 3.5: Prototype of Dual Camera operation.

gave a live stream so the lens could be focused to give a clear image. The refresh rate of a full image was slow ($\approx 0.6fps$), but the system was successful.

3.4 Motor Driver Development

3.4.1 Hardware

Tachometers are devices used to measure rotational speed of a wheel. They are commonly found in bicycles where a small magnet is attached to the wheel and a sensor is attached to the frame. The elapsed time between every rotation is measured by the and speed can be calculated given the wheel size.

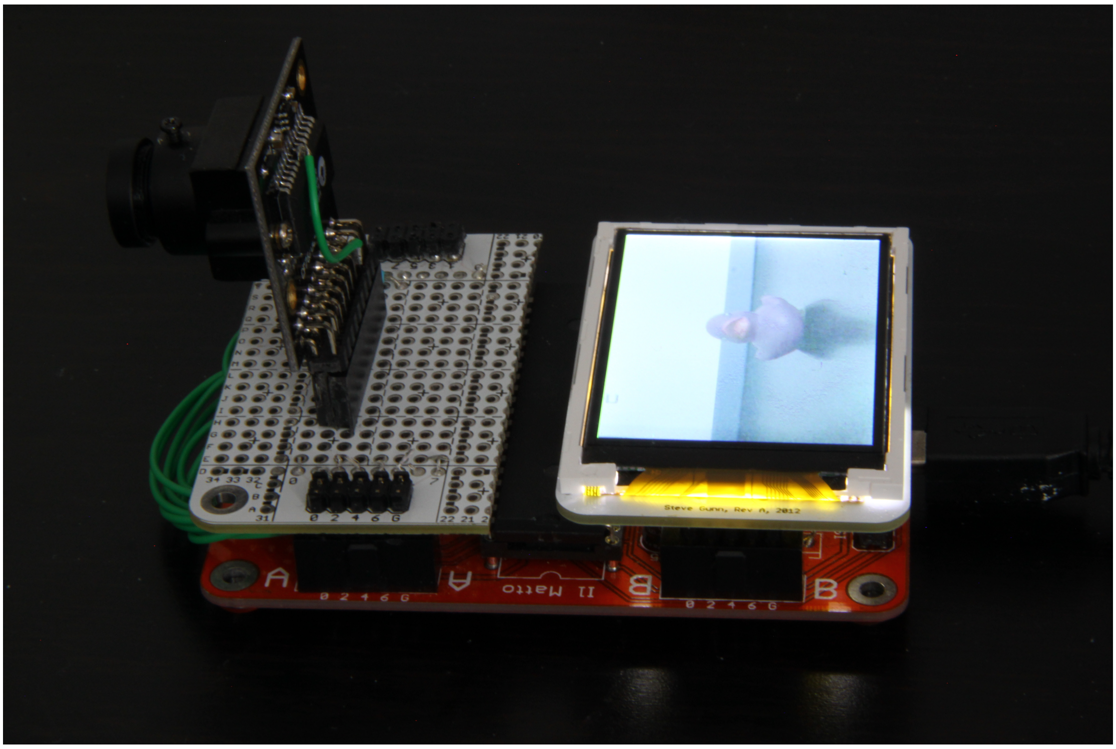


Figure 3.6: Prototype of streaming camera images to TFT screen.

Here an optosensor, the TCRT1010 made by Vishay Semiconductors (2012), is used to measure rotations of the wheel and therefore be able to move a distance determined by the microcontroller. The TCRT1010 package contains an IR LED and a phototransistor. The schematic of a simple transistor amplifier used can be seen in figure 3.7 which was reproduced from Gunn (2012).

A similar method to the way a bike measures speed was used. The wheel's rubber absorbed the IR from the LED, so a high voltage was always seen at the collector of the phototransistor when near the wheel. White Tipp-Ex marks, "tabs", were applied to the wheels at regular intervals. These tabs reflected IR, resulting in a lower collector voltage when aligned and thereby giving a way to detect wheel rotation. Figure 3.8 shows the voltage at the collector (read by the ADC on the AVR) against the angle of the wheel. Ten tabs were marked on the wheel, and ten dips in the voltage can be seen in figure 3.8. A lot of noise exists due to imperfect white tabs. However, the dips are prominent and can be detected with the correct threshold voltage.

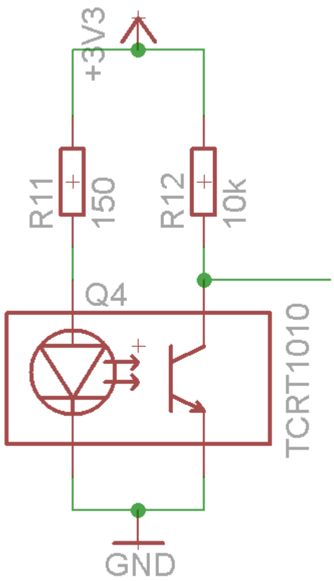


Figure 3.7: Circuit diagram of Optosensor

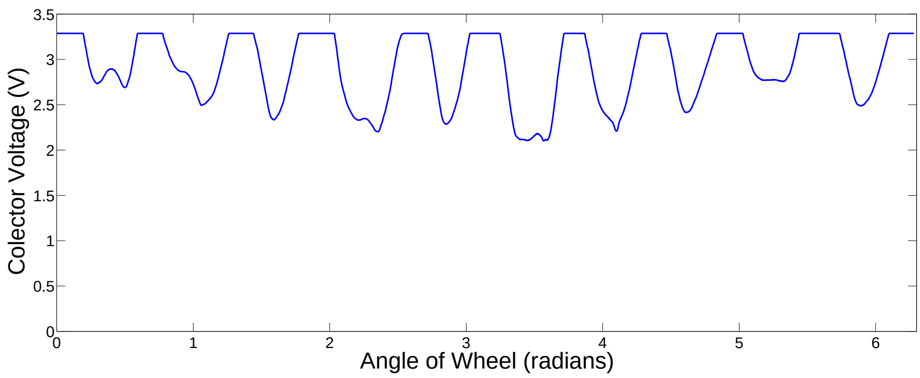


Figure 3.8: Graph of Wheel Angle against the Voltage read by the AVR

3.4.2 Firmware Development

To move a distance, the number of times a tab passes the sensor, a count, must be detected. The firmware sets up a PWM output and has set up and execute functions. A low duty cycle PWM signal was used to drive the motors slowly. This removed the need for a speed controller and meant any overshoot of the motors during breaking would be low. Straight line movement and spot rotations were implemented. More complex movements, for example arcs, would require more accurate tachometers and are not discussed here.

Table 3.3: A table to show the inputs to the motor drivers to set the movement

	Motor 1			Motor 2		
	STBY	IN1	IN2	STBY	IN1	IN2
Stop	0	X	X	0	X	X
Forward	1	1	0	1	1	0
Backward	1	0	1	1	0	1
Clockwise	1	1	0	1	0	1
Anti-clockwise	1	0	1	1	1	0

3.4.2.1 Set Up

The set up calculates the number of counts needed, and sets the direction of the wheels. Each motor driver has two inputs to control direction and table 3.3 shows the inputs needed to generate the two movements - straight line and rotational.

To move in a straight line, the number of counts can be calculated by equation (3.1).

$$\text{Counts} = \delta \times \frac{\gamma}{C_w} \tag{3.1}$$

For rotation, the radius from the centre of the robot to the wheels needs to be known, see figure 3.10(a). The circumference through the wheels is then easily calculated and the distance to move is calculated by equation (3.2). The total number of interrupts can be calculated using equation (3.1).

$$\delta_R = A \times \frac{C_b}{360} \tag{3.2}$$

Combining equations (3.1) and (3.2) gives:

$$Counts = \gamma \times \frac{C_b}{C_w} \times \frac{A}{360} \tag{3.3}$$

Figure 3.10 shows the dimensions of interest of the robot base. In the set up method, equations (3.1) and (3.3) are used to calculate the counts needed. The results are placed in the global struct and the execute method is then run.

3.4.2.2 Execute

The execute method detects the tabs and decrements the global counter. When the counter reaches zero the motor stops and when both motors have completed, the method exits.

As the voltage swing on the collector of the phototransistor did not reach near 0V, the AVR could not detect this as a logical 0. An external amplifier could have been used to generate a full swing voltage. However, the AVR has internal ADCs and analogue comparators which were used instead.

The ADC could be used to continually sample the collector voltage and detect dips in the signal. This method has the advantage of a variable threshold and the ability to filter noise. The operation, however, would be complex to implement in code.

An alternative was to use an analogue comparator. The UC3C has two on chip comparator interfaces, each with two comparators. They are a high gain operational amplifier with added options of hysteresis and the ability to interrupt amongst other attributes. The comparators can use two analogue inputs or use the internal DACs as inputs. Potentiometers were used to set the reference voltage externally. Using the comparators had the advantage of returning a boolean value of if the collector voltage is higher or lower than the threshold voltage. The detection is of a tab was then simpler. This method was decided on for an easier implementation.

The first method implemented was to use the analogue comparators to interrupt when the collector voltage crossed the threshold. Both wheels used the same comparator interface and therefore ran the same ISR when triggered. The ISR then had to read the output of the comparators and decrement the relevant counter for the correct wheel. This method had many downfalls. First, it was not possible to know which comparator caused the interrupt. If the left wheel triggered the interrupt while the right was below the threshold, both left and right counters would be decremented. This caused a large error each time it occurred. Another problem was noise; occasionally, the lowering voltage would cause multiple interrupts each time. This problem was reduced by setting the hysteresis on the comparators but did not solve the problem completely.

A second approach utilised a simple state machine and software based hysteresis to solve the problems. The state machine has two states 'On a Tab' and 'Not on

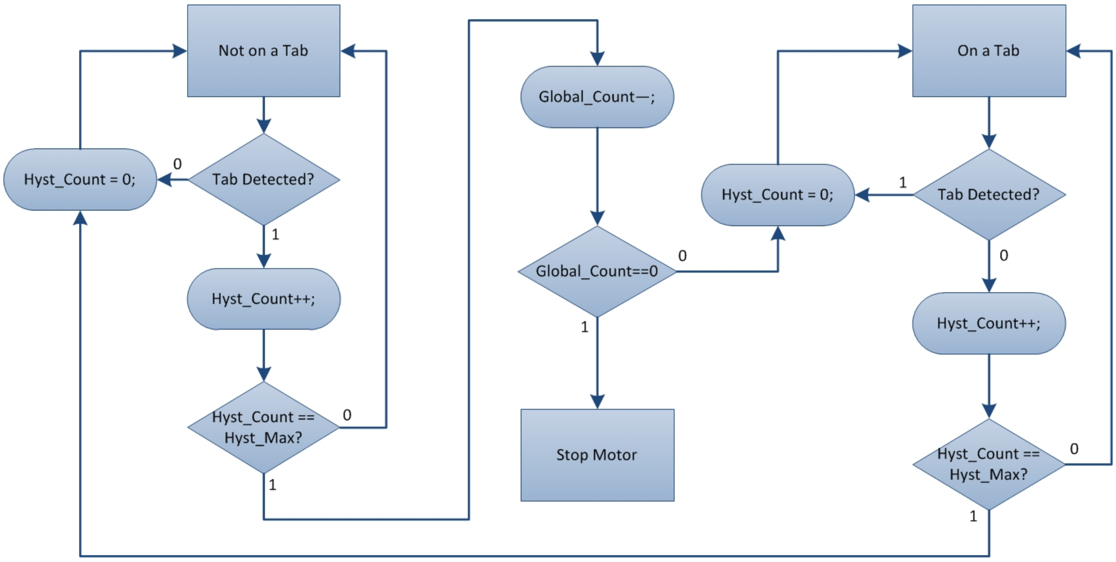


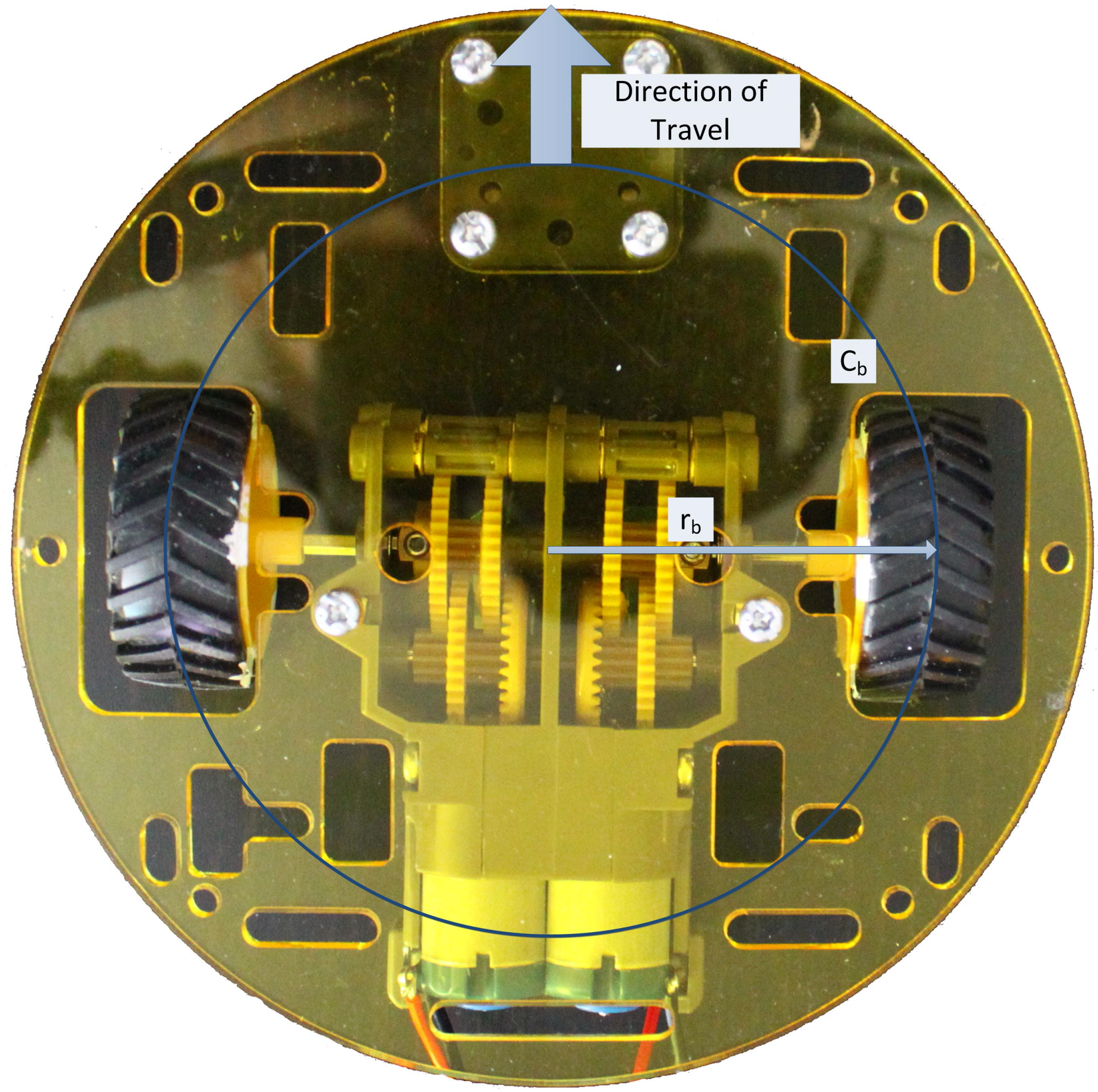
Figure 3.9: A State Machine showing the operation of the *Motor_Execute* method

a Tab'. When the method enters, the initial state is read from the comparators so that if the wheel is already on a tab, it won't be counted. The code then runs in a loop until the motors stop. A graphic representation of the code can be seen in figure 3.9. Starting in 'Not on a Tab' state, a tab must be detected for Hyst_Max amount of cycles in succession for the state to change. This is the same for going from 'On a Tab' to 'Not on a Tab'. This is to reduce noise in the form of false readings an increase the certainty that a tab is in detect. The Global_Count is only reduced on the transition to 'On a Tab' so it is difficult for this to decrement multiple times in normal operation. To increase the certainty, Hyst_Max can be increased at the expense of response time.

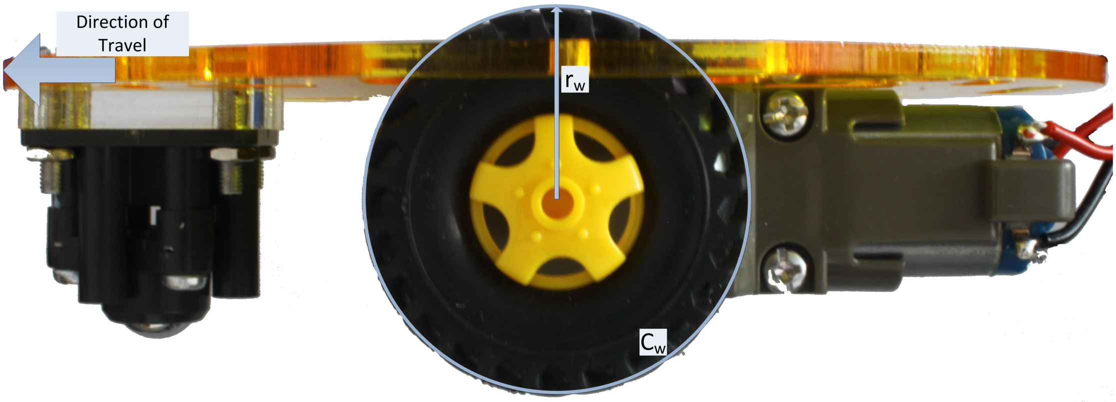
The final code can be seen in appendix H. *Motor_Init* method must be called before operation can occur. This sets up the PWM and analogue comparators. Methods *Motors_Move* and *Motors_Rotate* are the methods that can be called to move in a straight line or rotate on the spot. They both take an input which is a signed integer of either the distance to move (in millimetres) or the angle to rotate (in degrees, positive is a clockwise movement). They both return the actual movement distances, due to the low resolution of the system.

3.4.3 Testing

To test the motor system, different distances were given to the movement method. The method prints how many counts will be moved. The actual distance moved



(a) Top View of robot base showing dimensions of interest



(b) Side View of robot base showing dimensions of interest

Figure 3.10: Dimensions of Interest for Robot Movement

Table 3.4: Results of Motor Distance Test

Distance Specified (mm)	Number of Counts Calculated	Counts $\times$ Resolution (mm)	Average Measured Distance Moved (mm)	Error (%)
50	4	46.4	47.25	1.83
75	6	69.6	70.75	1.65
100	8	92.8	97.0	4.52
120	10	116.0	113.25	2.37
150	12	139.2	141.0	1.29
170	14	162.4	161.5	0.55
200	17	197.2	201.5	2.18
250	21	243.6	244.5	0.37
300	25	290.0	290.75	0.26

Table 3.5: Results of motor speed test

Wheel	Total Time Elapsed (s)	Calculated Speed ($mm.s^{-1}$)
Left	44.6	20.8
Right	49.6	18.7

was then measured and repeated four times. Table 3.4 shows the results of this test. They show that the maximum error observed is 4.52%, which related to 4.5mm. This error is acceptable as a half centimetre error over 10cm will not impact the performance of the robot. The error is calculated from the actual distance predicted to move.

A problem was seen that the robot moved in a slight arc. Speed tests were done on the wheel by measuring the total time taken to complete eight full revolutions, the equivalent of moving 928mm. The results and the calculated wheel speeds can be seen in table 3.5. It shows that the left motor runs slightly faster than the right, even though the PWM duty cycle is the same. A controller could be implemented to correct this error during operation. However, over the distances covered, the error introduced by this is small enough to neglect.

3.4.4 Conclusion

Due to the low resolution of the sensor (10 counts per revolution), there is a minimum distance that can be moved and a minimum angle of rotation, shown in equations (3.4) and (3.5) respectively. These show a that greater distance resolution could be obtained by decreasing the wheel size or increasing γ and a greater rotational resolution could be obtained by the same as distance, or by increasing the distance the wheels are from the centre of the robot. In general, the ratio $C_w : \gamma$ should be as large as possible to obtain the best resolution for movement.

$$\Delta_\delta = \frac{C_w}{\gamma} = 12mm \tag{3.4}$$

$$\Delta_\theta = \frac{360 \times C_w}{\gamma \times C_b} \approx 15^\circ \tag{3.5}$$

This method lacks on two points - lack of speed control and accuracy. A better controller could be implemented to help move at different speeds. This could use the remaining number of counts to gradually slow the speed of the motors down as well as correct the speed mismatch between the wheels. A PID controller could be implemented if greater accuracy is needed quicker, at the expense of computation time and potential overshoot.

Rotary encoders could be used to detect the direction of wheels as well. A good, but more costly, alternative would be the HUB-ee wheel by Creative Robotics Ltd (2013), which includes a 120 point quadrature encoder and sensor, motor driver and a geared motor all within the wheel. These wheels have 12 times the accuracy as the method described here and a similar interface.

Given that the robot does not need to move any more accurately that to 1cm, this method has proved to be cheap and successful. The robot is able to move a distance with reasonable accuracy, but to a fairly low resolution.

3.5 PCB Development

3.5.1 Circuit Design

Figure 3.11 shows a basic hierarchy of the robot. Each pin on the UC3C can have one of up to six special functions, as well as being a GPIO pin. Table 3.6 shows the pinout for the microcontroller used.

The circuit diagram for Revision A can be seen in section B.3. The schematic for the SDRAM and values and locations of decoupling capacitors were used from the schematic of the UC3C-EK development board (Atmel Corporation, 2012c).

Table 3.6: The Pinout of the AVR for the circuit. '-' means unavailable and blank means unused

	Port			
Pin	A	B	C	D
0	TCK	CAMERA_ 0		EBI-DATA13
1	TDI	CAMERA_ 1		EBI-DATA14
2	TDO	CAMERA_2	SDA	EBI-DATA15
3	TMS	CAMERA_3	SCL	EBI-ADDR0
4	USB ID	CAMERA_4	USART TXD	EBI-ADDR1
5		CAMERA_5	USART RXD	EBI-ADDR2
6	AC R	CAMERA_6		EBI-ADDR3
7	AC R	CAMERA_7	EBI NCS3	EBI-ADDR4
8	AC L	STBY1	EBI NCS0	EBI-ADDR5
9	AC L	IN11	EBI-ADDR23	EBI-ADDR6
10	VSYNC0	IN12	EBI-ADDR22	EBI-ADDR7
11	ADCREF	PWM1	EBI-ADDR21	EBI-ADDR8
12			EBI-ADDR20	EBI-ADDR9
13		PWM2		EBI-SDCK
14		IN22	EBI-SDCKE	EBI-ADDR10
15	RRST0	IN21	EBI-SDWE	EBI-ADDR11
16	ADCREF	STBY2	EBI-CAS	EBI-ADDR12
17	-		EBI-RAS	EBI-ADDR13
18	-		EBI-SDA10	EBI-ADDR14
19	RCLK0	SPI-MOSI	EBI-DATA0	EBI-ADDR15
20	WEN0	SPI-MISO	EBI-DATA1	EBI-ADDR16
21	WRST0	SPI-SCK	EBI-DATA2	EBI-ADDR17
22	RRST1	SPI-CS3	EBI-DATA3	EBI-ADDR18
23	RCLK1	SPI-CS2	EBI-DATA4	EBI-ADDR19
24	WEN1	SPI-CS1	EBI-DATA5	EBI-NWE1
25	WRST1	SPI-CS0	EBI-DATA6	EBI-NWE0
26	VSYNC1	SD- DETECT	EBI-DATA7	EBI-NRD
27	NOE1		EBI-DATA8	EBI NCS1
28	NOE0		EBI-DATA9	EBI NCS2
29			EBI-DATA10	
30	-	CLK	EBI-DATA11	EBI-NWAIT
31	-	CLK	EBI-DATA12	-

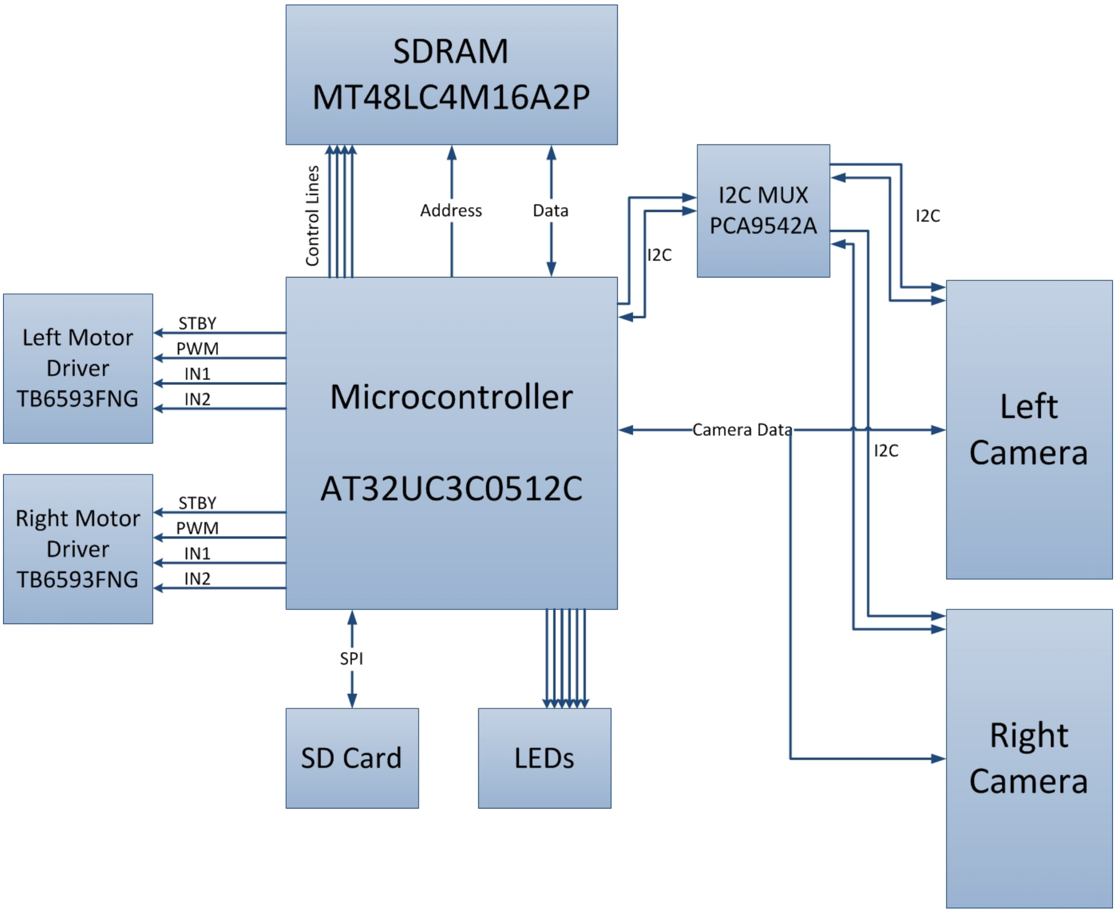


Figure 3.11: A hierarchical diagram of the robot

3.5.2 PCB Design

The PCB was designed using EAGLE CAD Software. A four layer board was decided to be used to reduce the number of tracks and more easily supply power and ground to the devices. Layer two is a 3.3V plane and layer three is a ground plane. A ground plane is also on the top and bottom layers to help eliminate any ground bounce that could occur.

The SDRAM uses the EBI protocol. In high speed systems, care is often taken to equalise track lengths (Liu and Lin, 2004). The UC3C maximum clock frequency is 33MHz (with no wait states), which is not fast enough to cause any track equalisation problems. Care, however, was taken on the USB lines to ensure correct impedance and the tracks lengths matched to each other.

Tracks were routed in order of priority, starting with the UC3C, SDRAM and cameras. All other devices were then routed (I^2C multiplexer, SD card, motor drivers etc). As a precaution, spare pins from the UC3C were routed to headers (J8 and J9, so that additions could be done if a pinout or connection was found to be incorrect. UART, I^2C and SPI connections were routed to headers J7, J4 and J5 respectively so logic analysers and a COM Port could be attached easily for debugging, or so that extra devices could be added onto the respective protocols for future developments.

Passives used were all surface mount of either 0603 or 1206 size to save space on the board. All headers used were 0.1" spaced for easy connections and a mini B USB socket was added to power the robot, program via the bootloader or so that the robot could potentially act as a USB device.

The layout of components was important. The cameras needed to be as far apart as possible and at the front of the PCB. The motor drivers were situated toward the back of the PCB and headers were added to connect the motors to. The optosensors were positioned such that they could be mounted directly on the PCB and be in the correct position to sense the wheels. Mounting holes were also added onto the board so the PCB could be mounted on to the robot base easily. The overall dimensions of the PCB were $100mm \times 70mm$. A full list of components and cost of each is documented in Appendix D

Finally, the name *"The Columbus"* was decided on as the original application for the project was a mapping robot that would search out an unknown area, so the robot was named after Christopher Columbus who explored and navigated parts of the American continents which were unknown at the time. The Eagle CAD Diagram of the PCB can be seen in Appendix C. The PCB was manufactured by PCB Cart (2013). The PCB cost £205 to manufacture and ship five PCBs. A photo of the PCB can be seen in figure 3.12.

Considerations - Power consumption of devices not exceeding VReg

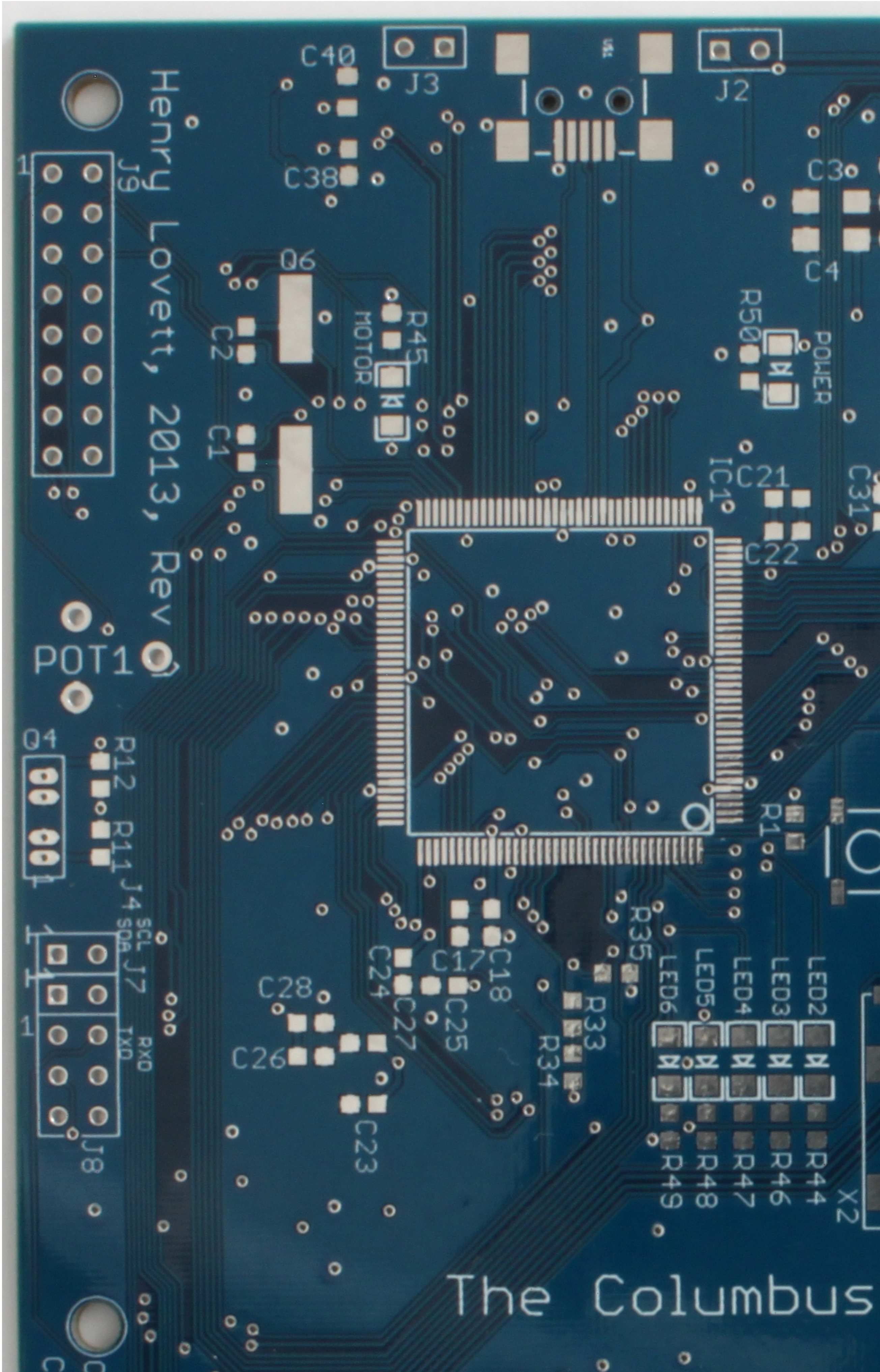
3.5.3 PCB Testing

A program was written to test all the devices on the PCB. The following tests are done and are explained in the subsequent sections:

3.5.3.1 UART Send and Receive

3.5.3.2 SD Card

3.5.3.3 LEDs



Henry Lovett, 2013, Rev
The Columbus
C40
J3
J2
C38
J9
Q6
C3
C4
R50
POWER
MOTOR
R45
C2
C1
IC1
C21
C22
C31
POT1
Q4
R12
R11
R1
J4
SCL
SDA
J7
RXD
TXD
J8
C17
C18
C24
C27
C25
C28
C26
C23
R35
R33
R34
LED6
LED5
LED4
LED3
LED2
R49
R48
R47
R46
R44
X2

3.5.3.4 SDRAM

3.5.3.5 I^2C

3.5.3.6 Camera

3.5.3.7 Motor

3.5.3.1 UART Test

When the test program begins, the microcontroller waits for a character input. All characters are echoed back. This enables the user to check the communications work. Once a carriage return key is received (D_{16}), the test program continues. Listing 3.1 shows the test code for the UART protocol.

Listing 3.1: UART Test Code

```
///////////////////////////////////////////////////
// UART Test //////////////////////////////////////
///////////////////////////////////////////////////
uint8_t ch;
while (true) {
  ch = usart_getchar(DBG_USART); // get one input character
  if (ch) {
    print_dbg(ch); // echo to output
  }
  if(ch == 13)
    break;
}
```

3.5.3.2 SD Card Test

The Atmel Software Framework (Atmel Corporation, 2009) provided drivers and code for SPI communications and use of a FAT32 file system. The code was configured to use the correct chip select pin for the SD Card and the correct SPI Bus. The test consisted of initialising the memory, reading the capacity of the card and printing it to the user.

The AVR then proceeds to delete any previous log file, create a new log file and writes *"Columbus Tester"* to it. The first 8 characters, which should be *"Columbus"* are read back and checked.

Listing 3.2: UART Test Code

```
//////////////////////////////////////////////////
// SD Card Test //////////////////////////////////
//////////////////////////////////////////////////
print_dbg("\n\n\rSD Card Memory Test:\n\r");
// Test if the memory is ready - using the control access memory abstraction
   layer (/SERVICES/MEMORY/CTRL_ACCESS/)
if (mem_test_unit_ready(LUN_ID_SD_MMC_SPI_MEM) == CTRL_GOOD)
{
  // Get and display the capacity
  mem_read_capacity(LUN_ID_SD_MMC_SPI_MEM, &VarTemp);
  print_dbg("OK:\t");
  //printf_ulong((VarTemp + 1) >> (20 - FS_SHIFT_B_TO_SECTOR));
  i = ((VarTemp + 1) >> (20 - FS_SHIFT_B_TO_SECTOR));
  print_dbg_ulong(i);
  print_dbg("MB\r\n");
  print_dbg("SD Card Okay.\n\r");
}
else
{
  // Display an error message
  print_dbg("Not initialized: Check if memory is ready...\r\n");
}
nav_reset();
// Use the last drive available as default.
nav_drive_set(nav_drive_nb() - 1);
// Mount it.
nav_partition_mount();
nav_filelist_reset();
if(nav_filelist_findname((FS_STRING)LOG_FILE, false))
{
  print_dbg("\n\rLog File Already Exists\n\rAttempting to delete...");
  nav_setcwd((FS_STRING)LOG_FILE, true, false);
  nav_file_del(false);

  if(nav_filelist_findname((FS_STRING)LOG_FILE, false))
    print_dbg("\n\rLog File Still Exists...");
  else
    print_dbg("\n\rLog File Deleted!");
}
print_dbg("\n\rCreating Log File.");

if(nav_file_create((FS_STRING)LOG_FILE) == true)
  print_dbg("\n\rSuccess!");
else
  print_dbg("\n\rNot worked...");
print_dbg("\n\rWriting to log file.");
Log_Write("Columbus Tester:\n\r", 18);
nav_filelist_reset();
nav_setcwd((FS_STRING)LOG_FILE, true, false);
file_open(FOPEN_MODE_R);//Open File
file_read_buf(Buffer, 8);
noErrors = 0;
if(Buffer[0] != 'C')
  noErrors++;
if(Buffer[1] != 'o')
  noErrors++;
if(Buffer[2] != 'l')
  noErrors++;
```

```
  if(Buffer[3] != 'u')
    noErrors++;
  if(Buffer[4] != 'm')
    noErrors++;
  if(Buffer[5] != 'b')
    noErrors++;
  if(Buffer[6] != 'u')
    noErrors++;
  if(Buffer[7] != 's')
    noErrors++;
  file_close();
  if(noErrors == 0)
    print_dbg("\n\rSD Card Read Successful\n\r");
  else
    print_dbg("\n\rSD Card Read Fail\n\r");
  noErrors = 0;
```

This exercises all basic file I/O functions, creating, reading and writing and checks that they work.

3.5.3.3 LED Test

All LEDs are turned on for 1 second, and then turned off. The user should check this occurs. It verifies that all the LEDs are functional and correctly mounted. The Power LED should be on when power is supplied to the PCB.

3.5.3.4 SDRAM Test

The SDRAM test consists of initialising the SDRAM, calculating the SDRAM size, writing a unique test pattern to the whole memory, and then reading it back and checking it. The total number of errors are reported.

The test was adapted from an example application from the ASF (Atmel Corporation, 2009). The code can be seen in listing 3.3. It consists of two *for* loops. In the first, the iteration number is assigned to the memory location. The second loop reads back the data and checks it is correct. An int, *noErrors*, is used to count errors.

Listing 3.3: SDRAM Test Code

```
  ///////////////////////////////////////////////////
  // SDRAM Test ////////////////////////////////////
  ///////////////////////////////////////////////////
  print_dbg("\n\n\rSDRAM Test:");
  sdram_size = SDRAM_SIZE >> 2;
```

```
  print_dbg("\n\rSDRAM size: ");
  print_dbg_ulong(SDRAM_SIZE >> 20);
  print_dbg(" MB\r\n");
  // Determine the increment of SDRAM word address requiring an update of the
  // printed progression status.
  progress_inc = (sdram_size + 50) / 100;
  // Fill the SDRAM with the test pattern.
  for (i = 0, j = 0; i < sdram_size; i++)
  {
    if (i == j * progress_inc)
    {
      print_dbg("\rFilling SDRAM with test pattern:");
      print_dbg_ulong(j++);
      print_dbg("%");
    }
    sdram[i] = i;

  }
  print_dbg("\rSDRAM filled with test pattern       \r\n");
  // Recover the test pattern from the SDRAM and verify it.
  for (i = 0, j = 0; i < sdram_size; i++)
  {
    if (i == j * progress_inc)
    {
      print_dbg("\rRecovering test pattern from SDRAM: ");
      print_dbg_ulong(j++);
      print_dbg("%");
    }
    tmp = sdram[i];
    if (tmp != i)//failed
    {
      noErrors++;
    }
  }
  print_dbg("\rSDRAM tested: ");
  print_dbg_ulong(noErrors);
  print_dbg(" corrupted word(s)       \r\n");
```

3.5.3.5 I^2C Test

The I^2C test checks the bus for devices. It prints out a table showing the address of any devices that acknowledges a probe. A probe is a set up to write to the address. If a device exists on the line, it should acknowledge (Phillips, 2012). The test is done three times, with no channel selected on the I^2C multiplexer, with channel 0 selected and with channel 1 selected. The two addresses expected at 21_{16} for the OV7670 Camera and 74_{16} for the I^2C multiplexer. The camera should only acknowledge when the I^2C multiplexer has the relevant channel selected. Listing 3.4 shows the test code for the I^2C bus and listing 3.5 shows the result from the full bus scan with channel 0 selected. The cameras are both checked to exist.

Listing 3.4: I^2C Test Code

```
/////////////////////////////////////////////////////
// TWI Test ////////////////////////////////////////
/////////////////////////////////////////////////////
print_dbg("\n\n\rTWI Test:\n\r");
Log_Write("\n\n\rTWI Test:\n\r", 14);
for(k = 0; tkmp < 3; k++)
{
  if(k == 0){
    print_dbg("Scanning all Channels\n\r");
  }
  else if (k == 1){
    //Channel 0
    PCA9542A_Chan_Sel(I2C_CHANNEL_0);
    print_dbg("\n\rScanning Channel 0\n\r");
  }
  else {
    //Channel 1
    PCA9542A_Chan_Sel(I2C_CHANNEL_1);
    print_dbg("\n\rScanning Channel 1\n\r");
  }

  print_dbg("h 0 1 2 3 4 5 6 7 8 9 A B C D E F\n\r");
  tmp = 0;
  for(i = 0; i < 8; i++)
  {
    print_dbg_ulong(i);
    print_dbg(" ");
    for(j = 0; j < 16; j++){
      int status = twim_probe(TWIM, tmp++);
      if(status == STATUS_OK){
        print_dbg("A");
      }
      else{
        print_dbg("-");
      }
      print_dbg(" ");
    }
    print_dbg("\n\r");
  }
}
noErrors = 0;
//Check cameras exist
PCA9542A_Chan_Sel(I2C_CHANNEL_0);
if(twim_probe(TWIM, 0x21) != STATUS_OK)
  print_dbg("\n\rCamera 0 Not Found;");
PCA9542A_Chan_Sel(I2C_CHANNEL_1);
if(twim_probe(TWIM, 0x21) != STATUS_OK)
  print_dbg("\n\rCamera 1 Not Found;");
```

Listing 3.5: Result of I^2C bus scan with Channel 0 of the I^2C multiplexer selected

```
Scanning Channel 0
h 0 1 2 3 4 5 6 7 8 9 A B C D E F
```

```
0 - - - - - - - - - - - - - - - - - -
1 - - - - - - - - - - - - - - - - - -
2 - A - - - - - - - - - - - - - - - -
3 - - - - - - - - - - - - - - - - - -
4 - - - - - - - - - - - - - - - - - -
5 - - - - - - - - - - - - - - - - - -
6 - - - - - - - - - - - - - - - - - -
7 - - - - A - - - - - - - - - - - - -
```

3.5.3.6 Camera Test

This test consists of initialising both cameras and checking it succeeds. Two photos are then taken and stored to the SD card. Success or failure of the methods is sent to the debug terminal. Two images should exist on the SD card from the two cameras. Listing 3.6 shows the code to conduct this test.

Listing 3.6: Camera Test Code

```
//////////////////////////////////////////////////
// Camera Test ///////////////////////////////////
//////////////////////////////////////////////////
print_dbg("\n\rInitialising Cameras");
OV7670_Init();
FIFO_Reset(CAMERA_LEFT | CAMERA_RIGHT);
if(STATUS_OK == OV7670_Status.Error)
{
  print_dbg("\n\rCamera Initialise Okay!");
}
else
  print_dbg("\n\rCamara Initialise Fail.");

print_dbg("\n\rTaking Photos");

TakePhoto(CAMERA_LEFT | CAMERA_RIGHT);
while(Photos_Ready() == false)
  ;

if(Store_Both_Images() == true)
  print_dbg("\n\rImages Stored Successfully!");
else
  print_dbg("\n\rImages Store Fail.");
```

3.5.3.7 Motor Driver Test

An extensive test of the motor driver is discussed in section 3.4.3. The test code in this application resets the motors so that they are aligned to a white tab on the wheel. This code can be seen in listing 3.7. The robot should move no further than 1cm to reach a white tab and the motors should drive forward. This test is useful here to ensure the motors are connected the correct way around and that the potentiometers are set to an appropriate level.

Listing 3.7: Motor Test Code

```
////////////////////////////////////////////////////
// Motor Test //////////////////////////////////////
////////////////////////////////////////////////////
print_dbg("\n\rMotor Testing:\n\rMotor Initialised");
Motor_Init();
Motors_Reset();//reset the motors to test them
while(Motors_Moving() == true)
  ;//wait for the motors to finish moving
```

3.5.4 PCB Faults

During the build and test of the PCB, a number of faults were found. Each is explained and the solution for the problem given.

3.5.4.1 TCRT1010 Footprint

The holes in the footprint for the optosensor were not large enough. This was a minor problem as the sensor was soldered in a surface mount style. This didn't affect the location of the sensor so had no other implications other than the connection being weaker than it should be.

3.5.4.2 SDRAM Footprint

The SDRAM footprint made was done exactly to the specification of the pad size and locations with no consideration for soldering. This meant the chip fits exactly on to the footprint and made soldering difficult as pads had to be preloaded with solder and the device's pins were heated and bound to the solder. It also put the device at risk as more heat had to be used that usually necessary. Figure 3.13 shows the SDRAM chip slightly offset against the footprint. It can be seen that there is no extra space on the pad to be able to easily solder the device. Though this made building difficult and increased soldering errors, it had no impact on the operation of the device.

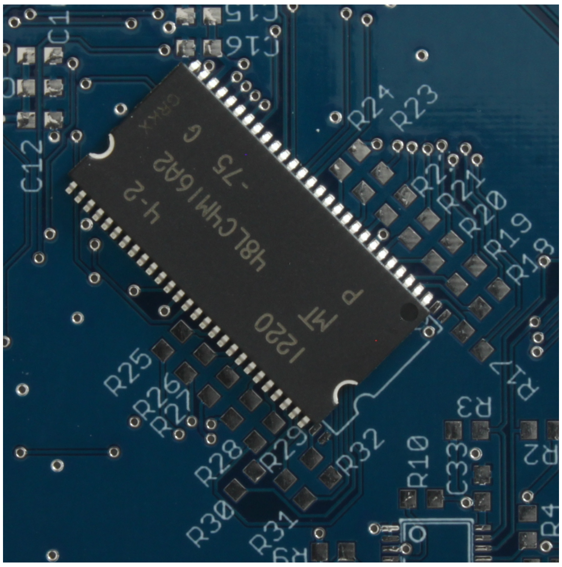


Figure 3.13: SDRAM Chip shown against its footprint.

To avoid this, existing footprints could be used from other libraries, or double checking the footprints made. The problem meant extra care during soldering had to be taken but has not impeded the operation of the device.

3.5.4.3 SDRAM Chip Select

The code was prototyped on the Atmel UC3C-EK development board prior to the PCB arriving. This used chip select 1 for the SDRAM and the PCB was designed using chip select 0. When the PCB was built, the code did not work. To diagnose this problem, the control lines of the SDRAM were probed with a logic analyser. On the UC3C-EK, the bus was busy with refresh cycles outside of SDRAM access. On the Columbus, no activity was seen.

The reason the correct control wasn't being seen was due to the UC3C device having a dedicated SDRAM controller, attached only to chip select line 1. Chip select 1 was available on an external pin, and the via on the CS0 and CS1 lines were close to each other. Therefore, to overcome the problem, a small enamelled wire was soldered to join the two vias together. This solved the problem and the correct signals were then seen on the control lines. The patch can be seen in figure 3.15(b).

This fault was caused by not reading the datasheet carefully and ignoring a proven circuit diagram. Operation of the device was not hindered and the fix was simple.

3.5.4.4 SDRAM Data Line Resistors

Once the chip select problem was solved, data returned was unreliable. The SDRAM is word (32 bit) addressed, but accessed in 16 bits. Each SDRAM access, therefore, consisted of two access cycles. Upon investigation of this problem, the 14th, 15th, 30th and 31st (top two bits of each 16 bit access) seemed to read as a 1 the majority of the time. This result wasn't repeatable and sometimes returned correct data. The other bits of the data were always correct. Table 3.7 shows some examples of the problematic data bits. The data written should match the data read back.

Table 3.7: A table showing examples of the incorrect data returned from the SDRAM

Data Written	Data Read
00000000 00000000 00000000 00000000	11000000 00000000 11000000 00000000
00001111 00001111 00001111 00001111	11001111 00001111 11001111 00001111

The problem was traced to resistors **R31** and **R32**. They were soldered on incorrectly so that the two data lines of the SDRAM were connected together and the two AVR GPIO pins were connected together. Data was then read back from, effectively, a high impedance line and therefore varied. Once the resistors were soldered correctly, the issue no longer persisted and the whole SDRAM test passed. By utilising the soldermask more, device orientations could be added to ensure correct placement. This can be extended to other devices, such as diodes and capacitors, especially in densely populated areas of the PCB.

3.5.4.5 Camera Interrupt Line

As discussed in section 3.1, the OV7670 needs an interrupt line to synchronise quickly to the start of the frame and is done by using an interrupt line. The UC3C has 9 external interrupt lines. On the PCB, interrupt lines 0 and 1 were used for this control.

Interrupt line 1 was easily configured and worked as expected. However, interrupt 0 did not seem to trigger the interrupt service routine. It was found that interrupt 0 was a "Non Maskable Interrupt" which has specific uses and cannot be used in to trigger a method.

The external interrupt 4 pin was wired to Junction 8 on the PCB. A wire was attached to the camera's VSYNC line and attached to the relevant pin on the

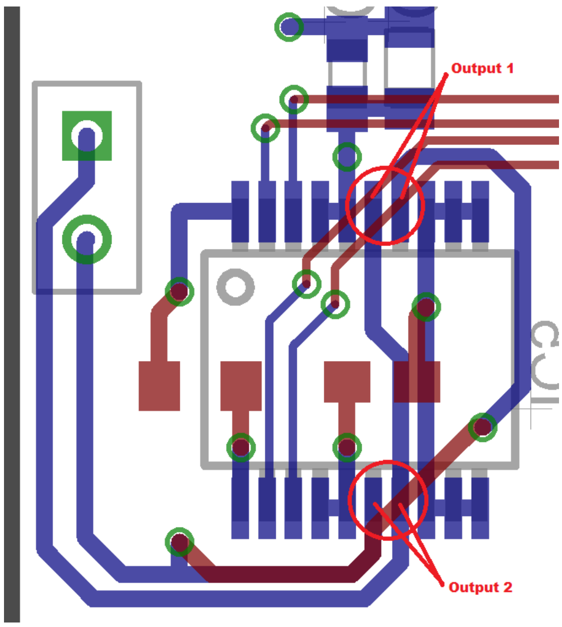


Figure 3.14: Motor Driver error. Outputs incorrectly connected

header. The operation was then easily obtained and the VSYNC line triggered correctly.

This issues would have been avoided with more understanding of the device before hand and checking the datasheet. The patch can also be seen in figure 3.15(b).

3.5.4.6 Motor Driver Footprint

An error was made in creating the device for the TB6593FNG Motor Driver in EAGLE. The chip has two outputs and each motor output has two pins to drive each pin of the motor. The pin assignment was mixed up when created and connected the two outputs together. Figure 3.14 shows the track errors on one of the motor drivers.

To solve this, pins 7 and 14 were lifted and removed so that output 1 and output 2 were not connected together. The devices were not damaged in the process of testing this and the motors functioned correctly after the modification. Double checking the footprints made against the datasheet would have avoided this problem. No impact to the operation of the drivers has been seen, but the patch may hinder the devices ability to sink current to the motors when driving at higher speeds.

3.5.5 PCB Conclusions

A number of faults were made in the PCB design. They are:

- TCRT1010 footprint
- SDRAM footprint
- SDRAM chip select line
- SDRAM data line resistors
- Camera interrupt line
- Motor driver footprint

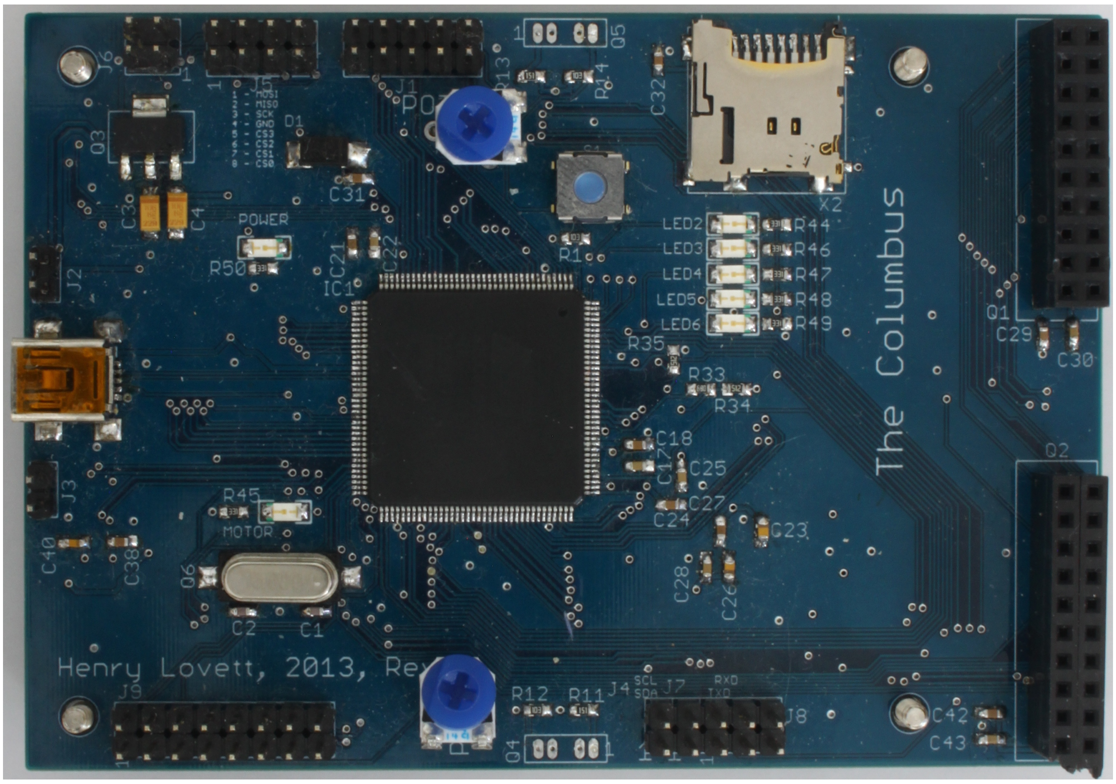
Three of the faults were due to footprint errors. Consulting the data sheets or using existing footprints would have avoided these problems. Two circuit errors were due to special operations of pins on the UC3C. More experience with this device could have prevented this but the errors were easily patches. By utilising the soldermask more would prevent errors during building.

For future PCBs, more care will be taken in circuit design, with prototyping of circuits with the hardware that will be used. This will highlight any pin specific operations (e.g. the non maskable interrupt) and reduce debugging post production. The effectiveness of a soldermask is also apparent, so more time spent on utilising this would be helpful during assembly.

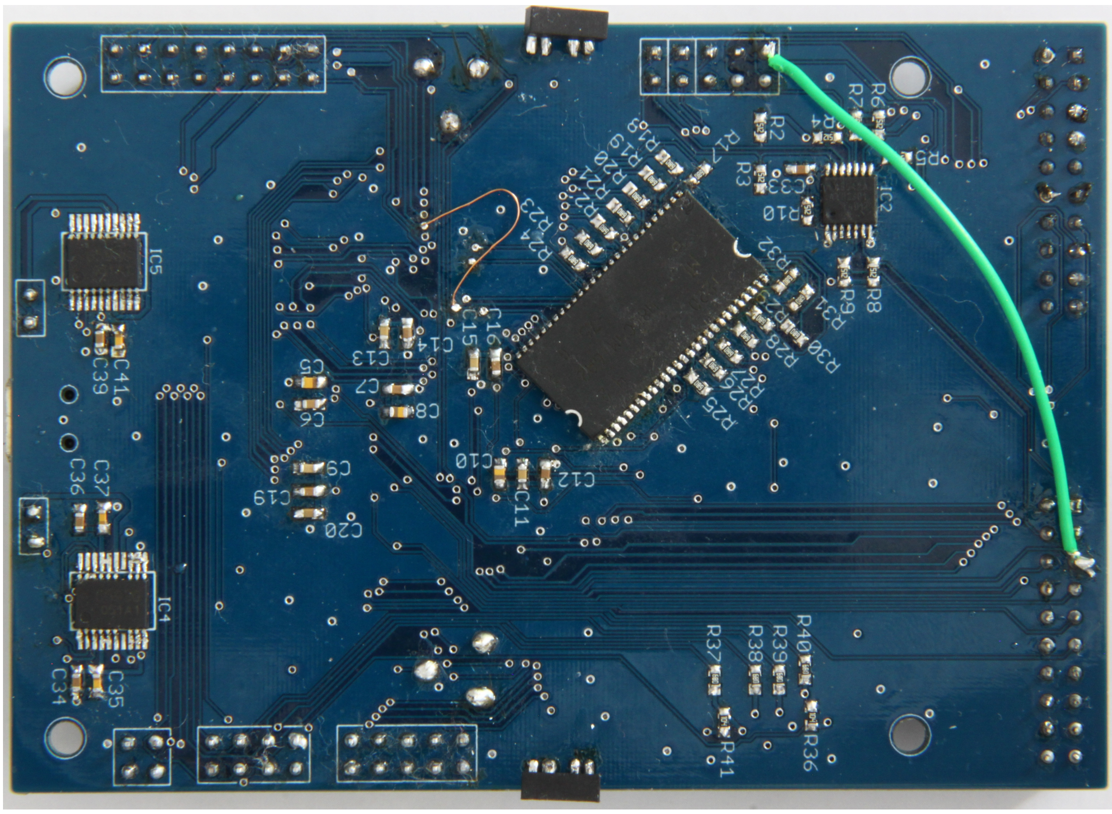
The PCB itself, was a success. It was a complex PCB with many potential things that could have gone wrong. All devices are functional (with a few small modifications) on the PCB so firmware development could continue with all hardware able to be used. Figure 3.15 shows the fully assembled PCB with the modifications needed.

3.6 Conclusions

Overall, the hardware design and firmware is a success. A few minor faults were apparent on the PCB, but these were easily patched and caused no problems. Using a firmware test, the components are seen to be fully functional giving the UC3C full ability to control motors, cameras, I^2C multiplexer, memory of both



(a) Top view of built PCB



(b) Bottom view of built PCB with SDRAM chip select patch

Figure 3.15: Pictures of the built PCB.

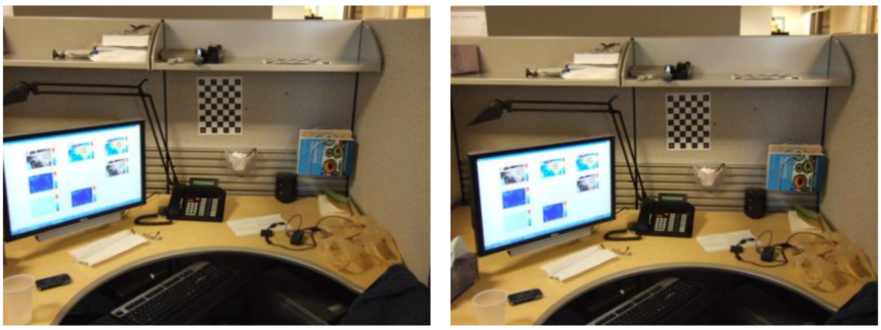
SD card (up to 2GB size) and external 4MB RAM. This provides a good platform for a manoeuvrable, stereoscopic vision robot to be developed.

Chapter 4

Vision Algorithms

4.1 Matching Algorithms

In computer vision, there are many different ways of comparing two similar images. These include the sum of absolute differences (S.A.D.) (Hamzah et al., 2010), the sum of squared differences (S.S.D.)(Mrovlje and Vrančić, 2008) and normalised cross correlation (N.C.C.)(Zhao et al., 2006). Each of these methods will be explained and tested to compare them. All testing will use images seen in figure 4.1. Each test uses the same size template (50×50) to compare the two images.



(a) Left Image (b) Right Image

Figure 4.1: Stereoscopic Test Images from MATLAB Examples

4.1.1 Sum of Absolute Differences

Given two identically sized two dimensional matrices, A, B, of dimensions I, J, SAD is defined as

$$SAD = \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} A[i,j] - B[i,j] \tag{4.1}$$

This method subtracts the observed template from the expected. All differences are then added together. This algorithm is simple and requires a small amount of computation. The algorithm returns values where a small result means the two images are well matched.

4.1.2 Sum of Squared Differences

$$SSD = \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} (A[i,j] - B[i,j])^2 \tag{4.2}$$

This is very similar to SAD but adds more complexity by squaring each difference. This removes the ability of equally different but opposite differences cancelling each other out (grey to white of one pixel will cancel out a white to grey difference in another with SAD). Again, a low result is a match in this case.

4.1.3 Normalised Cross Correlation

$$NCC = \frac{1}{n} \sum_{i,j} \frac{(A[i,j] - \bar{A}).(B[i,j] - \bar{B})}{\sigma_A.\sigma_B} \tag{4.3}$$

Where n is the number of pixels in A and B,
σ is the standard deviation of the image, and
$\bar{A}$ is the a average pixel value.

NCC is very similar to cross correlation, but normalised to reduce the error if one image is brighter than the other. This is common in computer vision (Tsai and Lin, 2003) and cross correlation is a often used in digital signal processing, so fast algorithms have been made to calculate this.

Unlike SSD and SAD, the normalised cross correlation gives a high value for a match. The downside to this algorithm comes with the complexity of the equation as it contains division and the a square root of a number in order to calculate the standard deviation. These operations are rarely implemented in hardware and are time consuming to carry out in software. They also require floating point registers and operates slowly on a microcontroller without any.

4.1.4 Comparison

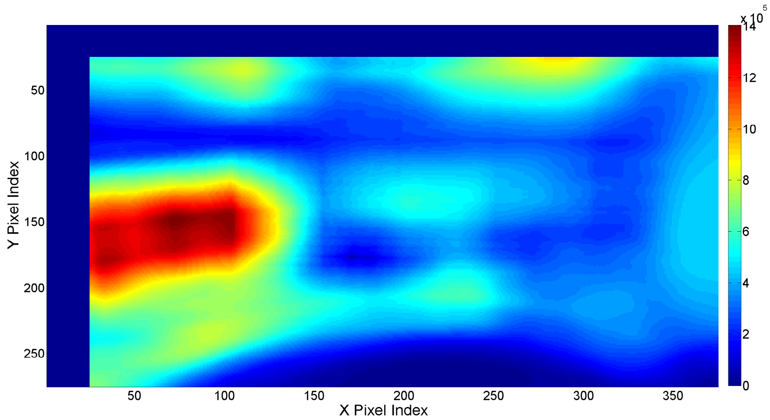
To compare these equations, a 50×50 template taken from the right picture was compared with the left image over the entire valid range. The coordinates on the graph give the centre pixel of the calculation.

Each graph shows the correct area being identified as a match, but this also highlights the downfalls of the SAD and SSD. The graphs in figure 4.2 are rotated to match the orientation of the images in figure 4.1. Each of the images is tested by attempting to match the desk phone from the right image to the entirety of the left image. The actual match should be around $(170, 176)$. An exact result cannot be estimated as the images are not matched perfectly - there isn't an exact integer of pixel difference between the images. This is the sub pixel problem (Haller and Nedevschi, 2012).

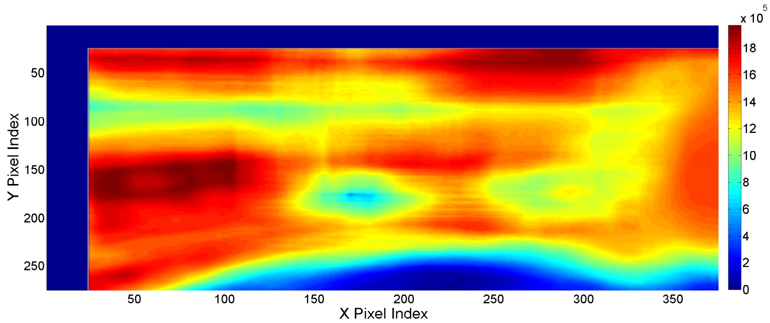
SAD results in figure 4.2(a) show large areas of matching. A minimum occurs around the location expected$(170, 175)$ of a value of 5.66×10^4. However, along the bottom of the image, where a dark area occurs below the desk in the lower part of figures 4.1, the SAD algorithm detects a greater comparison, with the lowest value in this area being 3370 at $(227, 275)$. This creates a false detection here.

SSD, figure 4.2(b), shows matches in the same two areas: where a match should occur and the dark area beneath the desk. The minimum value where the match should occur is 4.355×10^5 at location $(170, 176)$. However, there is a large match correlation between the dark area under the desk where the actual lowest value of 2.768×10^4 occurs at $(225, 274)$. This, again, is a false match and is a downfall of this algorithm.

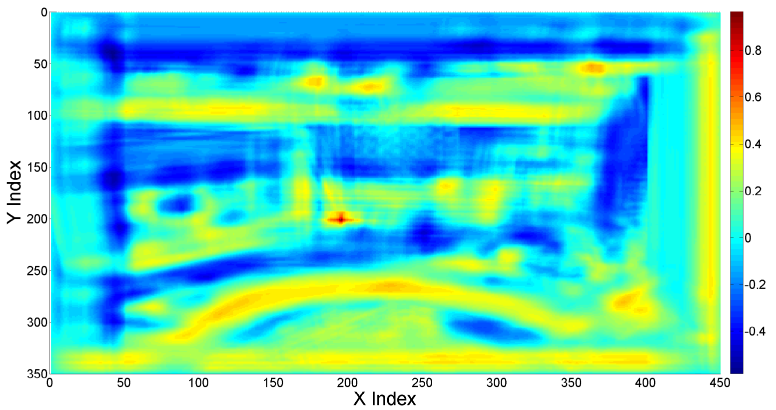
The NCC results are visible in figure 4.2(c). A match can be seen at coordinate $(195, 201)$ with a peak value of 0.9654. The coordinate is different to the previous results because the cross correlation works over the boundary of the image creating more results. The dimensions of the image are 300×400, but the NCC returns an data set of dimensions 350×450 when using a template size of 50×50. To get the actual match, half of the box size must be subtracted from the returned



(a) S.A.D Results (Low match)



(b) S.S.D. Results (Low match)



(c) N.C.C. Results (High match)

Figure 4.2: Result Graphs of Comparison Algorithms

coordinate. This means the match occurs at $(170, 176)$. With this algorithm, there is no area of the image which is close to a false detection.

4.1.5 Conclusion

It can be seen that there is a direct correlation between the complexity of the matching algorithm to the reliability of the match returned. In brightly lit, colourful environments absent of dark colours, SAD and SSD should provide a reliable result, but this cannot be guaranteed to always be the case. Therefore further development of the matching algorithm will start with using the normalised cross correlation. A comprise between complexity and reliability needs to be reached, where reliability is the more desirable of the two. Cross correlation is also widely used in digital signal processing, so optimised algorithms do exist.

4.2 Range Finding

4.2.1 Derivations

By using two images separated by a horizontal distance, B, the range of an object can be found given some characteristics of the camera. Appendix G contains the derivations for the follow scenarios:

1. Object is between the cameras (Figure G.1)
2. Object is in left or right hand sides of both images (Figure G.2)
3. Object is directly in front of a camera (Figure G.3)

4.2.2 Summary

There are three situations that can occur. These are listed below with their equations.

Object is between the two cameras:

$$D = \frac{Bx_0}{2\tan(\frac{\varphi_0}{2})(x_1 - x_2)} \tag{4.4}$$

Table 4.1: Table of results to calculate the field of view of the camera

L (mm)	D (mm)	$\varphi(^c)$
70	104	0.6435
90	135	0.6015
178	285	0.6054
214	345	0.6493
Average		0.6249

Object is to the same side in both images:

$$D = B.\frac{\cos(\varphi_2).\cos(\varphi_1)}{sin(\varphi_2 - \varphi_1)} \tag{4.5}$$

Object is directly in front of a camera:

$$D = B\tan\left(\frac{\pi}{2} - \varphi_2\right) \tag{4.6}$$

Where φ_1 is defined in equation (4.7) and φ_2 is defined in equation (4.8).

$$\varphi_1 = \arctan\left(\frac{2x_1}{x_0}\tan\left(\frac{\varphi_0}{2}\right)\right) \tag{4.7}$$

$$\varphi_2 = \arctan\left(\frac{2x_2}{x_0}\tan\left(\frac{\varphi_0}{2}\right)\right) \tag{4.8}$$

When the images have been matched, these equations can be used to calculate the range to an object.

4.2.3 Field of View

The field of view is an important characteristic to calculate distances and must be measured for the camera. Field of view was measured by placing a ruler at a distance in front of the camera and measuring the total distance seen across the image. Basic trigonometry was then used to calculate the field of view. This was done multiple times to confirm the measurement. Results can be seen in table 4.1. The field of view used is the average of the data set and was found to be $\varphi_0 = 0.6249^c$.

Figure to show this?

Table 4.2: Results of range finding test

Actual Distance (mm)	Pixel Difference in Images (pixel)	Calculated Distance (mm)	Error (%)
100	290	72	28
200	152	137	32
300	109	191	36
400	88	236	41
500	75	277	45
600	67	310	48
700	61	341	51
800	57	365	54
900	53	393	56
1000	51	408	59

4.2.4 Testing

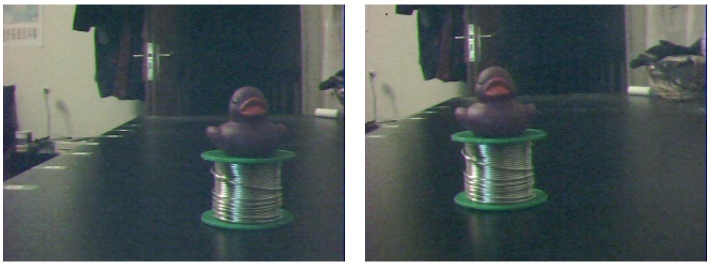
MATLAB was used to test the range finding, and the script can be found Appendix H. It is unable to automatically detect and object so the user must select the template when prompted.

A stereo pair of images of a rubber duck at different ranges were captured using the completed robot. To calculate the distance, a template from the right image of the ducks head was cross correlated with the left image. The maximum peak in the result was found and used as the match point. The distance was then calculated using equation (4.4), with $B = 42mm$, $\varphi_0 = 0.6249$ and $x_0 = 320$. Example images can be seen in figure 4.3 and an example NCC result can be seen in figure 4.4.

Table 4.2 shows the distances tested with the distance calculated using the above method. The ranges calculated were inaccurate. This could be down to the resolution of the camera being low, so one pixel change can potentially be a large distance error. Matching can only be done to the accuracy of a few pixels without extra effort and therefore can introduce a small error and cause a large distance error in the calculation.

4.2.5 Conclusion

Extensive testing of the effect of separation of the cameras has been done before (Mrovlje and Vrančić, 2008). Figure 4.5 shows that by increasing B will give you a larger range of detectable distances, but with a larger potential error with a low



(a) Left Image (b) Right Image

Figure 4.3: Stereo pair of images of a rubber duck on a reel of solder

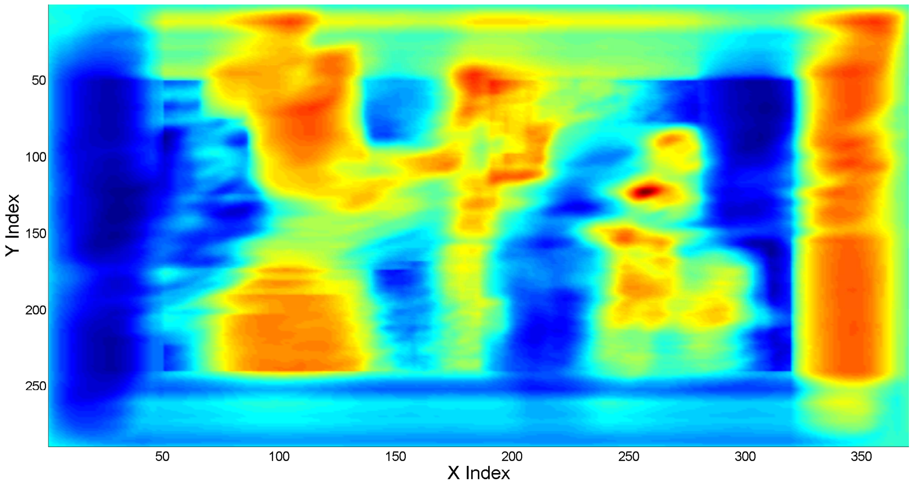


Figure 4.4: NCC results from matching using the ducks head from the right image as the template to the left image

resolution camera. A separation of more than 100 pixels will result in around the same calculated distance due to the function being of the form $D = kx^{-1}$. This gives a range of distances that the can be detected. For $B = 42mm$, it is around $200mm$ to $1m$. An external frame could be made to mount the cameras with a greater separation distance to alter this characteristic on the robot.

The robot was designed to be small, and the cameras were chosen as a cheap alternative to more expensive products on the market. The test results show that using the cameras with a small separation and low resolution, the system is unable to accurately measure distances to objects seen. The separation of objects in the images is a reciprocal function and the data gathered in the test matches this characteristic, see figure 4.6. This means the system can perceive depth from the separation of the objects, but not accurately calculate the distance.

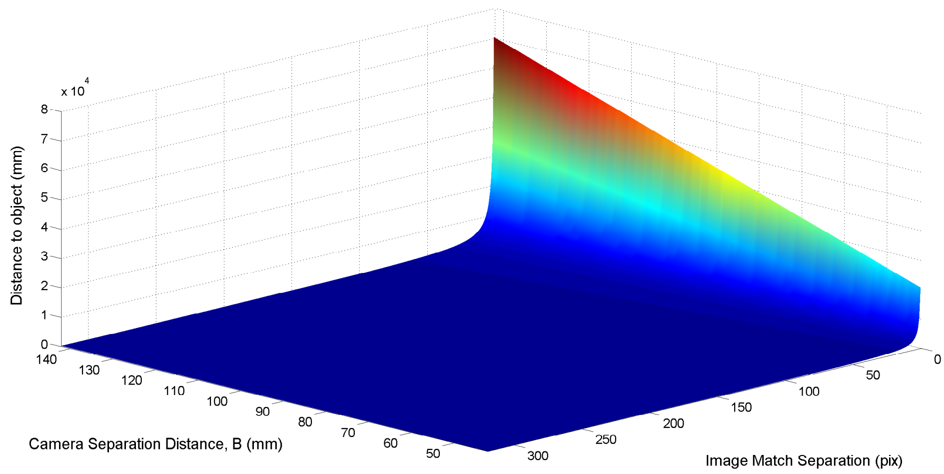


Figure 4.5: 3D Surface plot graph showing range of distances that can be calculated over a range of camera separation, B, values.

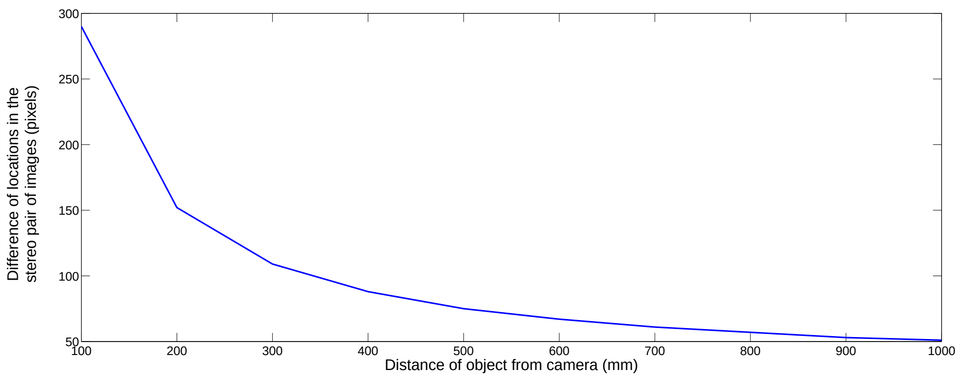
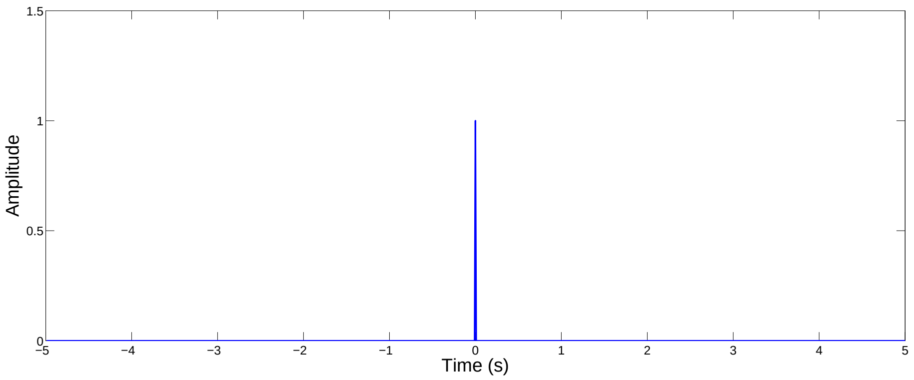


Figure 4.6: Graph showing Distance of the object against the difference in the match locations.

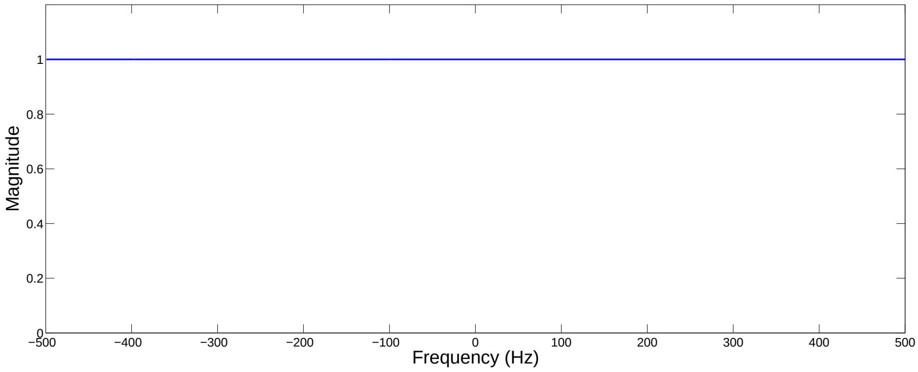
4.3 Fourier Transform

4.3.1 Background Research and the FFT

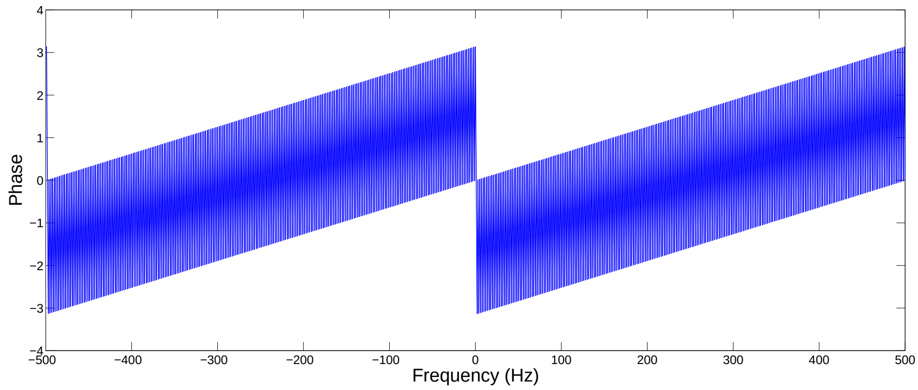
The Fourier transform is a common tool in signal processing. It transforms a time based signal to the frequency domain showing the frequency components contained in the signal as a complex number. This is often displayed as magnitude and phase. The Fourier Transform is defined in equation (4.9) and two examples of signals and their Fourier Transforms are shown in figures 4.7 and 4.8.



(a) A graph showing a Dirac Function



(b) A graph showing the magnitude of the Fourier transform of the Dirac Function

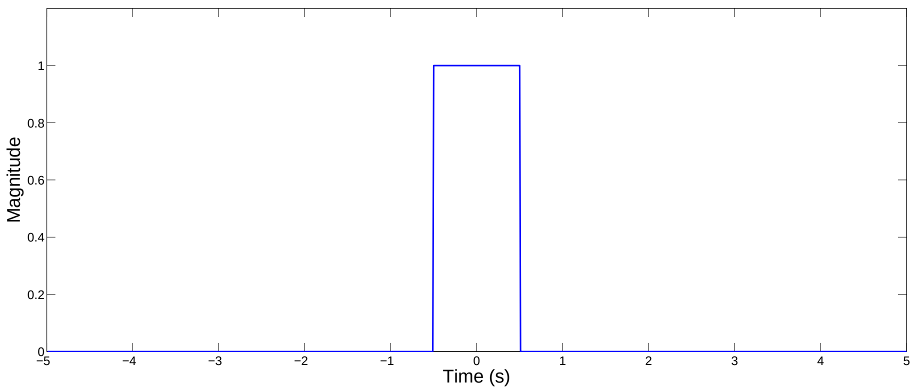


(c) A graph showing the phase of the Fourier transform of the Dirac Function

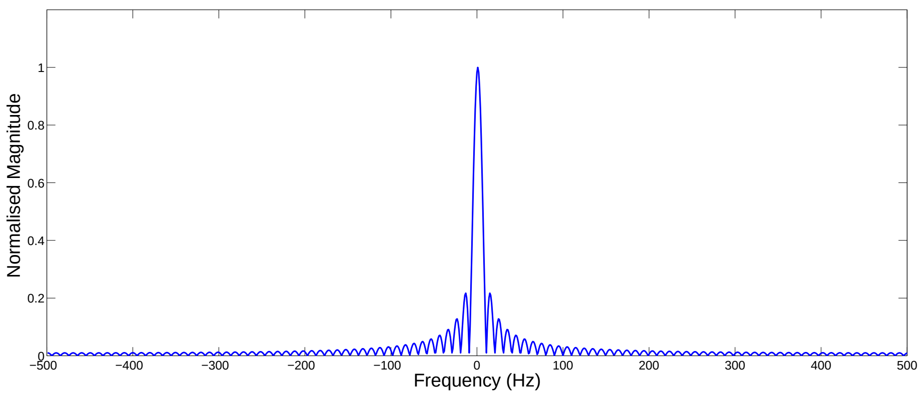
Figure 4.7: A Dirac signal and the phase and magnitude of its Fourier Transform

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-\jmath 2\pi f t} dt \tag{4.9}$$

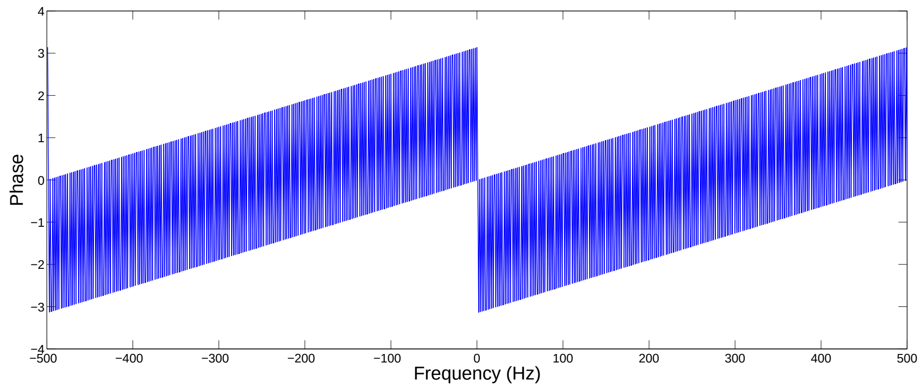
The Fourier transform is used in many areas of signal processing. It is used for



(a) A graph showing rectangular pulse



(b) A graph showing the magnitude of the Fourier transform of the rectangular pulse



(c) A graph showing the phase of the Fourier transform of the rectangular pulse

Figure 4.8: A 2D Rectangular pulse and the phase and magnitude of its Fourier Transform

filter design, system analysis and image processing as well as many others. It shows the frequency components of a time domain signal.

The equation for the Fourier transform in equation (4.9) is for continuous time. A discrete Fourier transform (DFT) exists for finite, equally spaced samples. This is commonly used in digital systems and is defined in equation (4.10). There exists a Fast Fourier Transform (FFT) which gives exactly the same results as the DFT, but is optimised in terms of number of multiplications done. The FFT will be used in implementation due to availability of code and speed of use.

$$X[k] = \sum_{0}^{N-1} x[n]e^{-\jmath\Omega_0 kn} \tag{4.10}$$

Where Ω_0 is the sample frequency

A property of the Fourier Transform of interest is the convolution theorem which states that convolution in time is multiplication in frequency and is defined mathematically in equation (4.11). The cross correlation is very similar to convolution. Cross correlation is defined in equation (4.12) and related to convolution. With images, $f(t)$ is a real signal, its conjugate is exactly the same, $f(t) \equiv f^*(t)$ given that$f(t) \in \Re$. This means that to compute a cross correlation, the Fourier transform of the image and the reversed template can be used and multiplied together. FFTs are quick and widely available so this should make the implementation more simple and fast.

$$\int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau = f(t) * g(t) = X(f) \cdot Y(f) \tag{4.11}$$

$$\int_{-\infty}^{\infty} f^*(\tau)g(t+\tau)d\tau = f(t) \star g(t) = f'(-t) * g(t) = X(-f) \cdot Y(f) \tag{4.12}$$

4.3.2 Two Dimensional Fast Fourier Transform

A two dimensional Fourier transform exists for analysing two dimensional signals, namely in this application, an image. The Fourier Transform is shown in equation (4.13) and the discrete version is shown in (4.14)

$$F(u,v) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-2\pi \jmath(xu+yv)}dxdy \tag{4.13}$$

$$F(u,v) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)e^{-\frac{2\pi \jmath(xu+yv)}{N}} \qquad x,y,u,v \in \{0\ldots N-1\} \tag{4.14}$$

Figures 4.9 and 4.10 show the two dimensional equivalent test signals of figures 4.7(a) and 4.8(a) and the phase and magnitudes of their Fourier Transforms. There is a direct similarity between the 1D and 2D spectra; the magnitudes of the Dirac (figures 4.7(b) and 4.9(b)) are both constant values and the rectangular pulses both have a modulus sinc function magnitude (figures 4.8(a) and 4.10(a)).

The 2D Fourier transform can also be optimised to a fast Fourier transform algorithm in a similar way as the 1D case. The algorithm, sometimes referred to as the Butterfly transform, is briefly discussed in Nixon and Aguado (2012) where it is explained that the algorithm can be easily applied to images with equal dimensions that are a power of 2. The algorithm utilises the separability property of the Fourier transform.

The 2D FFT can be implemented using a 1D FFT as follows:

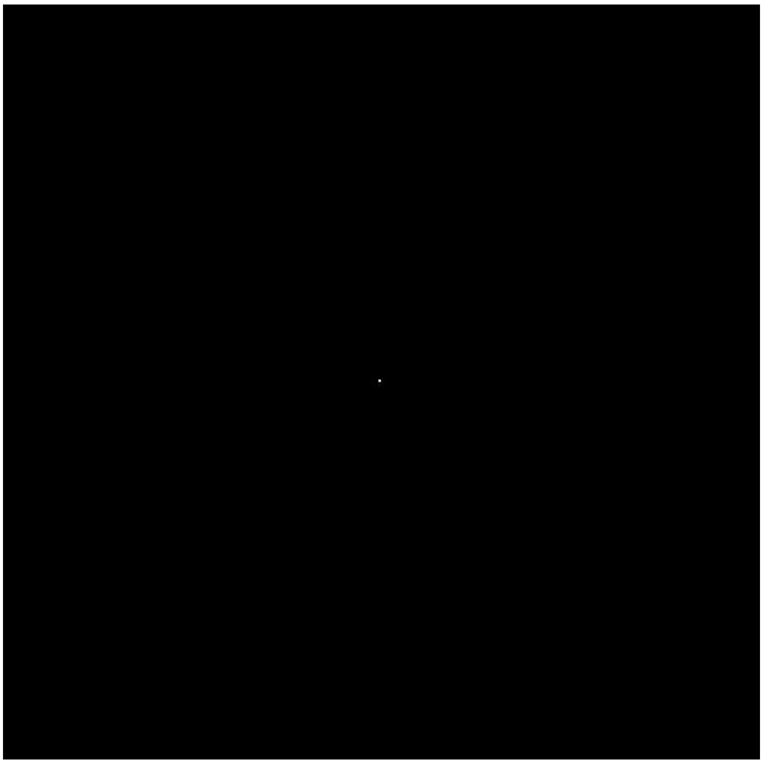
1. Calculate the 1D FFT of each of the rows of the 2D data. (An FFT of data of length n returns an array, also of length n)
2. Calculate the 1D FFT of each of the columns of the 2D data returned from the previous step.

Total number of FFTs done is $2n$ where n is the height/width of the image.

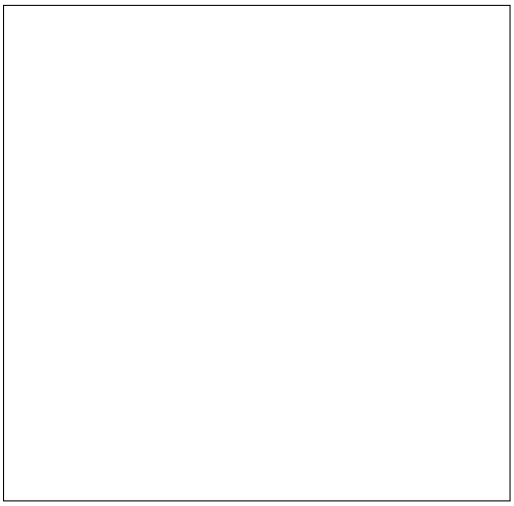
Maybe make a figure to help explain?

4.3.3 Implementing the FFT

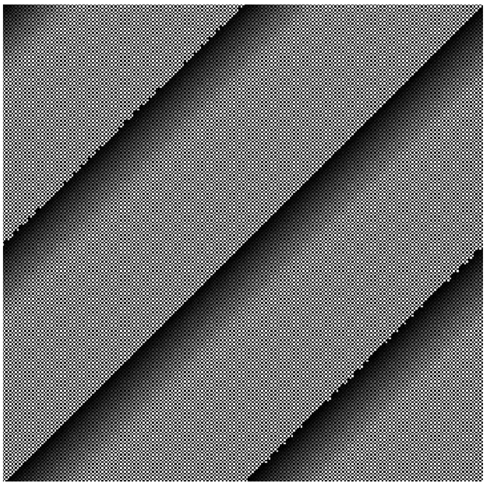
The Atmel Software Framework (Atmel Corporation, 2009) included a digital signal processing library. This contained functions to compute the FFT of a real or complex array, the inverse FFT and the magnitude of complex data. Further restrictions are imposed by the DSP library used as the data must be an even power of 2, and that the data is in fixed point notation. This gives a usable dimension of



(a) An image of a 2D Dirac Function

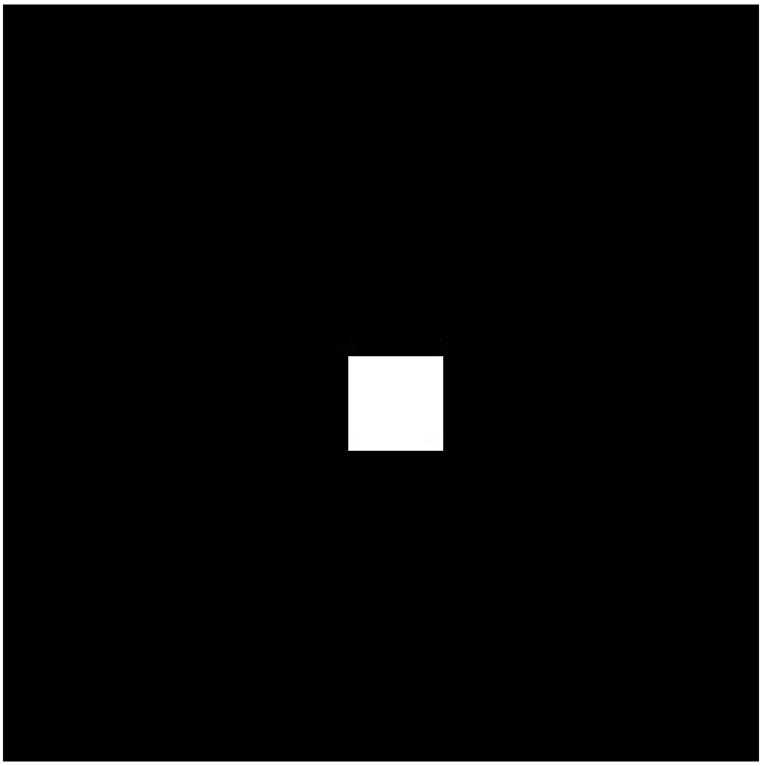


(b) An image of the magnitude of the Fourier transform of the 2D Dirac Function

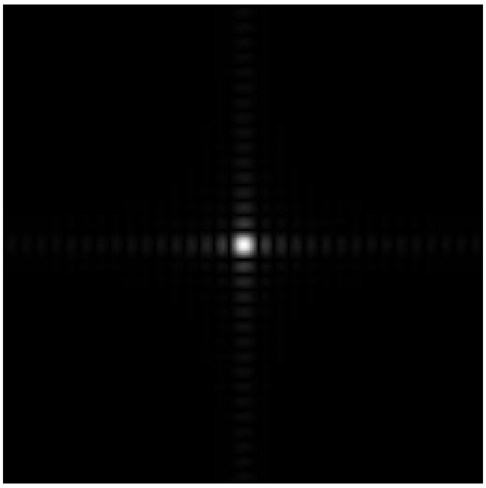


(c) An image of the phase of the Fourier transform of the 2D Dirac Function

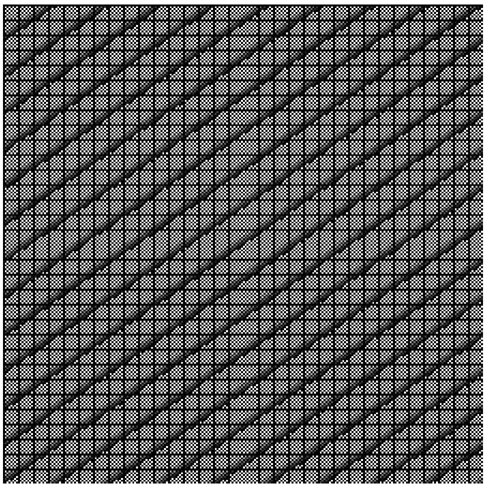
Figure 4.9: A 2D Dirac signal and the phase and magnitude of its Fourier Transform



(a) An image of the 2D rectangular pulse



(b) An image of the magnitude of the Fourier transform of the 2D rectangular pulse



(c) An image of the phase of the Fourier transform of the 2D rectangular pulse

Figure 4.10: A 2D Rectangular Pulse signal and the phase and magnitude of its Fourier transform

256×256 for processing images on the AVR. Though the height of an image from the OV7670 camera is 240 pixels, the image can be transformed so that it repeats for 16 rows at the bottom as the Fourier transform works on an assumption of the data being periodic.

The function *FFT2DCOMPLEX* in Appendix H.1.1.5 is the realisation of a two dimensional fast Fourier transform on the microcontroller. The FFT function requires the data to be 4 byte aligned (A_ALIGNED) and of type *dsp16_complex_t*. The data must be given in fixed point notation and it is returned in fixed point notation. A 16 bit representation was chosen over 32 bit due to being more functions for 16 bit data available.

4.3.4 Testing of the FFT on AVR

4.3.4.1 1D FFT Test

A Dirac function and a rectangular pulse were used as test signals.

Figure 4.11 shows the input signal given to the AVR. It is a 256 long array of a Dirac function. This was then converted to the internally defined fixed point notation and passed through the Fourier transform method. The resulting complex array was then saved to a Comma Separated Value file and read into MATLAB. Figure 4.12 shows the calculated phase and magnitude plots of the output complex array. The magnitude is relatively flat and around the value of 1. In comparison with figure 4.7(b), they are relatively similar. The phase, however, seems to be very different. Figure 4.7(c) shows what was expected, but the two phase results appear to be quite different. This could due to MATLAB having more accurate algorithms and a more accurate representation that the 16 bit fixed point used on the AVR. However, using a function in the DSP library to calculate the magnitude, the spectrum in figure 4.13 is obtained. This, though is not exactly 1 as expected, is completely flat and it is computed from the same transformed data. It suggests that there is some internal compensation in the algorithms. The actual value in figure 4.13 is 0.9897 to 4 decimal places giving an overall error of 1.03%.

Figures 4.14, 4.15 and 4.16 show the similar outputs from the AVR when transforming the rectangular pulse. The result was renormalised from fixed point notation and the data was shifted so that the centre of the plot is frequency 0. Again, it can be seen the magnitude calculated from the complex output (figure 4.15) is different to the result when the magnitude is calculated on the AVR (figure

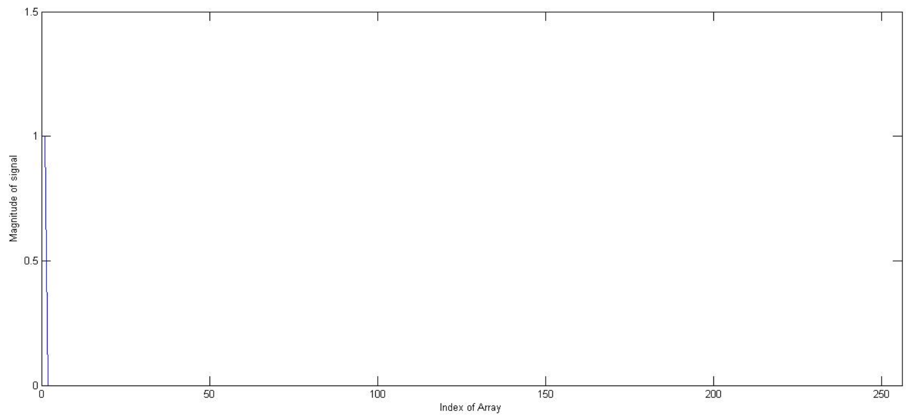


Figure 4.11: Input Dirac Signal for AVR fast Fourier transform

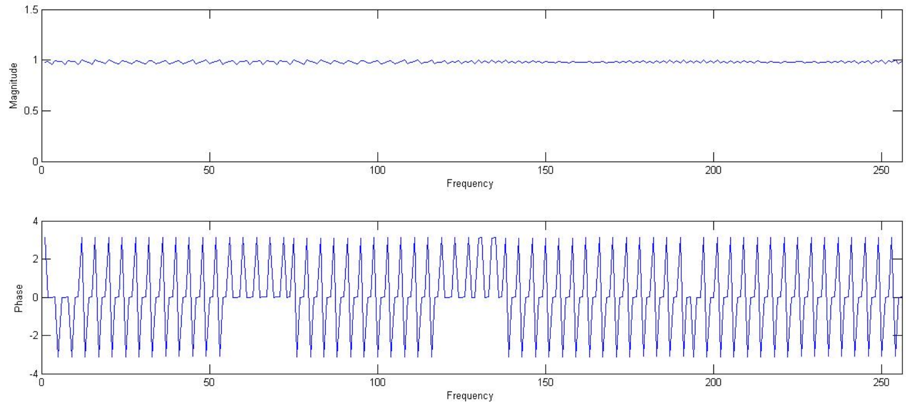


Figure 4.12: Output phase and magnitude of the complex output from AVR fast Fourier transform of a Dirac function

4.16). There are also differences in the result from the AVR and the result from MATLAB in figure 4.8, which can, again, be put down to the algorithms.

4.3.4.2 2D FFT Test

Two test signals were used to test the 2D FFT on the AVR, a Dirac signal and a square wave, seen in figures 4.9(a) and 4.10(a). The internal method on the AVR was not able to compute the magnitude due to the method scaling all the values down causing truncation errors. The complex Fourier transform was obtained, saved to the SD card in CSV format and viewed in MATLAB. All data was normalised to omit the effects of the fixed point notation and the data shifted so that frequency 0 is in the centre. These test were done with a 64×64 2D data

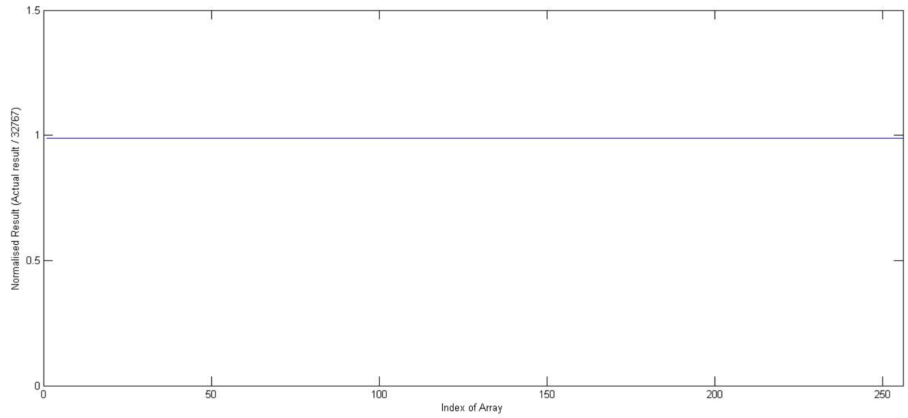


Figure 4.13: Magnitude calculated by the AVR of the Fourier transform of a Dirac function

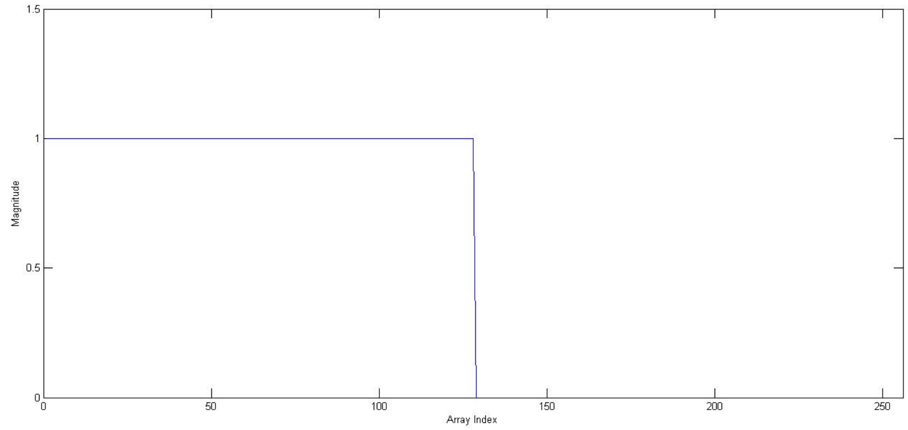


Figure 4.14: Input Rectangular Pulse for AVR fast Fourier transform

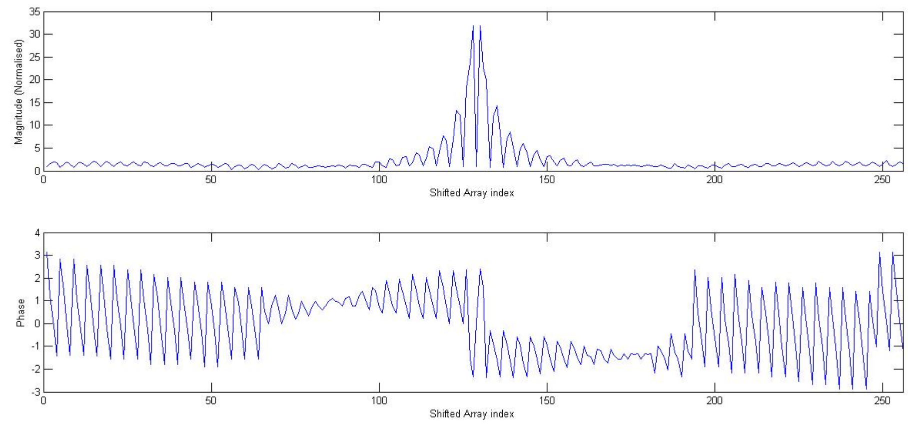


Figure 4.15: Output phase and magnitude of the complex output from AVR fast Fourier transform of a Rectangular Pulse

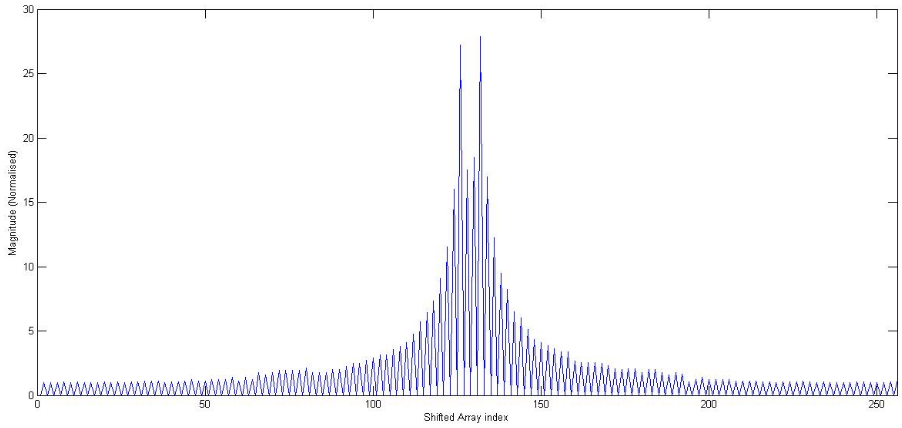


Figure 4.16: Magnitude calculated by the AVR of the Fourier transform of a Rectangular Pulse

as with a 256×256 array, the AVR runs out of internal RAM to calculate the transform.

I would like to get this to work by the end but not vital.

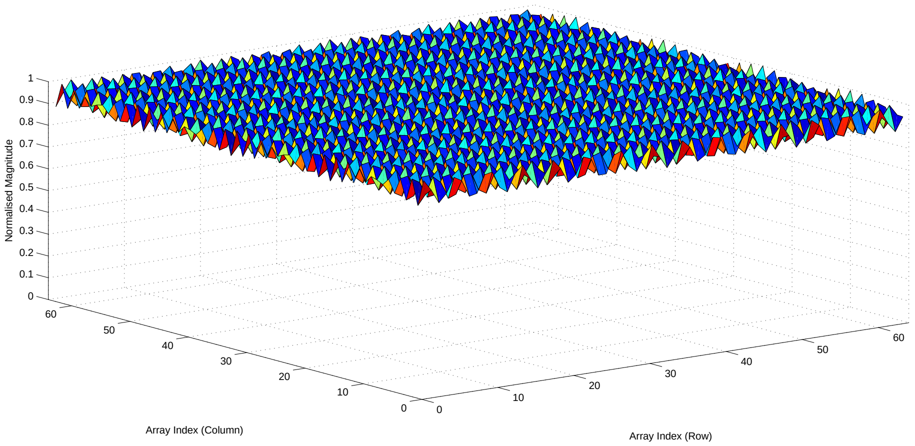
The result of the Dirac test is seen in figure 4.17. A similar error is found in the magnitude, but the spectrum is generally flat with a small amount of ripple as seen with the 1D FFT in figure 4.13. The phase has similar issues as with the 1D FFT. However, there appears to be the correct pattern with the 2D phase, but rotated about 45°. This is also the case with the square wave test. The magnitude in figure 4.18(a) is very similar, with a distinct peak in the centre (frequency 0) and a sinc function extending vertically and horizontally from this. Again, however, the phase (figure 4.18(b)) seems to differ a lot from the expected result in figure 4.10(c).

4.3.5 Conclusion

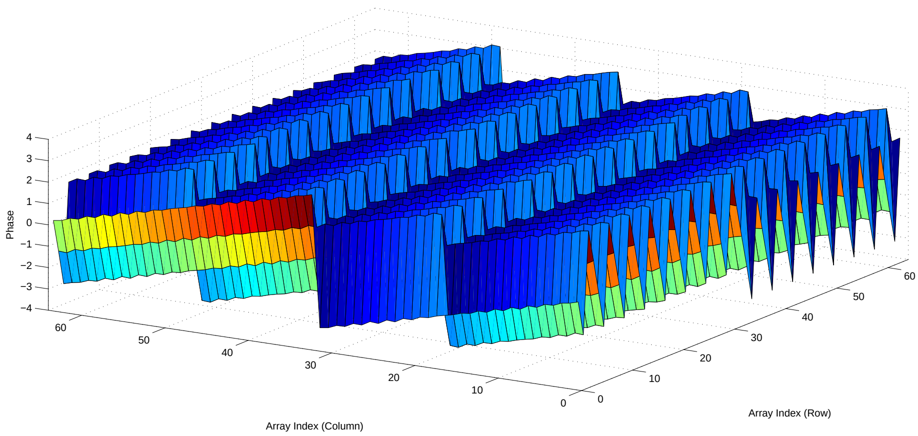
The transforms are calculated in real time with a 16MHz clock source. Table 4.3 shows the number of clock cycles taken to calculate the relevant sized transform. A 64×64 transform takes $39ms$ to compute with a 16MHz clock. This could be reduced by increasing the internal clock speed by using the PLLs available on the AVR, potentially taking it up to 33MHz and therefore halving the time to compute. A 64×64 image, however, is not practical for the application. Larger transforms can be done, but more effort needs to be taken and the external RAM could be utilised more effectively.

Table 4.3: Number of clock cycles taken to calculate the Transform of 64 or 256 long data set

	Size of FFT Data	
	64	256
1D	5019	23599
2D	618168	-

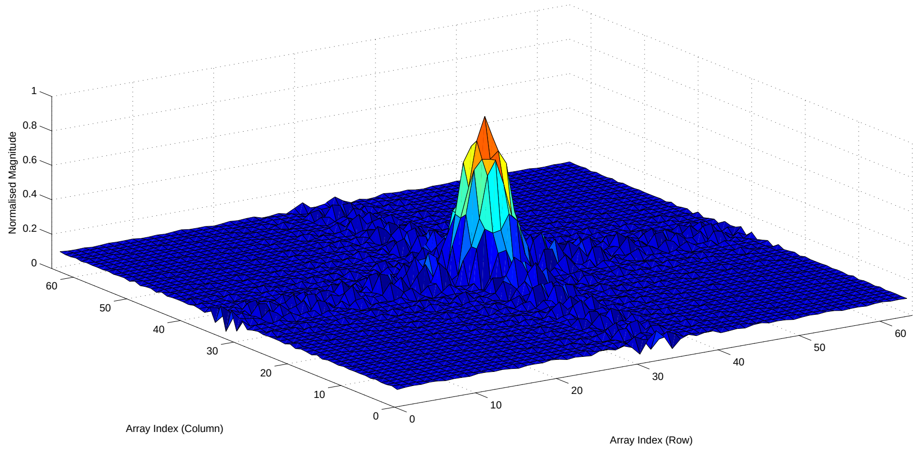


(a) A 3D Plot of the normalised magnitude of the complex data returned by the 2D fast Fourier Transform on the AVR

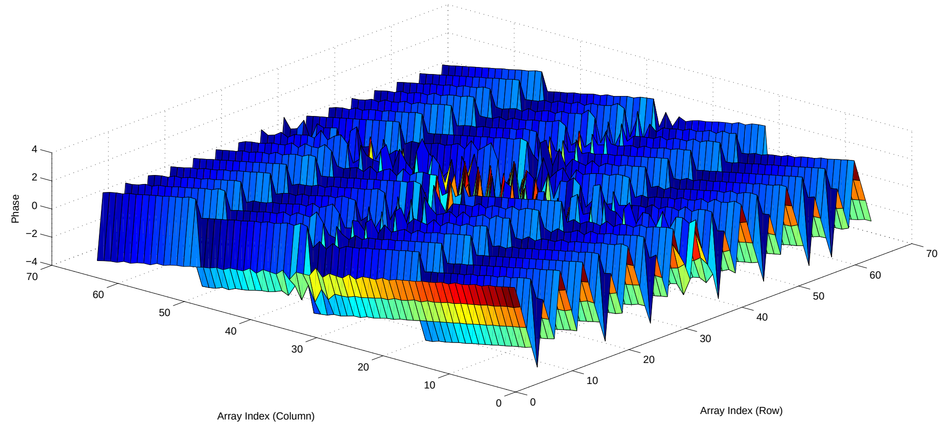


(b) A 3D Plot of the phase of the complex data returned by the 2D fast Fourier Transform on the AVR

Figure 4.17: 3D Plots of the phase and magnitude of the Complex Data returned from the 2D FFT on the AVR of a 2D Dirac Function



(a) A 3D Plot of the normalised magnitude of the complex data returned by the 2D fast Fourier Transform on the AVR



(b) A 3D Plot of the phase of the complex data returned by the 2D fast Fourier Transform on the AVR

Figure 4.18: 3D Plots of the phase and magnitude of the Complex Data returned from the 2D FFT on the AVR of a 2D Square Function

Show results of actual photo being transformed, need 2D 256 FFT working before

Maybe IFFT it too to find total errors in algorithm?

Chapter 5

Conclusions and Further Work

This work has led to a tested device which is mobile and has the capability to perform stereoscopic image processing. The system has the following parts:

1. Motor driving
2. Stereoscopic Cameras
3. SD Card memory
4. SDRAM
5. Image Processing

The motor system is a simple, cheap method to move distances with reasonable accuracy. A better controller would allow variable speed and speed matching between motors. The system was shown to work to 4.5% accuracy over a 300mm distance.

A stereo image pairs can be read captured by the cameras and stored on an SD card in a FAT32 file system. This gives the ability of removable memory that can be viewed on a computer to see any internal logs. Images are stored in QVGA format (320 by 240 pixels) as a bitmap image.

An additional 4MB of SDRAM memory is available to use on the robot allowing for large data arrays of the images to be kept in fast access RAM. The RAM is direct memory accessed so operation is almost seamless from internal memory.

Multiple comparison algorithms have been investigated and compared using the same test images. It was clear that, although at a necessity of more computations, the normalised cross correlation is the best with regard to overall reliability.

Range finding equations were then researched and derived, which use the characteristics of the camera and the separation distance between them to calculate distance to objects in view. The range finding capability was tested using MATLAB and it found that the system could not accurately calculate distances to objects. However, depth perception is possible even with low resolution cameras and small separation between them.

The Fourier transform was also investigated and implemented. The system allows for a 2D array of a square image with dimensions of 2^{2n} , where $n \in \mathbb{N}$, and is limited by RAM space and time. The transform is speed optimised and in tests proved to be fairly accurate.

All aspects implemented on the robot have been shown to be functional. A faster processor would have been a good idea to use for image processing, but this could have developed other problems with the PCB. The Raspberry Pi or Steve Gunn's *'L'Imperatrice'*, which both run a Linux operating system, would have been a good choice to remove the need for as much hardware design and existing image processing libraries could have been utilised to gain more functionality in the time.

Though the robots original application wasn't achieved, the end device is a base for a stereoscopic application. The system is a tested platform for future applications to be implemented on and additions can be made using the spare pins and bus connections on the PCB headers.

The system could be used in future projects to develop more functionality. Wireless communications could be added to the system to allow a connection to the computer and search algorithms can be implemented alongside distance calculations to make the robot aware of its surroundings.

5.1 System Operation

The system uses a debug USART available on J7 (57600bps, 8 data, 0 parity and 1 stop bit).

The system has two modes of operation, 'Auto Run' and Debug. By default, the Auto Run mode will run. Debug mode can be entered by the relevant command

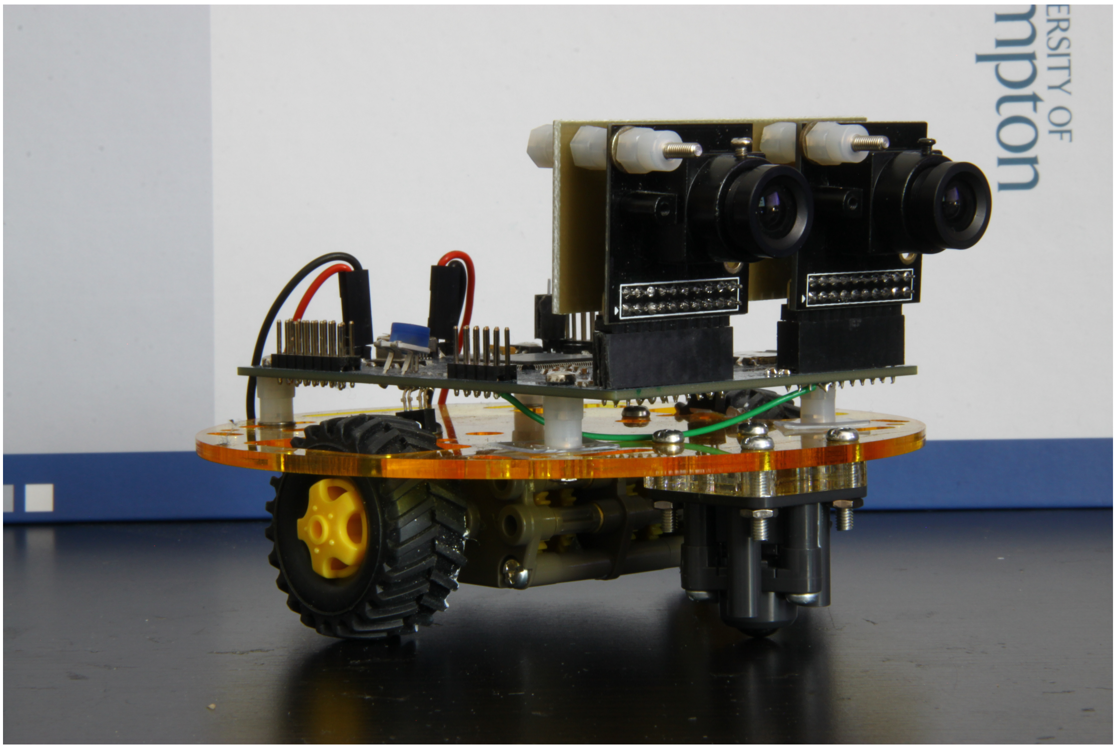


Figure 5.1: The completed Robot

in the Auto Run procedure (see table 5.1) or by connecting Pin D23, available at Pin 1 of J9, to ground on system start. The state of the robot is shown by the LEDs. Table 5.3 shows the meaning of the LEDs.

Auto Run mode runs a set of commands located in *"AutoRun.txt"* on the SD Card. If this file is not present, the system will run a default procedure defined in the code. A list of commands that can be run from this mode can be seen in table 5.1. The commands are specified by line If an invalid command is found, the system will exit and run the *System_Error* loop. The *System_Error* loop prints the status of all devices out once a second. By attaching a USART terminal, the error can be found.

Debug mode is a DOS-shell style terminal allowing the user to access methods and variables. This was used for development and debugging. A full list of commands can be seen in table 5.2.

The final robot can be seen in figure 5.1.

Table 5.1: Table showing the Auto Run Commands Implemented

Command	Argument	Operation
B	int	Move backward by the argument value (millimetres)
F	int	Move forward by the argument value (millimetres)
J	int	Jumps to the command specified (0 indexed)
P	N/A	Takes a stereo pair of photos
q	N/A	Quits Auto Run and enters debug mode
R	int	Rotates by argument (degrees)

Table 5.2: Table showing the available Debug Commands

Command	Argument	Operation
?		Shows the help prompt
A		Runs the Auto Run procedure in debug mode
B		Reads a Bitmap file and prints information
c		converts the working buffer from integer to fixed point
C		Converts the working buffer from fixed point to integer
d		Saves the Working Buffer to "Buffer_results.csv"
D		Frees the Memory pointed to by the Working Buffer
f		Reads "Buffer.csv" as a 2D Array of FFT_SIZE by FFT_SIZE
g		Saves the Complex Buffer to "Buffer_Complex.csv"
k		Prints the Complex Buffer
m		Computes the Magnitude of the 1D FFT of the Working Buffer
M F	(int)	Drive Robot forward by (int) millimetres (negative number for reverse)
M L		Dive Left Wheel Forward a full rotation
M q		Resets Motors
M R		Drive Right Wheel Forward a full rotation
M T	(int)	Rotate Robot by (int) degrees (positive turns Clockwise)
o		Displays the fixed point value for (int)1
P		Takes and stores Stereo Photos
r		displays the contents of the working buffer
R		Reads contents of "signal.bin", representing 1D Signal. Integers, Big Endian
T		Reads contents of "signal2d.bin", representing 2D Signal.
s		saves the working buffer
S		Saves the image in memory to a Bitmaps
v		Prints the status variables
1		computes the One Dimensional FFT of the working buffer. Returns magnitude.
2		Computes the Magnitude of the Two Dimensional FFT of the Working Buffer.
3		Computes the Complex 2D FFT of the working buffer and stores it in the Complex Buffer

Table 5.3: Table show the meaning of the LED lights. F - Flashing, X - Don't Care

LED						
MOTOR	2	3	4	5	6	Meaning
Off	Off	Off	Off	Off	Off	System Initialising
On	X	X	X	X	X	Robot moving
Off	X	X	X	X	X	Robot not moving
X	On	On	On	X	X	System in Debug Mode
X	X	X	X	On	X	Left Wheel on a 'Tab'
X	X	X	X	Off	X	Left Wheel not on a 'Tab'
X	X	X	X	X	On	Right Wheel on a 'Tab'
X	X	X	X	X	Off	Right Wheel not on a 'Tab'
Off	On	Off	Off	X	X	Auto Run Mode - Robot taking photos
On	Off	On	Off	X	X	Auto Run Mode - Robot rotating
On	Off	Off	On	X	X	Auto Run Mode - Robot moving
Off	F	Off	Off	X	X	System Error - Generic
Off	F	F	X	X	X	System Error - SD Card Error
Off	F	X	F	X	X	System Error - Camera Error

References

Atmel Corporation. Uc3c-ek rev 2 schematic, 2009.

Atmel Corporation. *AT32UC3C0512C Datasheet*, 2012a.

Atmel Corporation. *ATMega644P Datasheet*, 2012b.

Atmel Corporation. Atmel software framework, 2012c.

Atmel Corporation. *ATXMega256A3BU Datasheet*, 2012d.

Creative Robotics Ltd. Hub-ee powered wheels, 2013.

Electronic Lives Manufacturing. Fatfs - generic fat file system module, 2012.

Farnell. Farnell online store, 2012.

Wayne Fulton. Image file formats - jpg, tif, png, gif. which to use?, 2010.

Patrick Goebel. Robot cartography: Ros + slam, 2012.

Steve R. Gunn. *Electroincs First Year Lab: C9 - Analogue Input*. University of Southampton, 2012.

I. Haller and S. Nedevschi. Design of interpolation functions for subpixel-accuracy stereo-vision systems. *Image Processing, IEEE Transactions on*, 21(2):889–898, 2012.

Rostam Affendi Hamzah, Sani Irwan Md Salim, and Hasrul Nisham Rosly. An effective distance detection of obstacles in stereo vision application. *Canadian Journal on Electrical and Electronics Engineering*, 1(3):49–53, 2010.

Jin Liu and Xiaofeng Lin. Equalization in high-speed communication systems. *Circuits and Systems Magazine, IEEE*, 4(2):4–17, 2004.

Microsoft. Xbox 360 kinect, 2012.

Jernej Mrovlje and Damir Vrančić. Distance measuring based on stereoscopic pictures. Technical report, University of Ljubljana, 2008.

Mark Nixon and Alberto S Aguado. *Feature Extraction & Image Processing for Computer Vision*. Academic Press, 2012.

OmniVision. *OmniVision Serial Camera Control Bus (SCCB) Functional Specification*, 2007.

PCB Cart. Pcb manufacturer, 2013.

Phillips. *PCA9542A : 2-channel I2C-bus multiplexer and interrupt logic*, 2009.

Phillips. *UM10204 - I2C-bus specification and user manual*, 2012.

Ashutosh Saxena, Jamie Schulte, and Andrew Y. Ng. Depth estimation using monocular and stereo cues, 2007.

Stephen Se, David Lowe, and Jim Little. Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks. *The International Journal of Robotics Research*, pages 735–758, 2002.

Texas Instruments. *Stellaris LM3S9B96 Datasheet*, 2012.

S. Thrun. Learning occupancy grid maps with forward sensor models. *Autonomous robots*, 15(2):111–127, 2003.

Edwin Tjandranegara. Distance estimation algorithm for stereo pair images, 2005.

D.M. Tsai and C.T. Lin. Fast normalized cross correlation for defect detection, 2003.

Vishay Semiconductors. *TCRT1000, TCRT1010 Datasheet*, 2012.

F. Zhao, Q. Huang, and W. Gao. Image matching by normalized cross-correlation. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 2, pages II–II. IEEE, 2006.

Appendix A

Gantt Chart

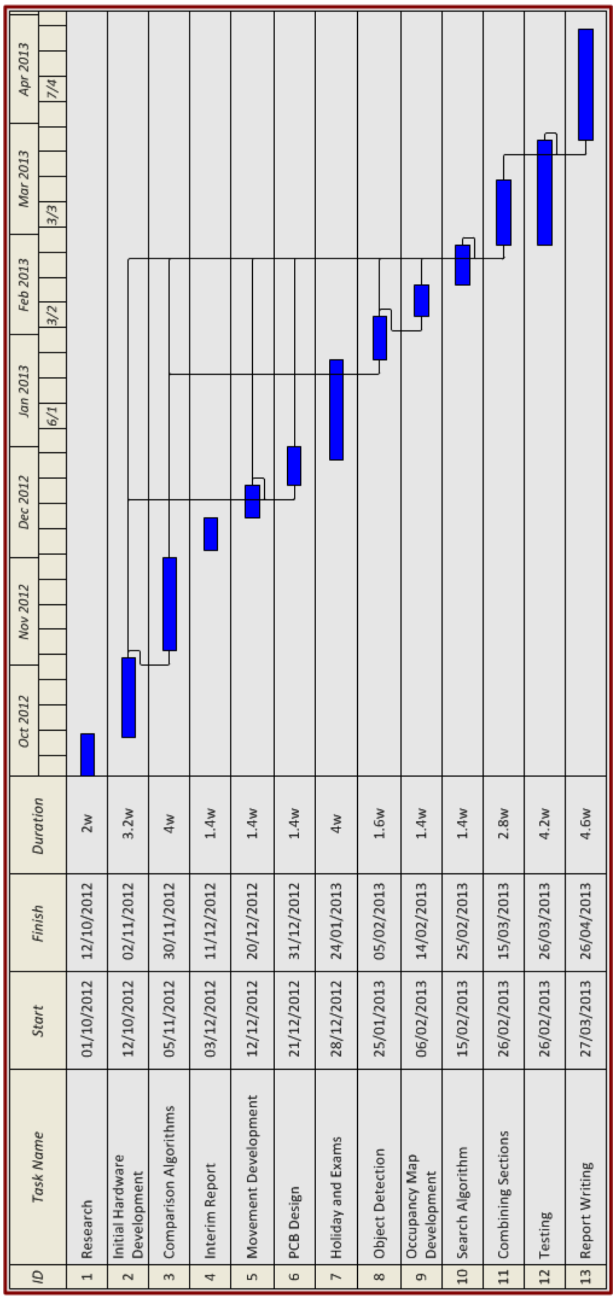


Figure A.1: Gantt Chart of how time will be spent in the areas of the project

Appendix B

Circuit Diagrams

B.1 OV7670 Breakout Board Schematic

B.2 Il Matto and Dual Camera Schematic

B.3 The Columbus Circuit Diagram

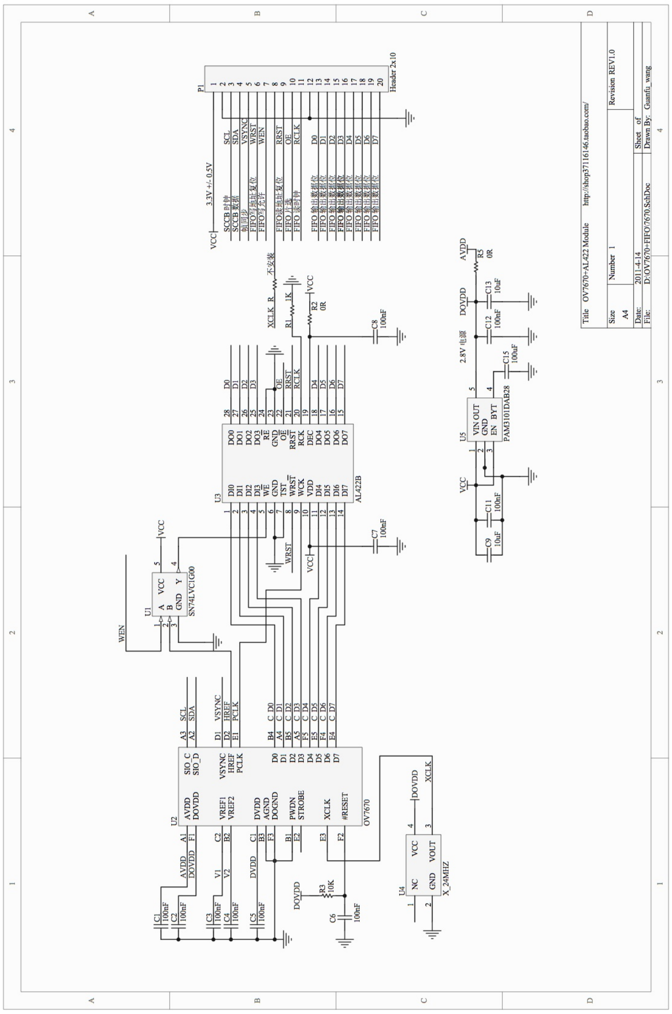


Figure B.1: The circuit diagram for the OV7670 breakout board

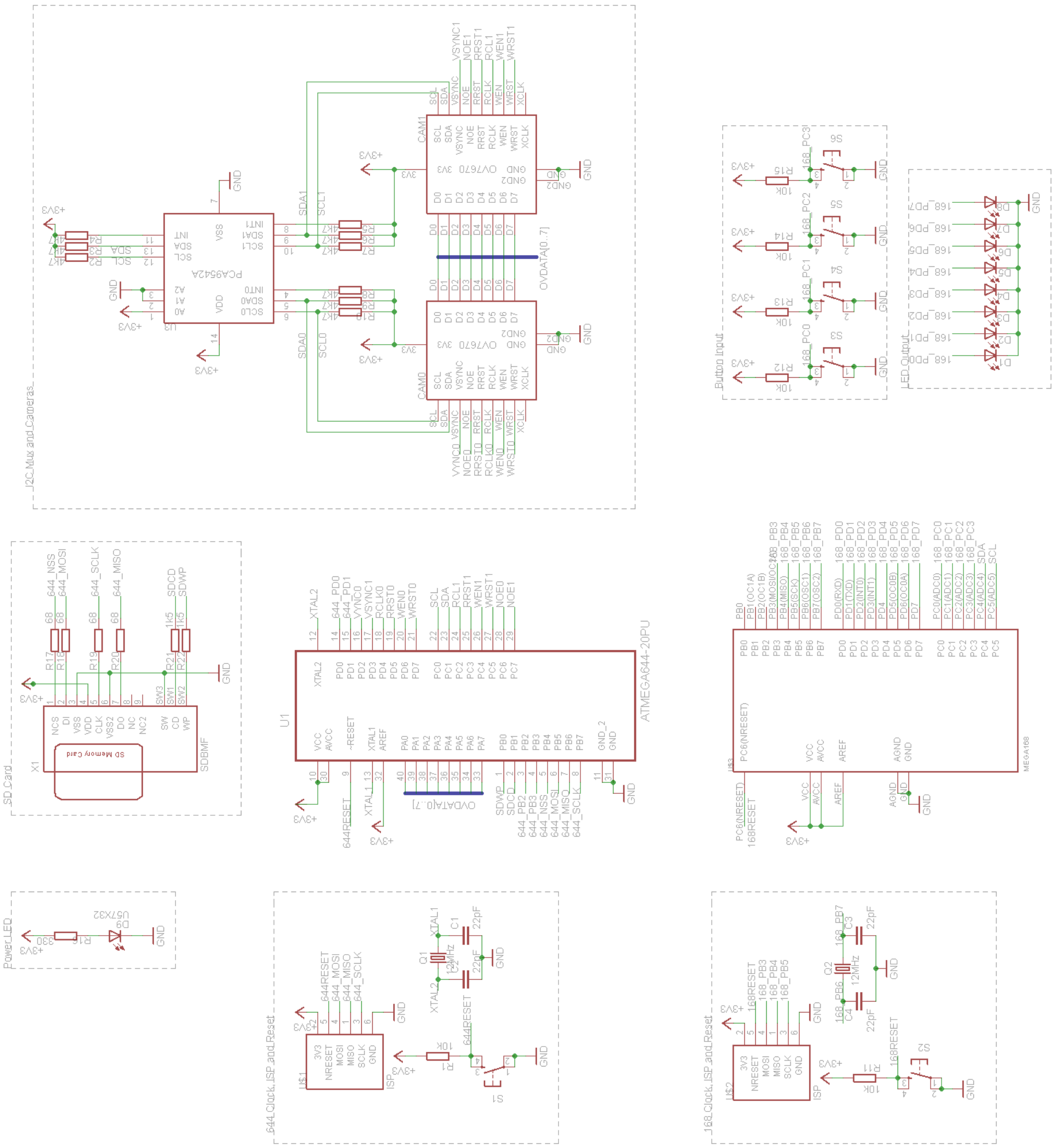


Figure B.2: The circuit diagram for Dual Cameras using the Il Matto Board

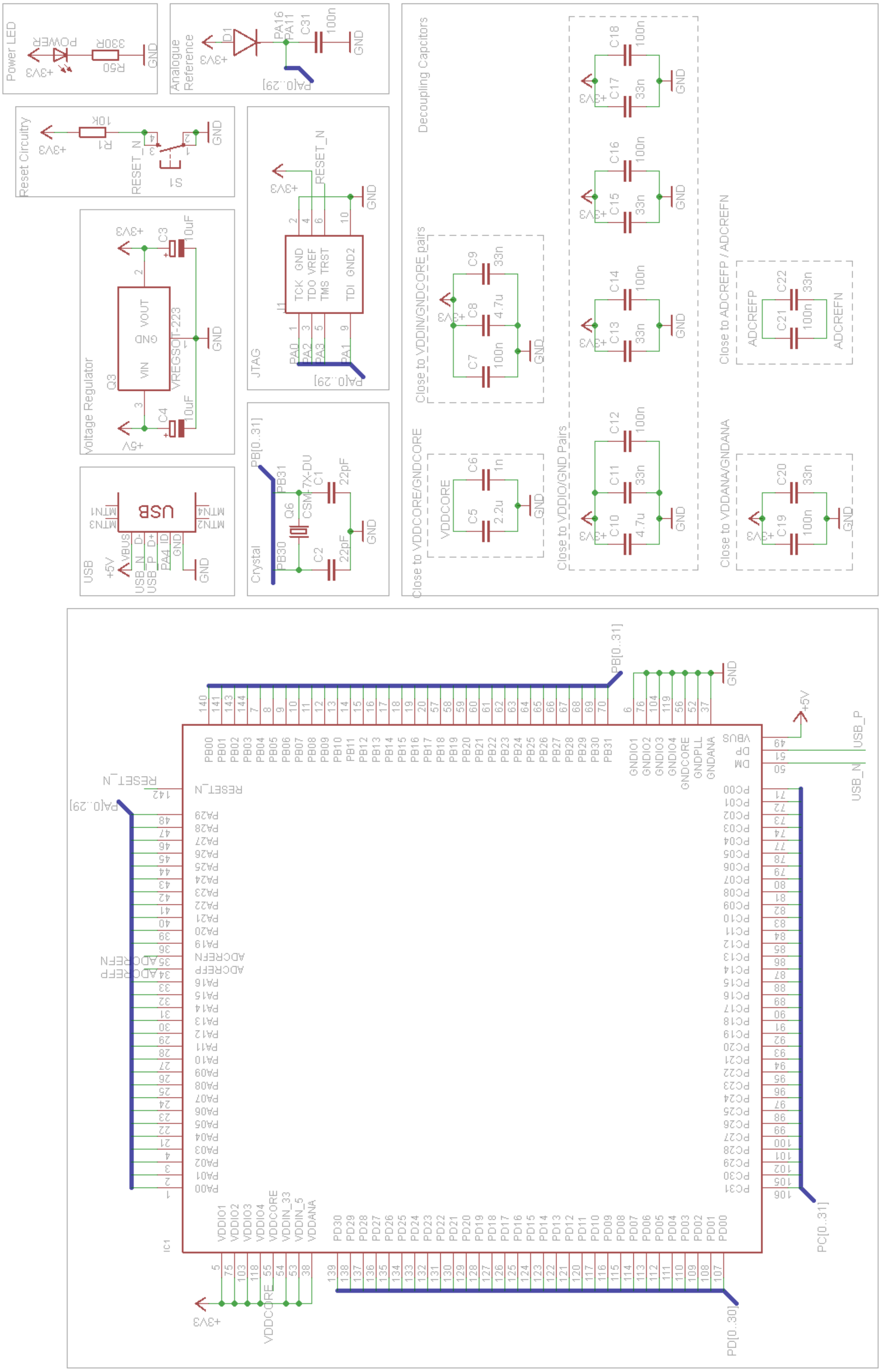
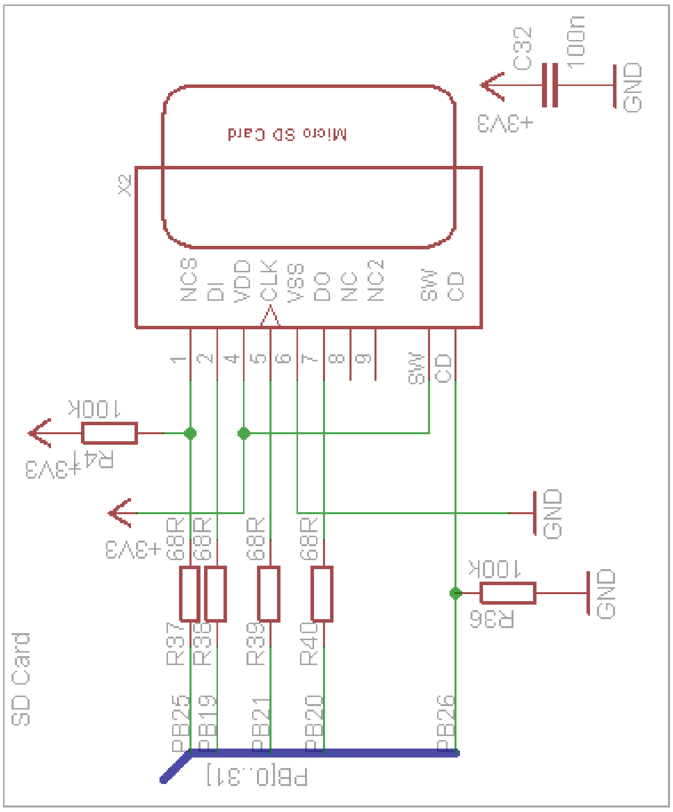


Figure B.3: The Columbus Circuit Diagram Page 1



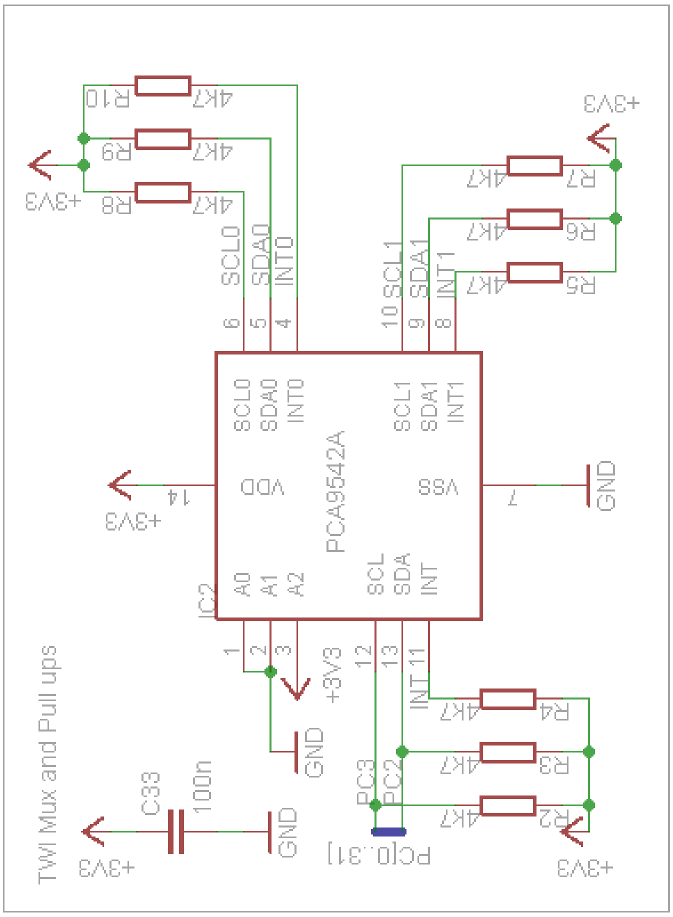


Figure B.4: The Columbus Circuit Diagram Page 2

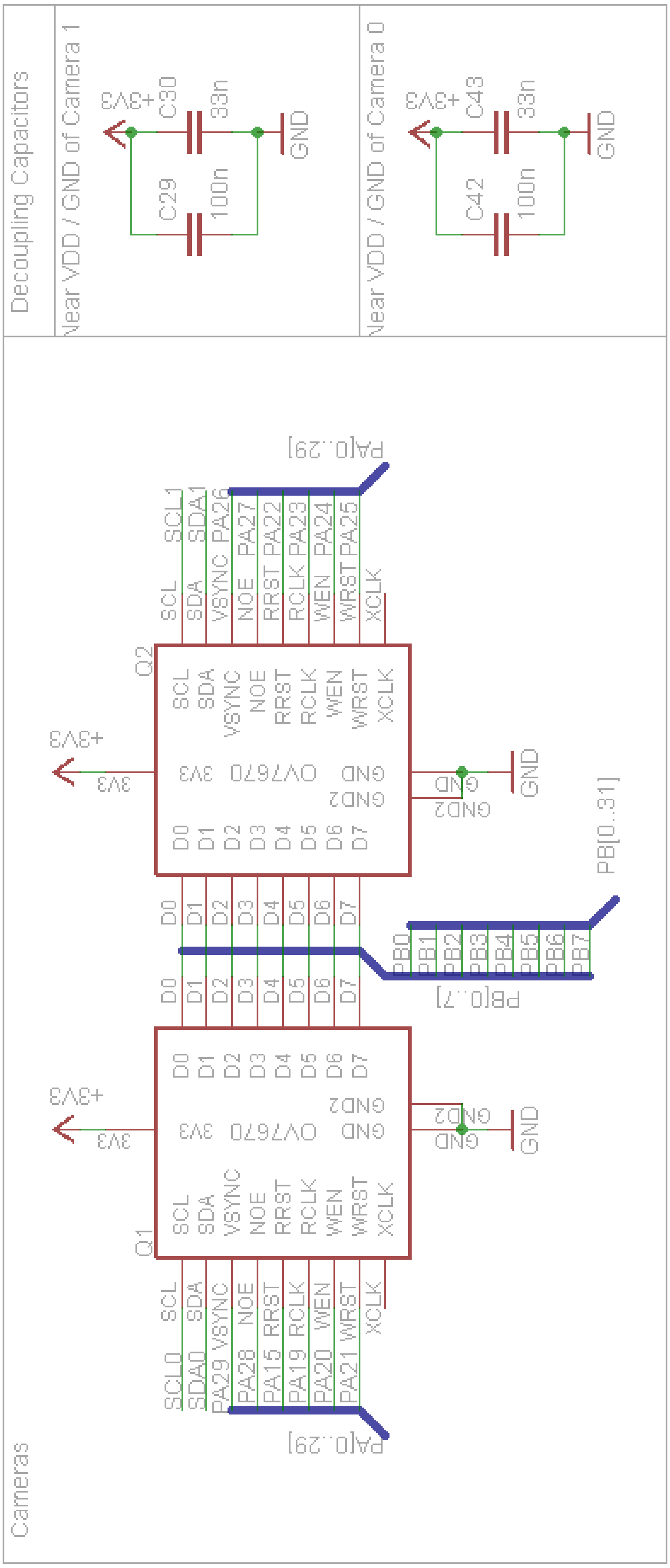


Figure B.5: The Columbus Circuit Diagram Page 3

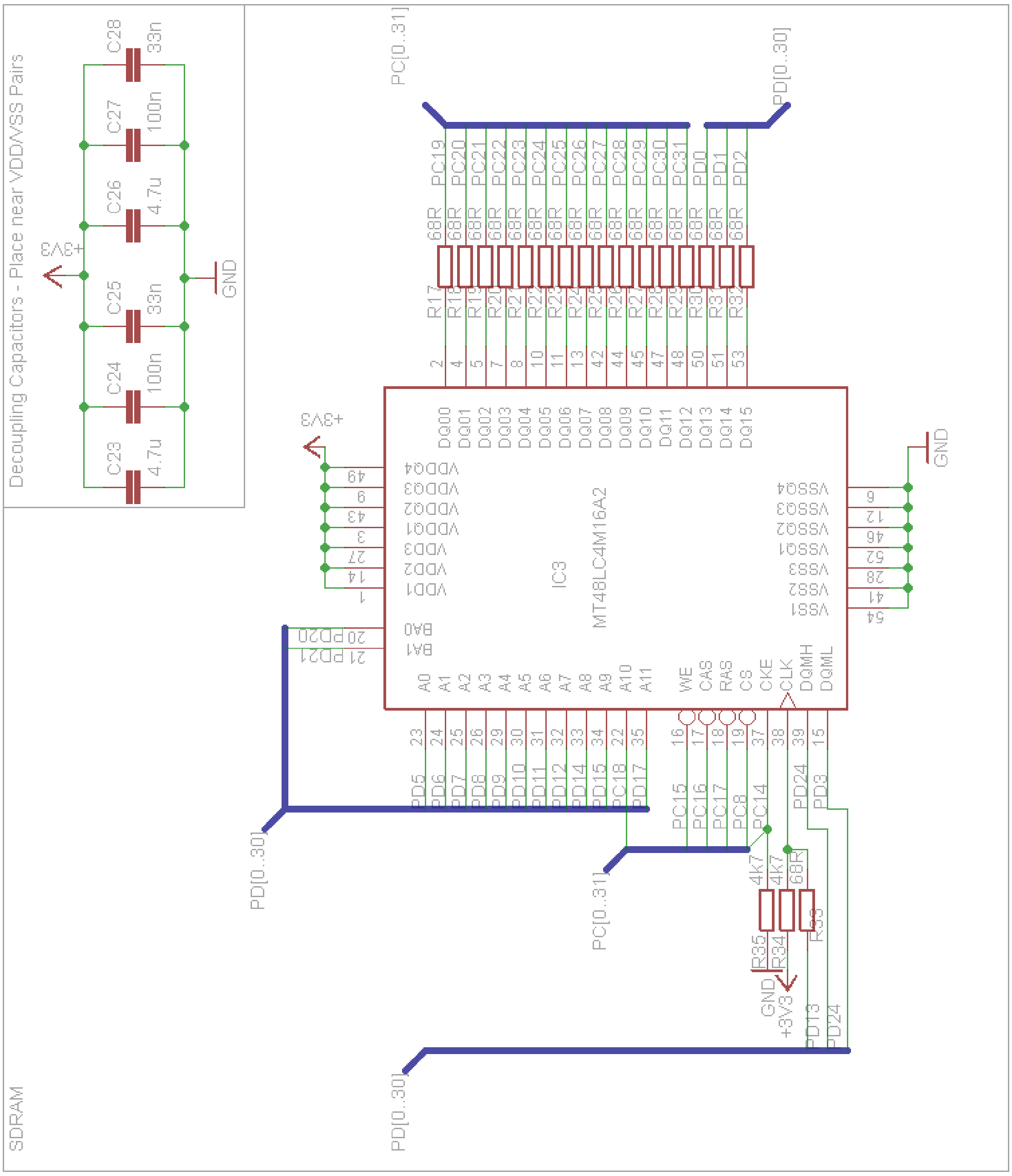


Figure B.6: The Columbus Circuit Diagram Page 4

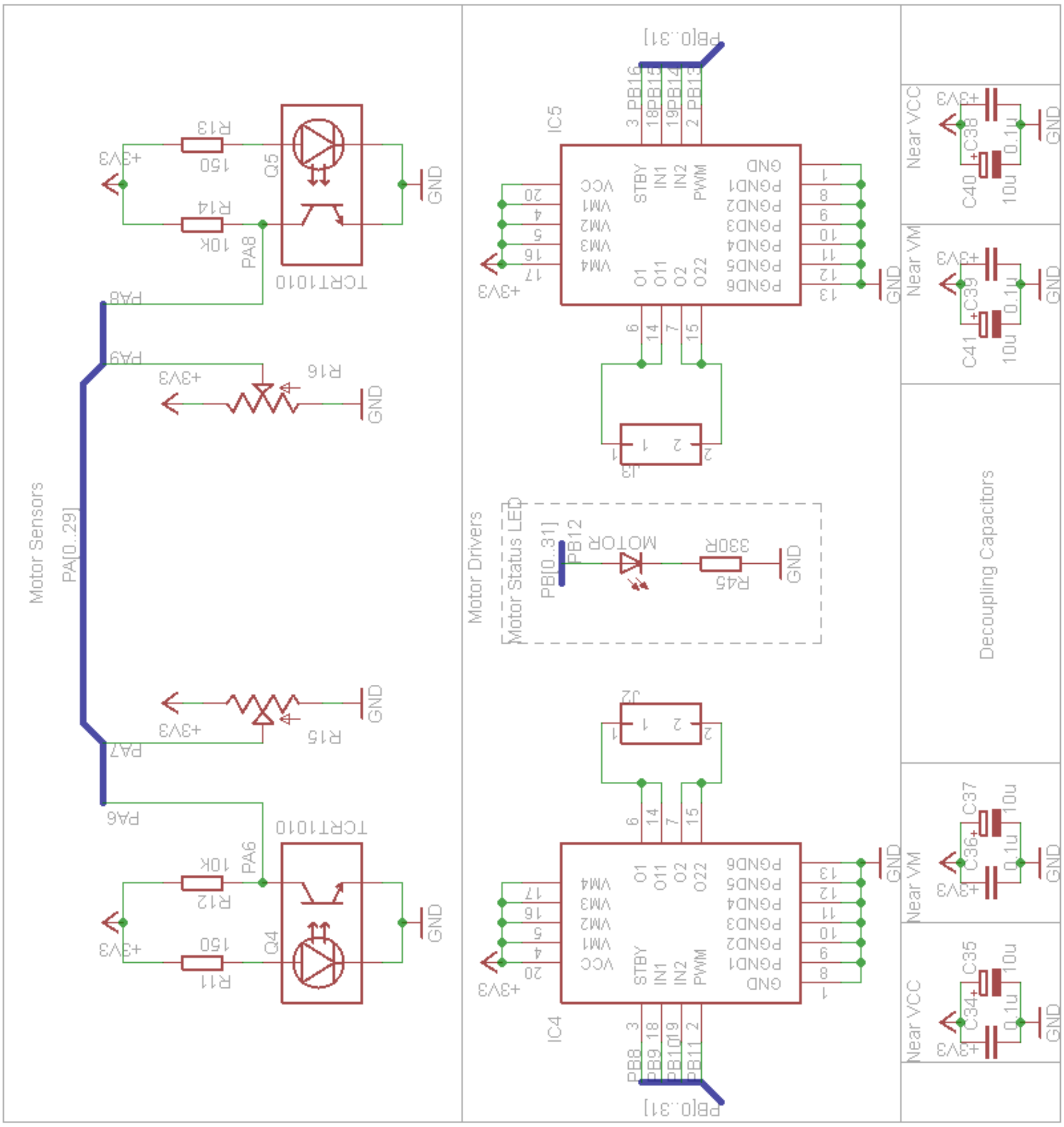


Figure B.7: The Columbus Circuit Diagram Page 5

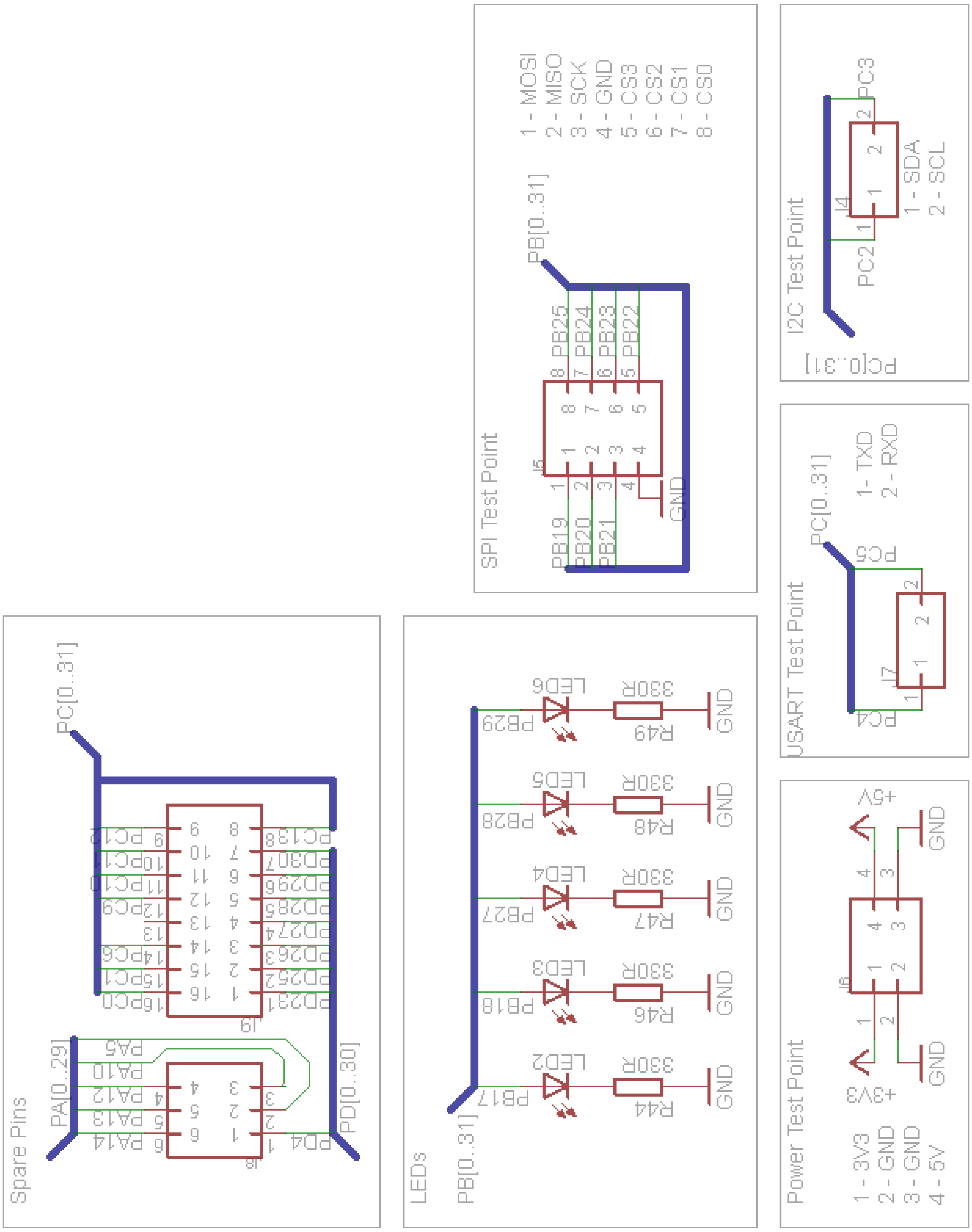


Figure B.8: The Columbus Circuit Diagram Page 6

Appendix C

PCB Design

C.1 PCB Top Side

C.2 PCB Bottom Side

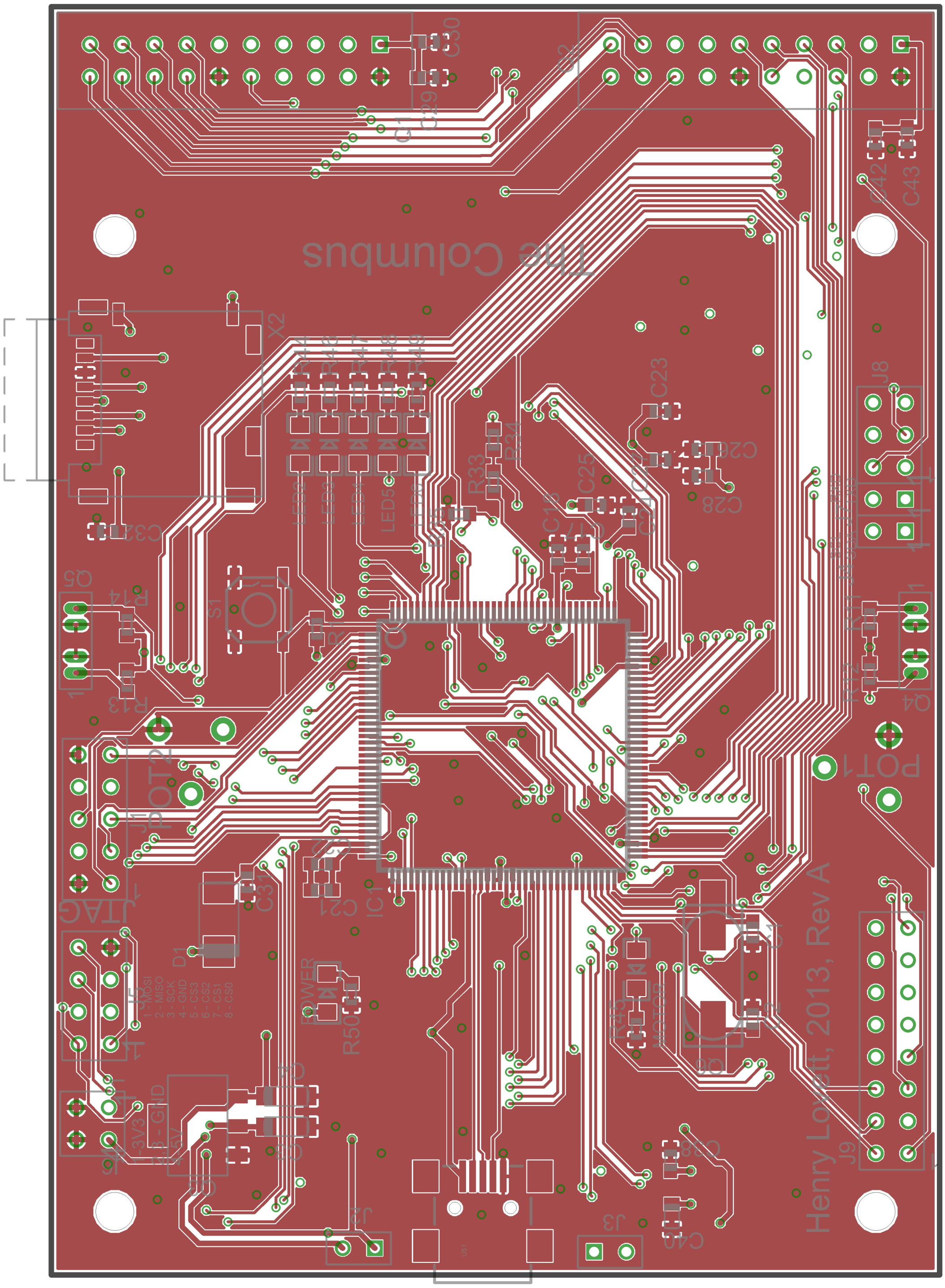


Figure C.1: The Top side of the CAD Design of the PCB

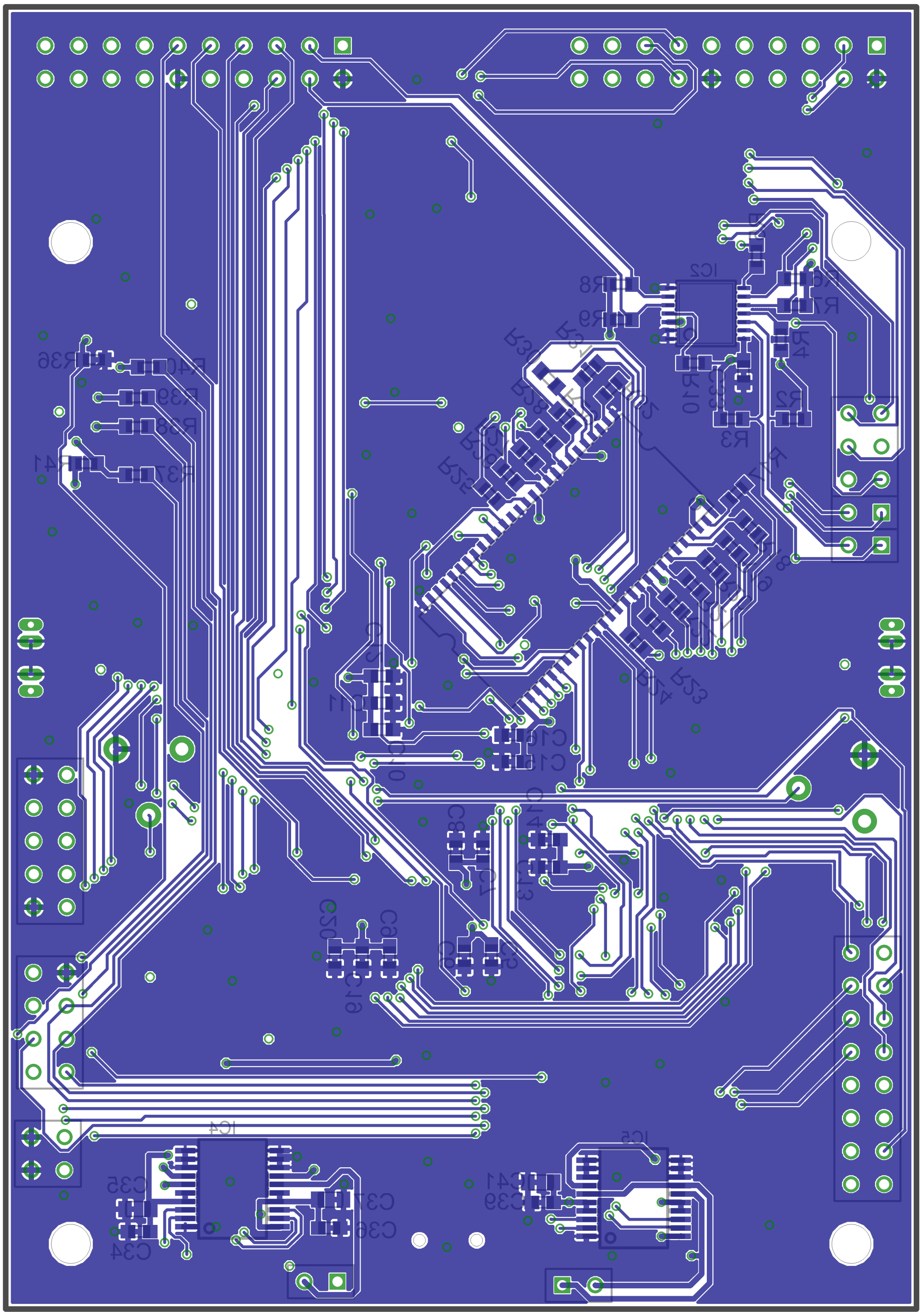


Figure C.2: The Bottom side of the CAD Design of the PCB

Appendix D

Costings and Components

Table D.1: A table of all components used and their costs.

Component	Cost per unit (£)	Quantity	Cost (£)	Source
PCB	205	1	205	PCBCart
Capactiors	0.155	43	6.67	Farnell
Clock	1.48	1	1.48	Farnell
Diode	0.48	1	0.48	Farnell
Headers	0.51	5	2.55	Farnell
I2C Mux PCA9542A	0.81	1	0.81	Farnell
LEDs	0.158	7	1.11	Farnell
Micro SD Card	4	1	4.00	Amazon
Micro SD Card Connector	2.04	1	2.04	Farnell
AT32UC3C0512C	15.39	1	15.39	Farnell
TB6593FNG	1.07	2	2.14	Farnell
Motors	0.42	2	0.84	Rapid
TCRT1010	0.94	2	1.88	Farnell
OV7670	17	2	34.00	
Potentiometer	0.43	2	0.86	Farnell
Resistors	0.066	46	3.04	Farnell
MT48LC4M16A2P	3.24	1	3.24	Farnell
Switch	0.45	1	0.45	Farnell
USB Socket	0.84	1	0.84	Farnell
LM1117MP	1.03	1	1.03	Farnell
		Total Cost	£287.84	

Table D.2: All components and their values (if applicable)

Compnent(s)	Value
IC1	AT32UC3C0512C
IC2	PCA9542A
IC3	MT48LC4M16A2
IC4, IC5	TB6593FNG
Q1, Q2	OV7670
Q3	LM1117MP-3.3
Q4, Q5	TCRT1010
S1	Tactile Switch
R15, R16	$10k\Omega$ Potentiometer
R17, R18, R19, R20, R21, R22, R23, R24, R25, R26, R27, R28, R29, R30, R31, R32, R33, R37, R38, R39, R40	68Ω
R11, R13	150Ω
R44, R45, R46, R47, R48, R49, R50	330Ω
R2, R3, R4, R5, R6, R7, R8, R9, R10, R34, R35	$4K7\Omega$
R1, R12, R14	$10k\Omega$
R36, R41	$100k\Omega$
C1, C2	$22pF$
C6	$1nF$
C9, C11, C13, C15, C17, C20, C22, C25, C28, C30, C43	$33Nf$
C7, C12, C14, C16, C18, C19, C21, C24, C27, C29, C31, C32, C33, C34, C36, C38, C39, C42	$100nF$
C5	$2.2\mu F$
C8, C10, C23, C26	$4.7\mu F$
C3, C4, C35, C37, C40, C41	$10\mu F$
D1	GF1A - Rectifier Diode
J1, J2, J3, J4, J5, J6, J7, J8, J9	0.1” header
LED2, LED3, LED4, LED5, LED6, Motor LED, Power LED	1206 LED
X2	Micro SD Card Socket

Appendix E

Contents of Files

Appendix F

Bitmap File Format

F.1 Bitmap File Format

Table F.1: Format of a Bitmap file with values used, to write an image from the camera to an SD Card

Section	Field	Description	Size (Bytes)	Value (hex)
Bitmap Header	Signature	Declares the file is a Bitmap Image	2	424D
	File Size	Size of the whole file including headers	4	36580200 (153654)[1]
	Reserved		4	00000000
	Offset to Pixel Array	The address of the start of the pixel data from the beginning of the file	4	36000000
DIB (Device Independent Bitmap) Header	Size	Size of the DIB Header (dictates the version)	4	7C000000
	Width	Width of the image (320 pixels)	4	40010000

Continued on next page

[1]This is different to the 225kB stated in table 3.1 due to omitting many optional fields

Table F.1 – continued from previous page

Section	Field	Description	Size (Bytes)	Value (hex)
	Height	Height of the image (240 pixels)	4	F0000000
	Planes	Number of colour planes	2	0100
	Bit Count	Number of bits per pixel	2	1000
	Compression	Compression Being Used, RGB Bit Fields	4	03 00 00 00
	Image Size	Size of the image	4	00 86 25 00
	X Resolution	Horizontal resolution in pixels per metre	4	13 0B 00 00
	Y Resolution	Vertical resolution in pixels per metre	4	13 0B 00 00
	Colours in Table	Number of colours in the colour table (not used)	4	00 00 00 00
	Important Colours	Number of Important Colours (0 means all colours are important)	4	00 00 00 00
	Red Mask	Bit mask of Red field	4	00 F8 00 00
	Green Mask	Bit mask of Green field	4	E0 07 00 00
	Blue Mask	Bit mask of Blue field	4	1F 00 00 00
	Alpha Mask	Bit mask of Alpha field	4	00 00 00 00
	Colour Space Type	Colour Space of the DIB	4	01 00 00 00
	Colour Space Endpoints	Sets endpoints for colours within the bitmap (not used)	36	Whole Field = 0
	Gamma Red	Gamma Value of Red Field (not used)	4	00 00 00 00

Continued on next page

Table F.1 – continued from previous page

Section	Field	Description	Size (Bytes)	Value (hex)
	Gamma Green	Gamma Value of Green Field (not used)	4	00 00 00 00
	Gamma Blue	Gamma Value of Blue Field (not used)	4	00 00 00 00
	Intent	Enum dictating the intent of the image (Picture)	4	03 00 00 00
	ICC Profile Data	Offset from the file start to the ICC Colour Profile (Not Used)	4	00 00 00 00
	ICC Profile Size	Size of the ICC Colour Profile (not used)	4	00 00 00 00
	Reserved		4	00 00 00 00
Image Data Format	Each field contains all the pixel data	Padding is used to make the table width a multiple of 4 (Not always needed)		
Pix[0, h-1]	Pix[1, h-1]	...	Pix[w-1, h-1]	Padding
⋮	⋮	⋮	⋮	⋮
Pix[0, 1]	Pix[1, 1]	...	Pix[w-1, 1]	Padding
Pix[0, 0]	Pix[1, 0]	...	Pix[w-1, 0]	Padding

Appendix G

Range Finding Derivations

G.1 Object is between the Cameras

Derivation from Mrovlje and Vrančić (2008).

$$B = B_1 + B_2 = D\tan(\varphi_1) + D\tan(\varphi_2) \tag{G.1}$$

$$D = \frac{B}{\tan(\varphi_1) + \tan(\varphi_2)} \tag{G.2}$$

$$D\tan\left(\frac{\varphi_0}{2}\right) = \frac{x_0}{2} \tag{G.3}$$

$$D\tan(\varphi_1) = x_1 \tag{G.4}$$

Dividing (G.4) by (G.3)

$$\frac{\tan(\varphi_1)}{\tan(\frac{\varphi_0}{2})} = \frac{2x_1}{x_0} \tag{G.5}$$

$$\tan(\varphi_1) = \frac{2x_1\tan(\frac{\varphi_0}{2})}{x_0} \tag{G.6}$$

This can also be shown for the right camera:

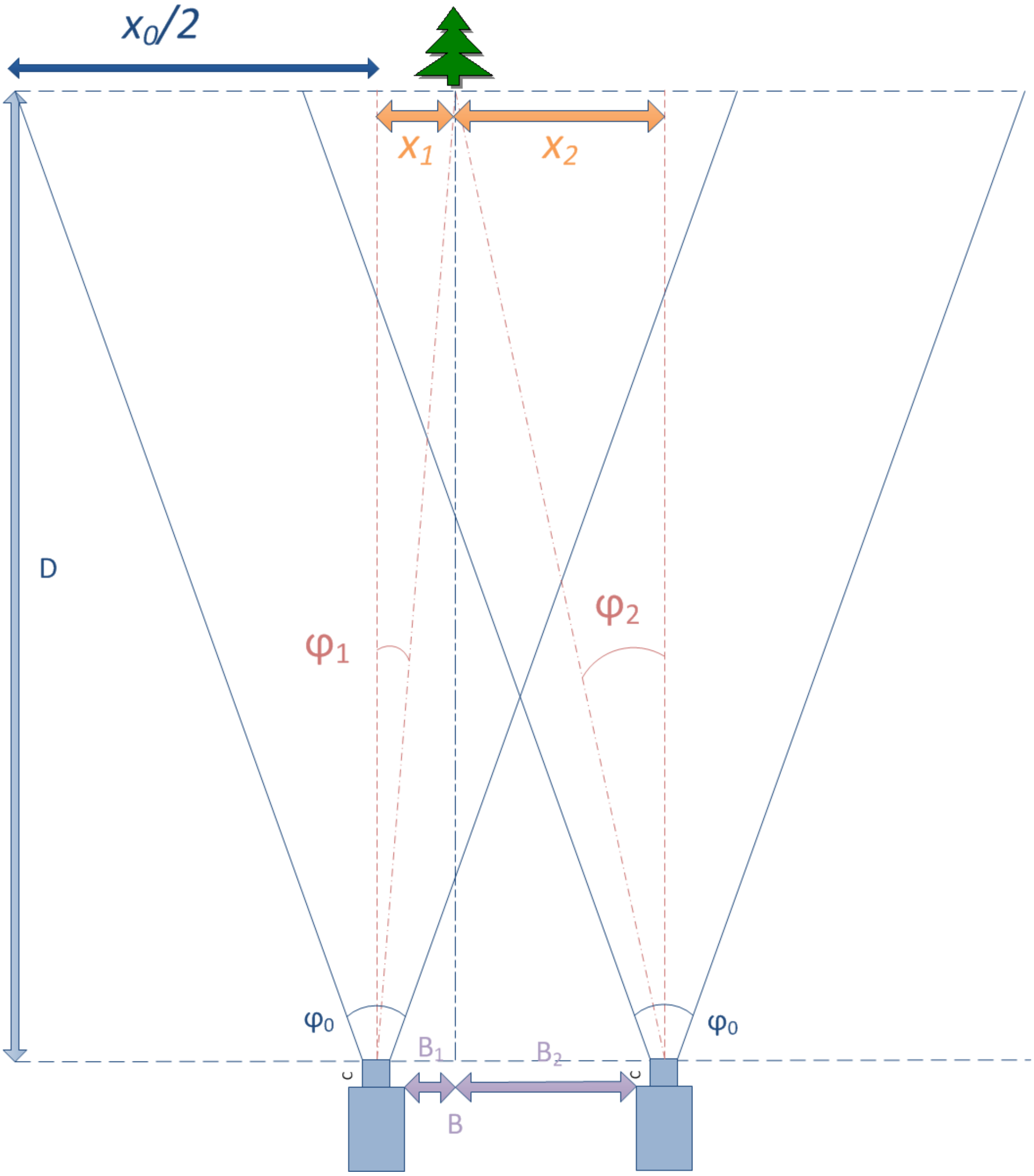


Figure G.1: Problem 1 - Object is between the Cameras

$$\tan(\varphi_2) = \frac{-2x_2\tan(\frac{\varphi_0}{2})}{x_0} \tag{G.7}$$

Substitution equations (G.6) and (G.7) into (G.2) gives

$$D = \frac{Bx_0}{2\tan(\frac{\varphi_0}{2})(x_1 - x_2)} \tag{G.8}$$

G.2 Object is to the same side in each camera

Derivation is based on the derivation from Tjandranegara (2005). Using figure G.2:

$$D.\tan(\varphi_1) = x_1 \tag{G.9}$$

$$D.\tan\left(\frac{\varphi_0}{2}\right) = \frac{x_0}{2} \tag{G.10}$$

$$\frac{\tan(\varphi_1)}{\tan(\frac{\varphi_0}{2})} = \frac{2x_1}{x_0} \tag{G.11}$$

$$\varphi_1 = \arctan\left(\frac{2x_1}{x_0}\tan\left(\frac{\varphi_0}{2}\right)\right) \tag{G.12}$$

and similarly

$$\varphi_2 = \arctan\left(\frac{2x_2}{x_0}\tan\left(\frac{\varphi_0}{2}\right)\right) \tag{G.13}$$

$$\theta = \varphi_2 - \varphi_1 \tag{G.14}$$

Using the sine equality rule:

$$\frac{R}{\sin(\frac{\pi}{2} - \varphi_2)} = \frac{B}{\sin(\theta)} \tag{G.15}$$

$$R = B.\frac{\sin(\frac{\pi}{2} - \varphi_2)}{\sin(\theta)} = B\frac{\cos(\varphi_2)}{\sin(\theta)} \tag{G.16}$$

$$D = \cos(\varphi_1).R \tag{G.17}$$

Substituting (G.14) into (G.16), and then into (G.17):

$$D = B.\frac{\cos(\varphi_2).\cos(\varphi_1)}{sin(\varphi_2 - \varphi_1)} \tag{G.18}$$

Where φ_1 is defined in equation (4.7) and φ_2 is defined in equation (4.8).

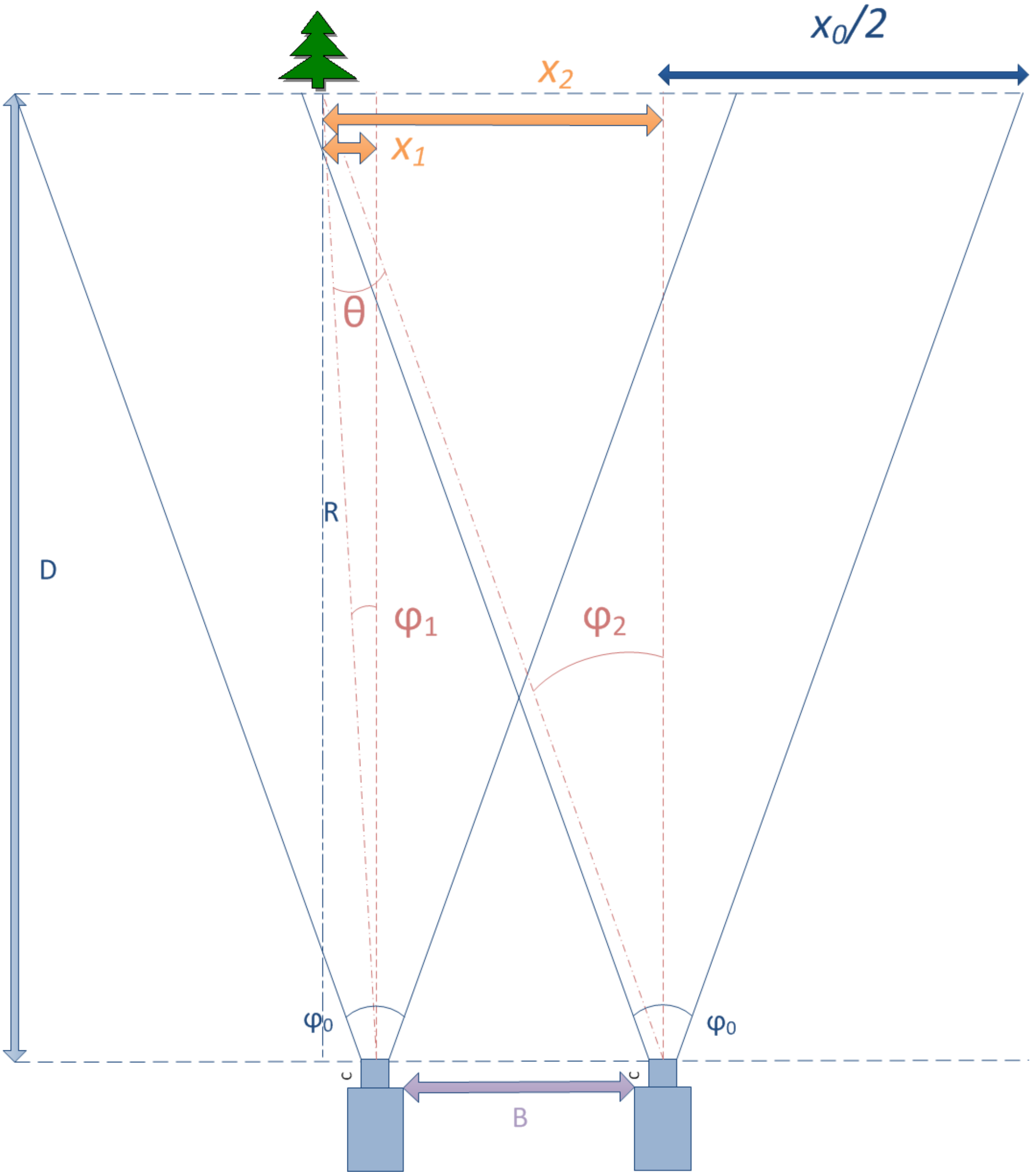


Figure G.2: Problem 2 - Object is to the same side in both cameras

G.3 Object is in front of a camera

The distance, D, in this problem is given by:

$$D = B \tan\left(\frac{\pi}{2} - \varphi_2\right) \tag{G.19}$$

Where φ_2 can be found from equation 4.8.

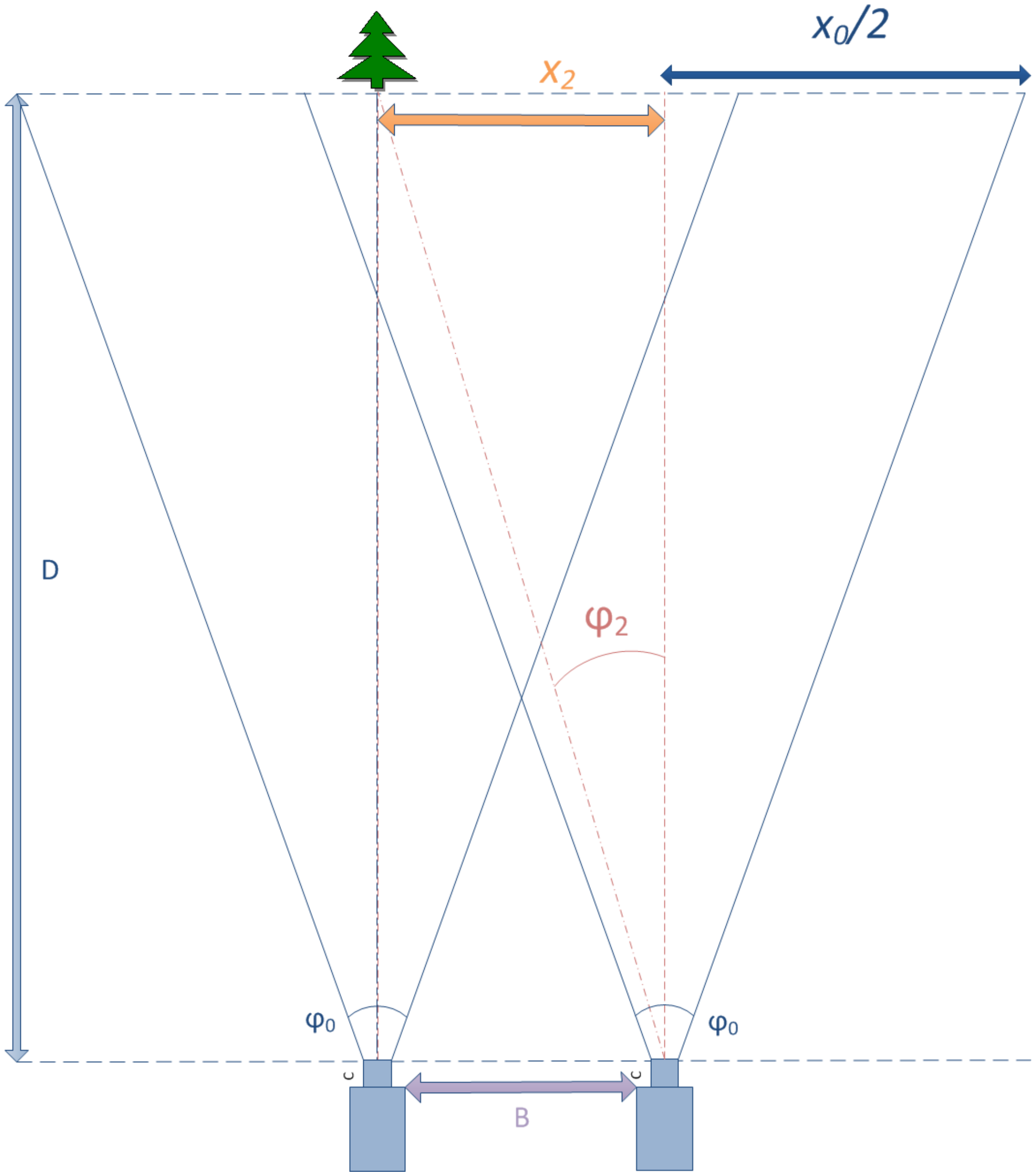


Figure G.3: Problem 3 - Object is directly in front of a camera

Appendix H

Source Code

H.1 C Code for AVR

H.1.1 The Columbus Source Code

H.1.1.1 main.c

../Code/The_Columbus/ColumbusTest/src/main.c

```c
/**
 * \file
 *
 * \brief Empty user application template
 *
 */

/*
 * Include header files for all drivers that have been imported from
 * Atmel Software Framework (ASF).
 */
#define DSP16_FORMAT 10
#define COMMAND_BUFFER_SIZE   128
#include <asf.h>
#include <conf_board.h>
#include "CustomDevices/CustomDevices.h"
#include "conf_sd_mmc_spi.h"
#include "fat.h"
#include "file.h"
#include "navigation.h"
#include "fastmath.h"
#include "delay.h"
#include "stdio.h"
```

```c
void Get_Line( char * CommandBuffer );
int Auto_Run();
void Debug_Mode();
void System_Error();


int main (void)
{
  Columbus_Status.SD_Card = &SD_Status;
  Columbus_Status.Cameras = &OV7670_Status;
  Columbus_Status.I2CMux = &PCA9542A;
  Columbus_Status.SD_Card = &SD_Status;
  Columbus_Status.Motors = &Motor_Control;
  board_init();
  print_dbg("\n\r");
  print_dbg(THE);
  print_dbg(COLUMBUS);
  print_dbg(ASCII_SHIP);
  System_Test();

  if(Columbus_Status.Status != STATUS_OK)
  {
    System_Error();
  }
  print_dbg("\n\rColumbus Ready!");

  //Enter into Auto Run if not in debug mode
  if(gpio_get_pin_value(DEBUG_PIN))
  {
    print_dbg("\n\rEntering Auto Run");
    if(1 == Auto_Run(0)) //if
      System_Error(); //use the system error loop to stop operation
  }
  print_dbg("\n\rEntering Debug Mode...");
  Debug_Mode();

}
void Get_Line( char * CommandBuffer )
{
  int c = 0;

  while(c != 13)
  {
    c = usart_getchar(DBG_USART);
    if(c == '\b')
    {
      CommandBuffer--;
      *(CommandBuffer) = 0;
    }
    else
    {
      *(CommandBuffer) = c;
      CommandBuffer++;
    }
    print_dbg_char(c);
  }
  usart_putchar(DBG_USART, 6);
```

```
}
void PrintStatus()
{
  print_dbg("\n\n\rColumbus Status:\t0x");
  print_dbg_hex(Columbus_Status.Status);
  print_dbg("\n\rSD Card:\n\rStatus:\t\t\t0x");
  print_dbg_hex(Columbus_Status.SD_Card->Status);
  print_dbg("\n\rMemory Size :\t\t0x");
  print_dbg_hex(Columbus_Status.SD_Card->Memory_size);
  print_dbg("\n\rMotors:");
  print_dbg("\n\rLeft State :\t\t0x");
  print_dbg_hex(Columbus_Status.Motors->Left_State);
  print_dbg("\n\rLeft Count :\t\t0x");
  print_dbg_hex(Columbus_Status.Motors->Left_Count);
  print_dbg("\n\rRight State :\t\t0x");
  print_dbg_hex(Columbus_Status.Motors->Right_State);
  print_dbg("\n\rRight Count :\t\t0x");
  print_dbg_hex(Columbus_Status.Motors->Right_Count);
  print_dbg("\n\rCameras:");
  print_dbg("\n\rStatus :\t\t0x");
  print_dbg_hex(Columbus_Status.Cameras->Status);
  print_dbg("\n\rVSYNC0 State :\t\t0x");
  print_dbg_hex(Columbus_Status.Cameras->VSYNC0_State);
  print_dbg("\n\rVSYNC1 State :\t\t0x");
  print_dbg_hex(Columbus_Status.Cameras->VSYNC1_State);
  print_dbg("\n\rImage Counter:\t\t0x");
  print_dbg_hex(Columbus_Status.Cameras->ImageCount);
  print_dbg("\n\rI2C Mux:");
  print_dbg("\n\rStatus :\t\t0x");
  print_dbg_hex(Columbus_Status.I2CMux->Status);
  print_dbg("\n\rChannel Selected :\t0x");
  print_dbg_hex(Columbus_Status.I2CMux->ChannelSelected);
}
void System_Error()
{
  while(1)
  {
    LED2_SET;
    if(Columbus_Status.Status & SD_ERR)
      LED3_SET;
    if(Columbus_Status.Status & CAM_ERR)
      LED4_SET;
    delay_ms(500);

    LED2_CLR;
    if(Columbus_Status.Status & SD_ERR)
      LED3_CLR;
    if(Columbus_Status.Status & CAM_ERR)
      LED4_CLR;
    delay_ms(500);
    PrintStatus();
  }
}

#define AutoRun_Commands_FileName "AutoRun.txt"
const AutoCommand_t DefaultCommands[13] = {
  {'F', 10},
  {'P', 0},
```

```
  {'R', 90},
  {'F', 10},
  {'P', 0},
  {'R', 90},
  {'F', 10},
  {'P', 0},
  {'R', -90},
  {'F', 10},
  {'P', 0},
  {'R', -90},
  { 'q', 0}
};
AutoCommand_t *AutoCommands = DefaultCommands;//Use the default by default.
#define ReadBuffSize 256
void LoadCommands()
{
  int i = 1;
  int j, k;
  char c = 0;
  char buff[ReadBuffSize];
  nav_filelist_reset();
  if(nav_filelist_findname((FS_STRING)AutoRun_Commands_FileName, false))
  { //If the file exists, load it
    print_dbg("\n\rAuto Run File Found");
    nav_setcwd((FS_STRING)AutoRun_Commands_FileName, false, false);
    file_open(FOPEN_MODE_R);
    while(!file_eof())//count how many commands there are
    {
      c = file_getc();
      if(c == 0xD)//if a carriage return if found
        i++;

    }
    sprintf(buff, "\n\r%d commands found\n\r", i);
    print_dbg(buff);

    AutoCommands = malloc(i*sizeof(AutoCommand_t));//Initalise the array

    file_seek(0, FS_SEEK_SET);
    i = 0;//command counter
    c = 0;
    while(!file_eof())
    {
      k = file_read_buf(buff, ReadBuffSize);//read a chunk of the files
      if(k > ReadBuffSize)
        k = ReadBuffSize;
      //parse it
      for(j = 0; j < k; )
      {
        while(buff[j] == ' ') //find next non-whitespace
          j++;
        AutoCommands[i].Command = buff[j++];
        while(buff[j] == ' ') //find next non-whitespace
          j++;
        if((AutoCommands[i].Command == 'F') || (AutoCommands[i].Command == 'B'
    ) || AutoCommands[i].Command == 'R' || AutoCommands[i].Command == 'J')
          AutoCommands[i].Arg = atoi(buff+j);
        else
```

```
          AutoCommands[i].Arg = 0;
        i++;
        while((buff[j++] != 0xA) && (j < k));//find the next line char

      }
    }
    file_close();
  }
  else
  { //Load default commands
    print_dbg("\n\rAuto Run File Not Found, Using Default Commands:");
  }
}

int Auto_Run(int Debug)//Debug = 1, don't run commands
{
  int PC = 0;

  LoadCommands();
  while(1)
  {
    switch(AutoCommands[PC].Command)
    {
      case 'F'://Move Forward
        print_dbg("\n\rMove Forward ");
        LED4_SET;
        print_dbg_ulong(AutoCommands[PC].Arg);
        if(!Debug){
        Motors_Move(AutoCommands[PC].Arg);
        }
        LED4_CLR;
        break;

      case 'B'://Move Backward
        print_dbg("\n\rMove Backward ");
        LED4_SET;
        print_dbg_ulong(AutoCommands[PC].Arg);
        if(!Debug){
        Motors_Move(-AutoCommands[PC].Arg);
        }
        LED4_CLR;
        break;

      case 'R'://Rotate
        print_dbg("\n\rRotate ");
        LED3_SET;
        print_dbg_ulong(AutoCommands[PC].Arg);
        if(!Debug){
        Motors_Rotate(AutoCommands[PC].Arg);
        }
        LED3_CLR;
        break;

      case 'P'://Take Photo
        print_dbg("\n\rTaking Photos");
        LED2_SET;
        if(!Debug){
        FIFO_Reset(CAMERA_LEFT | CAMERA_RIGHT);
```

```
        if(TakePhoto(CAMERA_LEFT | CAMERA_RIGHT) == CAMERAS_BUSY){
          print_dbg("\n\rCameras Busy");
          break;
        }
        while(Photos_Ready() == false)
        ;

        if(Store_Both_Images() == true)
        print_dbg("\n\rImages Stored Successfully!");
        }
        LED2_CLR;
        break;

      case 'J': //Jump
        print_dbg("\n\rProgram Jump to ");
        print_dbg_ulong(AutoCommands[PC].Arg);
        if(!Debug){
        PC = AutoCommands[PC].Arg;
        }
        break;

      case 'q': //End and enter debug
        print_dbg("\n\rQuit to Debug");
        return 0;

      default: //System Error
        if(!Debug){//Inevitable in Debug mode
        Columbus_Status.Status |= AutoRunCMD_ERR;
        print_dbg("\n\rSystem Error");
        }
        return 1;
    }
    PC++;
  }
}



void Debug_Mode()
{
  Image_t image;
  unsigned long i, j, tmp = 0;
  char *Ptr;
  //  volatile unsigned long *sdram = SDRAM;
  char CommandBuffer[COMMAND_BUFFER_SIZE];
  int *Working_Buffer = NULL;
  int SizeOfWorking_Buffer = 0;
  A_ALIGNED dsp16_complex_t *ComplexBuffer;
  int SizeOfComplex_Buffer = 0;
  LED2_SET;//Set these LEDs to indicate Columbus is in debug mode;
  LED3_SET;
  LED4_SET;
  while(1)
  {
    print_dbg(PROMPT);
    Get_Line(CommandBuffer);
    Ptr = CommandBuffer;
    switch(*Ptr++)
```

```
      {
        case '?':
          print_dbg(HELP);
          break;
        case 'A':
          print_dbg("\rRunning AutoRun in Debug Mode;");
          Auto_Run(1); //Runs in debug mode
          break;
        case '1':
          print_dbg("\r1D FFT;");
          SizeOfComplex_Buffer = FFT_SIZE;
          ComplexBuffer = mspace_malloc(sdram_msp, SizeOfComplex_Buffer *
      sizeof(ComplexBuffer));
          i = Get_sys_count();
          FFT1D(Working_Buffer, ComplexBuffer);
          i = Get_sys_count() - i;
          print_dbg("\n\rCycles Taken for 1D FFT = ");
          print_dbg_ulong(i);
          break;
        case '2':
          print_dbg("\r2D FFT;");
          FFT2Dabs(Working_Buffer);
          break;
        case '3':
          print_dbg("\rComplex FFT2D:");
          SizeOfComplex_Buffer = FFT_SIZE * FFT_SIZE;
          ComplexBuffer = mspace_malloc(sdram_msp, SizeOfComplex_Buffer *
      sizeof(ComplexBuffer));
          i = Get_sys_count();
          FFT2DCOMPLEX(Working_Buffer, ComplexBuffer, SizeOfWorking_Buffer);
          i = Get_sys_count() - i;
          print_dbg("\n\rCycles Taken for 2D FFT = ");
          print_dbg_ulong(i);
          break;
        case 'm':
          print_dbg("\r1D FFT Magnitude");
          FFT1D_Abs(Working_Buffer);
          break;
        case 'B':
          print_dbg("\rReading Bitmap;");
          ReadBitmap("Image_R_0.bmp", &image);
          print_dbg("\n\rBitmap Data Returned:\n\rImage Height = ");
          print_dbg_ulong(image.Height);
          print_dbg("\n\rImage Width = ");
          print_dbg_ulong(image.Width);
          break;

        case 'c':
          print_dbg("\rConverting Working Buffer to Fixed Point");
          for(i = 0; i < SizeOfWorking_Buffer ; i++)
          {
            Working_Buffer[i] = DSP16_Q (Working_Buffer[i]);
          }
          break;

        case 'C':
          print_dbg("\rConverting Working Buffer back from Fixed Point");
          j = DSP16_Q(1);
```

```
          for(i = 0; i < SizeOfWorking_Buffer ; i++)
          {
            Working_Buffer[i] = Working_Buffer[i] / j;
          }
          break;

        case 'D':
          print_dbg("\rFreeing Working Buffer");
          mspace_free(sdram_msp, Working_Buffer);
          break;

        case 'i':
          print_dbg("\rImage info:");
          print_dbg("\n\rImage Pointer = ");
          print_dbg_ulong(image.ImagePtr);
          print_dbg("\n\rImage Height = ");
          print_dbg_ulong(image.Height);
          print_dbg("\n\rImage Width = ");
          print_dbg_ulong(image.Width);
          break;

//        case 'I': //Not yet tested
//          print_dbg("\rInverse Fourier Transform;");
//          IFFT2D(ComplexBuffer);
//          break;

        case 'k':
          print_dbg("\rComplex Buffer:\n\r[");
          for (i = 0; i < SizeOfComplex_Buffer; i ++)
          {
            //            print_dbg_ulong(ComplexBuffer[i].real);
            //            print_dbg(" + ");
            //            print_dbg_ulong(ComplexBuffer[i].imag);
            //            print_dbg(", ");
            if(ComplexBuffer[i].imag >= 0)
            sprintf(CommandBuffer, "%d + %dj,", ComplexBuffer[i].real, ComplexBuffer[i].imag);
            else
            sprintf(CommandBuffer, "%d %dj,", ComplexBuffer[i].real, ComplexBuffer[i].imag);
            print_dbg(CommandBuffer);
          }
          print_dbg("]\n\r");
          break;
        case 'M': //Motor Related
          while(*Ptr == ' ')
          Ptr++; //Find next non - space char

          switch(*(Ptr++))
          {
            case 'q': // Reset Motors
              print_dbg("\rResetting Motors");
              Motors_Reset();
              break;

            case 'F': //Move Forward
              while(*Ptr == ' ')
                Ptr++; //Find next non - space char
```

```
              i = atoi(Ptr);
              Motors_Move(i);
              break;
            case 'T':
              while(*Ptr == ' ')
                Ptr++; //Find next non-space char
              i = atoi(Ptr);
              Motors_Rotate(i);
              break;
            case 'L':
              Columbus_Status.Motors->Left_Count = GAMMA + 1;
              Columbus_Status.Motors->Left_State = FORWARD;
              Motor_Start(MOTOR_L);
              Motors_Execute();
              break;
            case 'R':
              Columbus_Status.Motors->Right_Count = GAMMA + 1;
              Columbus_Status.Motors->Right_State = FORWARD;
              Motor_Start(MOTOR_R);
              Motors_Execute();
              break;
            default:
              print_dbg("\rCommand Not Recognised");
              break;
          }

        break;

//      case 'p': //Not yet tested
//        print_dbg("\rPreparing Image;");
//        PrepareImage(&image);
//        print_dbg("\rImage Prepared!");
//        break;

        case 'P'://take a photo
          FIFO_Reset(CAMERA_LEFT | CAMERA_RIGHT);
          print_dbg("\rTaking Photos");
          if(TakePhoto(CAMERA_LEFT | CAMERA_RIGHT) == CAMERAS_BUSY){
            print_dbg("Cameras Busy");
            break;
          }
          while(Photos_Ready() == false)
          ;

          if(Store_Both_Images() == true)
          print_dbg("\n\rImages Stored Successfully!");
          break;

        case 'r':
          if (Working_Buffer == 0)
          {
            print_dbg("\rWorking Buffer Not Initialised");
            break;
          }
          print_dbg("\rWorking Buffer:\n\r[");
          for(i = 0; i < SizeOfWorking_Buffer; i++)
          {
            sprintf(CommandBuffer, "%d,", (dsp16_t)Working_Buffer[i]);
```

```
          print_dbg(CommandBuffer);
        }
        print_dbg("\b\b]\n\r");
        break;
      case 'R':
        Working_Buffer = mspace_malloc(sdram_msp, FFT_SIZE);
        SizeOfWorking_Buffer = FFT_SIZE;
        print_dbg("\rReading in signal.bin");
        ReadSignal(Working_Buffer);
        break;

      case 's'://save the working buffer
        print_dbg("\rSaving Working Buffer;");
        SaveBuff(Working_Buffer, SizeOfWorking_Buffer);
        break;
      case 'd':
        print_dbg("\rSaving Working Buffer as CSV;");
        SaveBuff_CSV("Buffer_result.csv", Working_Buffer, SizeOfWorking_Buffer);
        break;
      case 'g':
        print_dbg("\rSaving Complex Buffer as CSV");
        SaveCBuff_CSV("Buffer_Complex.csv", ComplexBuffer, SizeOfComplex_Buffer);
        break;
      case 'f':
        SizeOfWorking_Buffer = FFT_SIZE*FFT_SIZE;
        Working_Buffer = mspace_malloc(sdram_msp, SizeOfWorking_Buffer);
        print_dbg("\rReading in Buffer.csv");
        Read_CSV("Buffer.csv", Working_Buffer, SizeOfWorking_Buffer);
        print_dbg("\n\rComplete!");
        break;

      case 'S':
        print_dbg("\rSaving Bitmap;");
        SaveBitmap(image.ImagePtr, image.Width, image.Height, "ResavedImage.bmp");
        print_dbg("\rSaved Bitmap!;");
        break;

      case 'T':
        print_dbg("\rReading in 2D Signal");
        Working_Buffer = mspace_malloc(sdram_msp, FFT_SIZE * FFT_SIZE);
        SizeOfWorking_Buffer = FFT_SIZE * FFT_SIZE;
        Read2DSignal(Working_Buffer);
        break;
      case 'v':
        PrintStatus();
        break;
      case 'o':
        print_dbg("\r1 in Fixed point = ");
        i = DSP16_Q(1);
        print_dbg_ulong(i);
        break;


      default:
        print_dbg("\rCommand Not Recognised;");
```

```
        break;
      }
    }
  }
```

H.1.1.2 Bitmap.c

../Code/The_Columbus/ColumbusTest/src/CustomDevices/Bitmap.c

```
/*
 * Bitmap.c
 *
 * Created: 16/02/2013 23:14:34
 *  Author: hslovett
 */
#include "CustomDevices/CustomDevices.h"

const uint8_t DIBHead[DIBHEADERSIZE] = {  0x7C, 0x00, 0x00, 0x00, //Number of
    bytes
                  0x40, 0x01, 0x00, 0x00, //Width - 320
                  0xF0, 0x00, 0x00, 0x00, //Height - 240
                  0x01, 0x00,             //Planes
                  0x10, 0x00,             //Bits per Pixel
                  0x03, 0x00, 0x00, 0x00, //Compression
                  0x00, 0x58, 0x02, 0x00, //Size of Raw Data
                  0x13, 0x0B, 0x00, 0x00, //Horizontal Resolution
                  0x13, 0x0B, 0x00, 0x00, //Vertical Resolution
                  0x00, 0x00, 0x00, 0x00, //Colours in Palette
                  0x00, 0x00, 0x00, 0x00, //Important Colours
                  0x00, 0xF8, 0x00, 0x00, //Red Mask
                  0xE0, 0x07, 0x00, 0x00, //Green Mask
                  0x1F, 0x00, 0x00, 0x00, //Blue Mask
                  0x00, 0x00, 0x00, 0x00, //Alpha Mask
                  0x01, 0x00, 0x00, 0x00, //Colour Space Type
                  0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                  0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                  0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                  0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                  0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                  0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                  0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                  0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                  0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                  0x00, 0x00, 0x00, 0x00, //Gamma Red
                  0x00, 0x00, 0x00, 0x00, //Gamma Green
                  0x00, 0x00, 0x00, 0x00, //Gamma Blue
                  0x03, 0x00, 0x00, 0x00, //Intent - Photo
                  0x00, 0x00, 0x00, 0x00, //ICC Profile Data
                  0x00, 0x00, 0x00, 0x00, //ICC Profile Size
                  0x00, 0x00, 0x00, 0x00}; //Reserved

const uint8_t BMPHeader[BMPHEADERSIZE] = {  0x42, 0x4D,
                    0x8A, 0x58, 0x02, 0x00, //Size
                    0x00, 0x00, 0x00, 0x00, //Reserved
```

```
                    0x8A, 0x00, 0x00, 0x00 //Offset to Pixel Array
                    };
```

H.1.1.3 CustomDevices.h

../Code/The_Columbus/ColumbusTest/src/CustomDevices/CustomDevices.h

```
/*
 * CustomDevices.h
 *
 * Created: 16/02/2013 14:30:50
 *  Author: hslovett
 */


#ifndef CUSTOMDEVICES_H_
#define CUSTOMDEVICES_H_

//Camera
#include "CustomDevices/OV7670.h"
//I2C Mux
#include "CustomDevices/PCA9542A.h"
//MotorDriver
#include "CustomDevices/MotorDriver.h"
//SDCard
#include "CustomDevices/SD_Card.h"
//Image Processing Functions
#include "CustomDevices/ImageProcessor.h"

typedef struct {
  int Status;
  SD_Status_t *SD_Card;
  Motor_Control_t *Motors;
  OV7670_t *Cameras;
  PCA9542A_t *I2CMux;
} Columbus_Status_t;
typedef struct  {
  char Command;
  int Arg;
} AutoCommand_t;

#define AUTO_COMMAND_LENGTH   14

#define SD_ERR    0x1
#define CAM_ERR   0x2
#define I2CMux_ERR  0x4
#define CAM0_NotFOUND 0x08
#define CAM1_NotFOUND 0x10
#define AutoRunCMD_ERR  0x20
#define FFT_SIZE 64


//REF : http://www.chris.com/ASCII/index.php?art=transportation/nautical
```

```
#define ASCII_SHIP "\t\t\t             |    |    | \n\r\t\t\t            )_)
    )_)  )_) \n\r\t\t\t           )___))___))___)\\  \n\r\t\t\t           )
    ____)____)_____)\\\\ \n\r\t\t\t         _____|____|____|____\\\\\\__ \n\r\
    t\t\t---------\\                  /--------- \n\r\t\t\t  ^^^^^
    ^^^^^^^^^^^^^^^^^^^^^ \n\r\t\t\t    ^^^^      ^^^^     ^^^    ^^\n\r\t\t\t
             ^^^^      ^^^\n\r"
#define COLUMBUS " _______  _______  _                _______  ______
        _______ \n\r(  ____ \\(  ___  )( \\      |\\     /|(       )(  ___ \\
    |\\     /|(  ____ \\\n\r| (    \\/| (   ) || (      | )   ( || () () || (
      ) )| )   ( || (    \\/\n\r| |      | |   | || |      | |   | || || || ||
     (__/ / | |   | || (_____ \n\r| |      | |   | || |      | |   | || |(_)|
    ||  __ (  | |   | |(_____  )\n\r| |      | |   | || |      | |   | || |
    | || (  \\ \\ | |    | |       ) |\n\r| (____/\\| (___) || (____/\\| (___)
    || )   ( || )___) )| (___) |/\\____) |\n\r(_______/(_______)(_______/(
    _______)|/     \\||______/ (_______)\\_______)\n\n\r"
#define THE "\t\t\t_________          _______ \n\r \t\t\t\\__   __/|\\     /|(
      ____ \\ \n\r \t\t\t   ) (   | )   ( || (    \\/\n\r\t\t\t   | |   | (___
    ) || (__    \n\r \t\t\t   | |   |  ___  ||  __)   \n\r \t\t\t   | |   | (
      ) || (      \n\r \t\t\t   | |   | )   ( || (____/\\\n\r \t\t\t   )_(
    |/     \\|(_______/\n\r"
#define PROMPT "\n\r$>"

#define HELP  "\rColumbus Prompt Help:\n\r\
\t? : Shows this prompt\n\r \
\tA : Runs the AutoRun procedure in debug mode\r\n \
\tB : Reads a Bitmap file and prints information \n\r \
\tc : converts the working buffer from integer to fixed point\n\r\
\tC : Converts the working buffer from fixed point to integer\n\r\
\td : Saves the Working Buffer to \"Buffer_results.csv\" \n\r \
\tD : Frees the Memory pointed to by the Working Buffer \n\r \
\tf : Reads \"Buffer.csv\" as a 2D Array of FFT_SIZE by FFT_SIZE\n\r \
\tg : Saves the Complex Buffer to \"Buffer_Complex.csv\" \n\r\
\tk : Prints the Complex Buffer \n\r \
\tm : Computes the Magnitude of the 1D FFT of the Working Buffer \n\r\
\tM : Motor Driving Options, prefix to the following: \n\r \
\t\t F (int) : Drive Robot forward by (int) millimetres (negative number for
    reverse)\n\r\
\t\t L : Dive Left Wheel Forward a full rotation \n\r\
\t\t q : Resets Motors \n\r\
\t\t R : Drive Right Wheel Forward a full rotation \n\r\
\t\t T (int) : Rotate Robot by (int) degrees (postitive turns Clockwise)\n\r\
\to : Displays the fixed point value for (int)1 \n\r\
\tP : Takes and stores Stereo Photos \n\r \
\tr : displays the contents of the working buffer\n\r\
\tR : Reads contents of signal.bin, representing 1D Signal. Integers, Big
    Endian\n\r\
\tT : Reads contents of signal2d.bin, representing 2D Signal. \n\r \
\ts : saves the working buffer\n\r\
\tS : Saves the image in memory to a Bitmaps \n\r\
\tv : Prints the status variables \n\r \
\t1 : computes the One Dimensional FFT of the working buffer. Returns
    magnitude.\n\r\
\t2 : Computes the Magnitude of the Two Dimensional FFT of the Working Buffer.
     \n\r \
\t3 : Computes the Complex 2D FFT of the working buffer and stores it in the
    Complex Buffer \n\r"
```

```
////////////////////////////////////////////////////////////////////////////
// Globals
////////////////////////////////////////////////////////////////////////////
mspace sdram_msp;
Columbus_Status_t Columbus_Status;
//TWI Methods
void twim_init (void);
void System_Test();
#endif /* CUSTOMDEVICES_H_ */
```

H.1.1.4 ImageProcessor.h

../Code/The_Columbus/ColumbusTest/src/CustomDevices/ImageProcessor.h

```
/*
 * ImageProcessor.h
 *
 * Created: 28/02/2013 17:46:37
 *  Author: hslovett
 */


#ifndef IMAGEPROCESSOR_H_
#define IMAGEPROCESSOR_H_

#define BMP_FORMAT_RGB565     1
#define BMP_FORMAT_RGB555     2
#define BMP_FORMAT_GREYSCALE    3
#define BMP_FORMAT_1xUINT     4
#define BMP_FORMAT_2xUINT8T     5
typedef struct {
  uint16_t *ImagePtr;
  int Height;
  int Width;
  uint8_t Format;
  } Image_t;



void FFT1D( int *Signal, dsp16_complex_t *ComplexBuffer);
int FFT2Dabs(int *Signal);
int log_2(int i);

void FFT2DCOMPLEX( int *Signal, dsp16_complex_t *ComplexBuffer, int size );
void PrepareImage(Image_t *Image);
//int* IFFT2D (dsp16_complex_t *Result, dsp16_complex_t *Input);
void IFFT2D (dsp16_complex_t *Signal); /*Need to test this! */;
int FFT1D_Abs( int *Signal);
void Complex_Abs( int *Signal, dsp16_complex_t *ComplexBuffer, int size);
#endif /* IMAGEPROCESSOR_H_ */
```

H.1.1.5 ImageProcessor.c

../Code/The_Columbus/ColumbusTest/src/CustomDevices/ImageProcessor.c

```
/*
 * ImageProcessor.c
 *
 * Created: 28/02/2013 17:46:50
 *  Author: hslovett
 */
#include <asf.h>
#include "CustomDevices/CustomDevices.h"

/*#define FFT_SIZE 64*/
//Returns log base 2 of i - checks if it is an integer power of 2
int log_2(int i)
{
  int ret = 0;
  if((i & (i - 1)) != 0)
  {
    return -1;
  }
  while((i & 1) == 0) //while the bit isn't in the lowest bit (already
    established this is a integer power of 2)
  {
    i >>= 1;
    ret++;
  }

  return ret;
}
//************************************
// Method:    FFT2DCOMPLEX
// FullName:  FFT2DCOMPLEX
// Access:    public
// Returns:   int*
// Qualifier:
// Parameter: int * Signal
// Parameter: A_ALIGNED dsp16_complex_t * ComplexBuffer
// Parameter: int size
//************************************
void FFT2DCOMPLEX( int *Signal, dsp16_complex_t *ComplexBuffer, int size )
{
  int i, j = 0;
  int Ptr;
  Ptr = 0;
  A_ALIGNED dsp16_complex_t Input_C_1D[FFT_SIZE];
  A_ALIGNED dsp16_complex_t Result_C_1D[FFT_SIZE];
  A_ALIGNED dsp16_complex_t Result_C_2D[FFT_SIZE*FFT_SIZE];
  A_ALIGNED dsp16_t Input_R_1D[FFT_SIZE];
  //Stage 1 - FFT Real values from Signal. Store VERTICALLY in Result_2D
  //print_dbg("\n\rStage 1 Reached. Iteration: \n\r");
  for(i = 0; i < FFT_SIZE; i ++){ //for each row
    for(j = 0; j < FFT_SIZE; j++){
      Input_R_1D[j] = Signal[Ptr++]; //copy the data across
    }
    //Do the FFT
```

```
//    print_dbg("\r");
//    print_dbg_ulong(i);

    dsp16_trans_realcomplexfft(Result_C_1D, Input_R_1D, log_2(FFT_SIZE));
    //Copy data into 2D reult TRANSPOSED
    for(j = 0; j < FFT_SIZE; j++){
      Result_C_2D[i + (j * FFT_SIZE)].imag = Result_C_1D[j].imag * FFT_SIZE;// scale back up
      Result_C_2D[i + (j * FFT_SIZE)].real = Result_C_1D[j].real * FFT_SIZE;
    }
  }
  //print_dbg("\n\rStage 2 Reached. Iteration: \n\r");
  //Stage 2 - FFT Complex Values from Result_2D, put back into Rows
  for(i = 0; i < FFT_SIZE; i++){//for each row
    for(j = 0; j < FFT_SIZE; j++){//copy the data across
      Input_C_1D[j].imag = Result_C_2D[j + i * FFT_SIZE].imag;
      Input_C_1D[j].real = Result_C_2D[j + i * FFT_SIZE].real;
    }
//    print_dbg("\r");
//    print_dbg_ulong(i);
    //Do Fourier
    dsp16_trans_complexfft(Result_C_1D, Input_C_1D, log_2(FFT_SIZE));
    //Copy back
    for(j = 0; j < FFT_SIZE; j++){
      ComplexBuffer[i + j * FFT_SIZE].imag =  Result_C_1D[j].imag;
      ComplexBuffer[i + j * FFT_SIZE].real =  Result_C_1D[j].real;
    }
  }
  return;
}

//One Dimensional Fast Fourier Transform
int FFT1D_Abs( int *Signal)
{
  int log2Size, i =0;
  A_ALIGNED dsp16_complex_t vect1[FFT_SIZE];
  A_ALIGNED dsp16_t vect2[FFT_SIZE];
  for(i = 0; i < FFT_SIZE; i++)
  {
    vect2[i] = (dsp16_t)Signal[i];
  }
  dsp16_trans_realcomplexfft(vect1, vect2, log_2(FFT_SIZE));
  for(i = 0; i < FFT_SIZE; i++)
  {
    vect1[i].imag = vect1[i].imag * FFT_SIZE;
    vect1[i].real = vect1[i].real * FFT_SIZE;
  }
  dsp16_vect_complex_abs(vect2, vect1, FFT_SIZE);
  for(i = 0; i < FFT_SIZE; i++)
  {
    Signal[i] = vect2[i];// * FFT_SIZE;
  }

  return Signal;
}

void Complex_Abs( int *Signal, dsp16_complex_t *ComplexBuffer, int size)
{
```

```
  int log2Size, i =0;
  A_ALIGNED dsp16_complex_t vect1[FFT_SIZE];
  A_ALIGNED dsp16_t vect2[FFT_SIZE];
  for(i = 0; i < FFT_SIZE; i++)
  {
    vect1[i].imag = ComplexBuffer[i].imag;
    vect1[i].real = ComplexBuffer[i].real;
  }
  dsp16_vect_complex_abs(vect2, vect1, FFT_SIZE);
  for(i = 0; i < FFT_SIZE; i++)
  {
    Signal[i] = vect2[i];// * FFT_SIZE;
  }
}
//One Dimensional Fast Fourier Transform returning complex values
void FFT1D( int *Signal, dsp16_complex_t *ComplexBuffer)
{
  int log2Size, i =0;
  A_ALIGNED dsp16_complex_t vect1[FFT_SIZE];
  A_ALIGNED dsp16_t vect2[FFT_SIZE];
  for(i = 0; i < FFT_SIZE; i++)
  {
    vect2[i] = Signal[i];
  }
  dsp16_trans_realcomplexfft(vect1, vect2, log_2(FFT_SIZE));
  for(i = 0; i < FFT_SIZE; i++)
  {
    ComplexBuffer[i].imag = vect1[i].imag * FFT_SIZE;
    ComplexBuffer[i].real = vect1[i].real * FFT_SIZE;
  }
}
int FFT2Dabs( int *Signal )
{
  int i, j = 0;
  int Ptr;
  Ptr = 0;
  A_ALIGNED dsp16_complex_t Input_C_1D[FFT_SIZE];
  A_ALIGNED dsp16_complex_t Result_C_1D[FFT_SIZE];
  A_ALIGNED dsp16_complex_t Result_C_2D[FFT_SIZE*FFT_SIZE];
  A_ALIGNED dsp16_t Input_R_1D[FFT_SIZE];

  //Stage 1 - FFT Real values from Signal. Store VERTICALLY in Result_2D
  for(i = 0; i < FFT_SIZE; i ++) //for each row
  {
//    print_dbg("\n\rInput to FFT: \n\r[");
    for(j = 0; j < FFT_SIZE; j++)
    {
      Input_R_1D[j] = Signal[Ptr++]; //copy the data across
//      print_dbg_ulong(Input_R_1D[j]);
//      print_dbg(", ");
    }
//    print_dbg("\b\b]");
    //Do the FFT
    dsp16_trans_realcomplexfft(Result_C_1D, Input_R_1D, log_2(FFT_SIZE));
    //Copy data into 2D reult TRANSPOSED
//    print_dbg("\n\rOutput of FFT:\n\r[");
    for(j = 0; j < FFT_SIZE; j++)
    {
```

```
      Result_C_2D[i + (j * FFT_SIZE)].imag = Result_C_1D[j].imag * FFT_SIZE;//
    scale back up
      Result_C_2D[i + (j * FFT_SIZE)].real = Result_C_1D[j].real * FFT_SIZE;
//      print_dbg_ulong(Result_C_2D[i + (j * FFT_SIZE)].real);
//      print_dbg(" + j");
//      print_dbg_ulong(Result_C_2D[i + (j * FFT_SIZE)].imag);
//      print_dbg(" , ");
    }
//    print_dbg("\b\b]");
  }
  //Stage 2 - FFT Complex Values from Result_2D, put back into Rows

  for(i = 0; i < FFT_SIZE; i++)//for each row
  {
//    print_dbg("\n\rInput to FFT: \n\r[");
    for(j = 0; j < FFT_SIZE; j++)//copy the data across
    {
      Input_C_1D[j].imag = Result_C_2D[j + i * FFT_SIZE].imag;
      Input_C_1D[j].real = Result_C_2D[j + i * FFT_SIZE].real;
//      print_dbg_ulong(Input_C_1D[j].real);
//      print_dbg(" + j");
//      print_dbg_ulong(Input_C_1D[j].imag);
//      print_dbg(" , ");
    }
//    print_dbg("\b\b]");
    //Do Fourier
    dsp16_trans_complexfft(Result_C_1D, Input_C_1D, log_2(FFT_SIZE));
    //Copy back

//    print_dbg("\n\rOutput to FFT: \n\r[");
//    for(j = 0; j < FFT_SIZE; j++)//copy the data across
//    {
//      print_dbg_ulong(Result_C_1D[j].real);
//      print_dbg(" + j");
//      print_dbg_ulong(Result_C_1D[j].imag);
//      print_dbg(" , ");
//    }
//    print_dbg("\b\b]");
    //Calculate Abs and put back into Signal TRANSPOSED
    dsp16_vect_complex_abs(Input_R_1D, Result_C_1D, FFT_SIZE);

    for(j = 0; j < FFT_SIZE; j++)
    {
      Signal[i + (j*FFT_SIZE)] = Input_R_1D[j] * FFT_SIZE;
    }
  }
  return Signal;
}




void PrepareImage(Image_t *Image)
{
  int row, col;
  uint16_t *PreparedImage;
  //Allocate some memory in the RAM
  PreparedImage = mspace_malloc(sdram_msp, 256*256 );
```

```
  //print_dbg("\n\rPrepared Image Pointer = ");
  //print_dbg_ulong(PreparedImage);
  for(row = 0; row < 256; row ++)
  {
    for(col = 0; col < 256; col++)
    {
      if(row < 240)
        PreparedImage[row*256 + col] = Image->ImagePtr[row * 256 + col];
      else
        PreparedImage[row *256 + col] = 0;//Image->ImagePtr[(row - 240) * 256 + col + 32];
    }
  }

  mspace_free(sdram_msp, Image->ImagePtr); //free up the old image
  Image->ImagePtr = PreparedImage; //move the pointer to the prepared image
  Image->Height = 256;
  Image->Width = 256;
  //SaveBitmap(PreparedImage, 256, 256, "PreparedImage.bmp");
  //mspace_free(sdram_msp, PreparedImage);
  //return PreparedImage;
}


//************************************
// Method:    IFFT2D
// FullName:  IFFT2D
// Access:    public
// Returns:   void
// Qualifier:
// Parameter: dsp16_complex_t * Signal
//************************************
void IFFT2D (dsp16_complex_t *Signal) //Need to test this!
{
  int i, j = 0;
  int Ptr;
  Ptr = 0;
  A_ALIGNED dsp16_complex_t Input_C_1D[FFT_SIZE];
  A_ALIGNED dsp16_complex_t Result_C_1D[FFT_SIZE];
  A_ALIGNED dsp16_complex_t Result_C_2D[FFT_SIZE*FFT_SIZE];
  A_ALIGNED dsp16_t Input_R_1D[FFT_SIZE];


  //Stage 1 - FFT Real values from Signal. Store VERTICALLY in Result_2D
  for(i = 0; i < FFT_SIZE; i ++) //for each row
  {
    for(j = 0; j < FFT_SIZE; j++)
    {
      Input_C_1D[j].real = Signal[Ptr].real; //copy the data across
      Input_C_1D[j].imag = Signal[Ptr].imag;
    }

    //Do the FFT
    dsp16_trans_complexifft(Result_C_1D, Input_C_1D, log_2(FFT_SIZE));
    //Copy data into 2D result TRANSPOSED

    for(j = 0; j < FFT_SIZE; j++)
```

```
    {
      Result_C_2D[i + (j * FFT_SIZE)].imag = Result_C_1D[j].imag * FFT_SIZE;// scale back up
      Result_C_2D[i + (j * FFT_SIZE)].real = Result_C_1D[j].real * FFT_SIZE;
    }

  }
  //Stage 2 - FFT Complex Values from Result_2D, put back into Rows


  for(i = 0; i < FFT_SIZE; i++)//for each row
  {

    for(j = 0; j < FFT_SIZE; j++)//copy the data across
    {
      Input_C_1D[j].imag = Result_C_2D[j + i * FFT_SIZE].imag;
      Input_C_1D[j].real = Result_C_2D[j + i * FFT_SIZE].real;

    }

    //Do Fourier
    dsp16_trans_complexifft(Result_C_1D, Input_C_1D, log_2(FFT_SIZE));
    //Copy back


    //Put back into Signal TRANSPOSED
    //dsp16_vect_complex_abs(Input_R_1D, Result_C_1D, FFT_SIZE);

    for(j = 0; j < FFT_SIZE; j++)
    {
      Signal[i + j * FFT_SIZE].imag =   Result_C_1D[j].imag;
      Signal[i + j * FFT_SIZE].real =   Result_C_1D[j].real;
      //Signal[i + (j*FFT_SIZE)] = Input_R_1D[j] * FFT_SIZE;
    }
  }

  //return Signal;
}

void ComplexMultiply(dsp16_complex_t *Result_Input1, dsp16_complex_t *Input2, int size)
{
  int i = 0;
  dsp16_complex_t c;
  for(i = 0; i < size; i++)
  {
    //(a+jb).(c+jd) = (ac - bd) + j(ad + bc)
    c.real = (Result_Input1[i].real * Input2[i].real) - (Result_Input1[i].imag * Input2[i].imag);
    c.imag = (Result_Input1[i].real * Input2[i].imag) + (Result_Input1[i].imag * Input2[i].real);
    Result_Input1[i].imag = c.imag;
    Result_Input1[i].real = c.real;
  }
}
```

H.1.1.6 MotorDriver.h

../Code/The_Columbus/ColumbusTest/src/CustomDevices/MotorDriver.h

```
/*
 * MotorDriver.h
 *
 * Created: 10/02/2013 18:11:55
 *  Author: hslovett
 */


#ifndef MOTORDRIVER_H_
#define MOTORDRIVER_H_
#include <asf.h>
//Definitions
#define MOTOR_L     ML_PWM_CHANNEL_ID
#define MOTOR_R   MR_PWM_CHANNEL_ID

#define FORWARD     2
#define BACKWARD    3
#define LEFT_SPOT   4
#define RIGHT_SPOT    5
#define SPOT_PIVOT    6
#define STOP      7


#define ENABLE_ACA_INTERRUPT    //{AVR32_ACIFA1.ier = 1;}
#define DISABLE_ACA_INTERRUPT   //{AVR32_ACIFA1.idr = 1;}
#define ENABLE_ACB_INTERRUPT    //{AVR32_ACIFA1.ier = 2;}
#define DISABLE_ACB_INTERRUPT   //{AVR32_ACIFA1.idr = 2;}
#define GAMMA 10  //Interrupts caused per full rotation of a wheel
#define CIRCUMFERENCE_WHEEL_MM    116 //in millimeters
//#define CIRCUMFERENCE_WHEEL_CM    12 //in centimeters
#define MIN_DISTANCE_RESOLUTION   CIRCUMFERENCE_WHEEL_MM / GAMMA
#define MIN_ROTATION_RESOLUTION   (CIRCUMFERENCE_WHEEL_MM*360)/(GAMMA * C_b)
#define C_b             282
#define ROTATION_CONST        (GAMMA * C_b) / (CIRCUMFERENCE_WHEEL_MM * 360)
#define ROTATION_CONST_INV        (CIRCUMFERENCE_WHEEL_MM * 360) / (GAMMA *
    C_b)
//Type Defs
typedef struct {
  int Left_State;
  int Right_State;
  int Left_Count;
  int Right_Count;
  } Motor_Control_t;

//Globals
pwm_opt_t pwm_opt;                // PWM option config.
avr32_pwm_channel_t pwm_channel;
Motor_Control_t Motor_Control;

void Motor_Init();
void Motor_Go();
//void Analogue_Comparator_Init();
void Motor_Start(int Motors);
```

```
void Motors_Reset(void);
void Motor_Stop(int Motors);
bool Motors_Moving();
int Motors_Move(int centimetres_fwd)/*Move this amount forward in centimeters */;
void Motors_Execute();
int Motors_Rotate(int angle_degs);
/*static void ACInterruptHandler(void);*/
#endif /* MOTORDRIVER_H_ */
```

H.1.1.7 MotorDriver.c

../Code/The_Columbus/ColumbusTest/src/CustomDevices/MotorDriver.c

```
/*
 * MotorDriver.c
 *
 * Created: 10/02/2013 18:12:07
 *  Author: hslovett
 */
#include <asf.h>
#include "CustomDevices/CustomDevices.h"
#include <delay.h>



static void local_start_highfreq_clock(void)
{
  const scif_pll_opt_t opt = {
          .osc = SCIF_OSC0,     // Sel Osc0/PLL0 or Osc1/PLL1
          .lockcount = 16,      // lockcount in main clock for the PLL wait lock
          .div = 1,             // DIV=1 in the formula
          .mul = 6,             // MUL=7 in the formula
          .pll_div2 = 1,        // pll_div2 Divide the PLL output frequency by 2 (this settings does not change the FVCO value)
          .pll_wbwdisable = 0,  // pll_wbwdisable 1 Disable the Wide-Bandith Mode (Wide-Bandwith mode allow a faster startup time and out-of-lock time). 0 to enable the Wide-Bandith Mode.
          .pll_freq = 1,        // Set to 1 for VCO frequency range 80-180MHz, set to 0 for VCO frequency range 160-240Mhz.
  };
  // Switch main clock to Osc0.
//  pcl_switch_to_osc(PCL_OSC0, FOSC0, OSC0_STARTUP);

  /* Setup PLL0 on Osc0, mul=7 ,no divisor, lockcount=16, ie. (16Mhzx7)/(div2) = 56MHz output */
  scif_pll_setup(SCIF_PLL0, &opt); // lockcount in main clock for the PLL wait lock

  /* Enable PLL0 */
  scif_pll_enable(SCIF_PLL0);

  /* Wait for PLL0 locked */
```

```
  scif_wait_for_pll_locked(SCIF_PLL0) ;
}
static void pwm_start_gc(void)
{
  scif_gc_setup(AVR32_SCIF_GCLK_PWM,
                SCIF_GCCTRL_PLL0,
                AVR32_SCIF_GC_NO_DIV_CLOCK,
                0);
  scif_gc_enable(AVR32_SCIF_GCLK_PWM);
}
void Analogue_Comparator_Init()
{
  static const gpio_map_t ACIFA_GPIO_MAP =
  {
  {POT0_AC1AP1_PIN, POT0_AC1AP1_FUNCTION},
  {POT1_AC1BP1_PIN, POT1_AC1BP1_FUNCTION},
  {SENSE0_AC1AN1_PIN, SENSE0_AC1AN1_FUNCTION},
  {SENSE1_AC1BN1_PIN, SENSE1_AC1BN1_FUNCTION},
  };

  gpio_enable_module(ACIFA_GPIO_MAP, sizeof(ACIFA_GPIO_MAP) / sizeof(
    ACIFA_GPIO_MAP[0]));
  //Make it an interrupt
  Disable_global_interrupt();

  acifa_configure_hysteresis(&AVR32_ACIFA1, ACIFA_COMP_SELA, 2);
  acifa_configure(&AVR32_ACIFA1,
  ACIFA_COMP_SELA,
  POT0_AC1AP1_INPUT,
  SENSE0_AC1AN1_INPUT,
  FOSC0);

  acifa_configure_hysteresis(&AVR32_ACIFA1, ACIFA_COMP_SELB, 2);
  acifa_configure(&AVR32_ACIFA1,
  ACIFA_COMP_SELB,
  POT1_AC1BP1_INPUT,
  SENSE1_AC1BN1_INPUT,
  FOSC0);

  //  acifa_enable_interrupt(&AVR32_ACIFA1, (1 << AVR32_ACIFA_ACBINT )| (1 <<
    AVR32_ACIFA_ACAINT));//Enable ACBINT and ACAINT
  //  AVR32_ACIFA1.ier = 3; //enable interrupts
  //  acifa_enable_interrupt_toggle(&AVR32_ACIFA1, ACIFA_COMP_SELA);
  //  acifa_enable_interrupt_toggle(&AVR32_ACIFA1, ACIFA_COMP_SELB);
  //  acifa_enable_interrupt_inp_lower(&AVR32_ACIFA1, ACIFA_COMP_SELA);
  //  acifa_enable_interrupt_inp_lower(&AVR32_ACIFA1, ACIFA_COMP_SELB);
  acifa_start(&AVR32_ACIFA1, (ACIFA_COMP_SELA|ACIFA_COMP_SELB));

  //  INTC_register_interrupt(&ACInterruptHandler,AVR32_ACIFA1_IRQ ,
    AVR32_INTC_INT0);

  Enable_global_interrupt();
}
void Motor_Init()
{
  //Turn boths motors off
  ML_STANDBY;
  MR_STANDBY;
```

```
    ML_IN1_CLR;
    ML_IN2_CLR;

    MR_IN1_CLR;
    MR_IN2_CLR;

    Motor_Control.Left_Count = 0;
    Motor_Control.Right_Count = 0;
    Motor_Control.Left_State = STOP;
    Motor_Control.Right_State = STOP;

    avr32_pwm_channel_t pwm_channel = { {0}, // cmr
                      {0}, // cdty
                      {0}, // cdtyupd
                      {0}, // cprd
                      {0}, // cprdupd
                      {0}, // ccnt
                      {0}, // dt
                      {0}};// dtupd  ;  One channel config.
  /*  unsigned int channel_id;*/

    // Start PLL for PWM
    local_start_highfreq_clock();
    // Start Enable Generic Clock with PLL as source clock
    pwm_start_gc();
    gpio_enable_module_pin(ML_PWM_H_PIN, ML_PWM_H_FUNCTION);
    gpio_enable_module_pin(MR_PWM_H_PIN, MR_PWM_H_FUNCTION);
    // PWM controller configuration.
    pwm_opt.diva = AVR32_PWM_DIVA_CLK_OFF;
    pwm_opt.divb = AVR32_PWM_DIVB_CLK_OFF;
    pwm_opt.prea = AVR32_PWM_PREA_CCK;
    pwm_opt.preb = AVR32_PWM_PREB_CCK;
    pwm_opt.fault_detection_activated = false;
    pwm_opt.sync_channel_activated    = true;
    pwm_opt.sync_update_channel_mode  =
      PWM_SYNC_UPDATE_MANUAL_WRITE_MANUAL_UPDATE;
    pwm_opt.sync_channel_select[0]    = false;
    pwm_opt.sync_channel_select[1]    = false;
    pwm_opt.sync_channel_select[2]    = false;
    pwm_opt.sync_channel_select[3]    = false;
    pwm_opt.cksel                     = PWM_CKSEL_GCLK;
    pwm_init(&pwm_opt);
    // Update the period
    pwm_update_period_value(10);
    // Channel configuration
    pwm_channel.CMR.dte   = 0;             // Enable Deadtime for complementary
       Mode
    pwm_channel.CMR.dthi  = 0;             // Deadtime Inverted on PWMH
    pwm_channel.CMR.dtli  = 0;             // Deadtime Not Inverted on PWML
    pwm_channel.CMR.ces   = 0;             // 0/1 Channel Event at the End of
      PWM Period
    pwm_channel.CMR.calg  = PWM_MODE_LEFT_ALIGNED;      // Channel mode.
    pwm_channel.CMR.cpol  = PWM_POLARITY_HIGH;          // Channel polarity.
    pwm_channel.CMR.cpre  = AVR32_PWM_CPRE_CCK;         // Channel prescaler.
    pwm_channel.cdty      = 50;            // Channel duty cycle, should be <
      CPRD.
    pwm_channel.cprd      = 200;           // Channel period.
```

```
  pwm_channel_init(ML_PWM_CHANNEL_ID, &pwm_channel); // Set channel
    configuration to channel 0
  pwm_channel_init(MR_PWM_CHANNEL_ID, &pwm_channel); // Set channel
    configuration to channel 0
  Analogue_Comparator_Init();
}

// __attribute__((__interrupt__)) static void ACInterruptHandler(void)
// {
//
// }

void Motor_Start(int Motors)
{
  if(Motors & MOTOR_L)
  {
    if(Motor_Control.Left_State == FORWARD)
    {
      ML_IN1_SET;
      ML_IN2_CLR;
    }
    else if (Motor_Control.Left_State == BACKWARD)
    {
      ML_IN1_CLR;
      ML_IN2_SET;
    }
    else //Somethings gone wrong
    {
      ML_IN1_CLR;
      ML_IN2_CLR;
      return;//don't start any pwm channel
    }
    ML_GO;
    pwm_start_channels((1 << MOTOR_L)); //Start PWM Channel on M0 line
  }

  if(Motors & MOTOR_R)
  {
    if(Motor_Control.Right_State == FORWARD)
    {
      MR_IN1_SET;
      MR_IN2_CLR;
    }
    else if (Motor_Control.Right_State == BACKWARD)
    {
      MR_IN1_CLR;
      MR_IN2_SET;
    }
    else //Somethings gone wrong
    {
      MR_IN1_CLR;
      MR_IN2_CLR;
      return;//don't start any pwm channel
    }
    MR_GO;
    pwm_start_channels((1 << MOTOR_R));
  }
```

```
}

#define HYST_MAX  10
#define SAMPLE_TIME 10
void Motors_Execute()
{
  LEDMOTOR_SET;
  int Left_State = acifa_is_acb_inp_higher(&AVR32_ACIFA1);
  int Right_State = acifa_is_aca_inp_higher(&AVR32_ACIFA1);
  int Left_Hyst = 0;
  int Right_Hyst = 0;
  bool Left_OnTab = false;
  bool Right_OnTab = false;

  while(Motors_Moving())
  {
    ///////////////////////////////////////////////////////
    // LEFT MOTOR ////////////////////////////////////////
    ///////////////////////////////////////////////////////
    Left_OnTab = acifa_is_acb_inp_higher(&AVR32_ACIFA1); //Read the status of the comparator

    if(Left_State == 0) //State Not on Tab
    {
      LED5_CLR;
      if(Left_OnTab == true)
      {
        Left_Hyst++;
        if(Left_Hyst == HYST_MAX)//reached threshold, change state
        {
          Left_State = 1;
          Motor_Control.Left_Count--; //Decrement the global counter
          Left_Hyst = 0;
          print_dbg("\n\rLeft Count Decrement");
        }
      }
      else //Reset counter
      {
        Left_Hyst = 0;
      }
    }
    else //State On Tab
    {
      LED5_SET;
      if(Left_OnTab == true)//still on a tab
      {
        Left_Hyst = 0;//Reset counter
      }
      else
      {
        Left_Hyst ++;
        if(Left_Hyst == HYST_MAX)//Reached Threshold. Change state
        {
          Left_State = 0;
          Left_Hyst = 0;
        }
      }
    }
```

```
    ////////////////////////////////////////////////////////
    // RIGHT MOTOR ////////////////////////////////////////
    ////////////////////////////////////////////////////////
    Right_OnTab = acifa_is_aca_inp_higher(&AVR32_ACIFA1); //Read the status of
     the comparator

    if(Right_State == 0) //State Not on Tab
    {
      LED6_CLR;
      if(Right_OnTab == true)
      {
        Right_Hyst++;
        if(Right_Hyst == HYST_MAX)//reached threshold, change state
        {
          Right_State = 1;
          Motor_Control.Right_Count--; //Decrement the global counter
          Right_Hyst = 0;
          print_dbg("\n\rRight Count Decrement");
        }
      }
      else //Reset counter
      {
        Right_Hyst = 0;
      }
    }
    else //State On Tab
    {
      LED6_SET;
      if(Right_OnTab == true)//still on a tab
      {
        Right_Hyst = 0;//Reset counter
      }
      else
      {
        Right_Hyst ++;
        if(Right_Hyst == HYST_MAX)//Reached Threshold. Change state
        {
          Right_State = 0;
          Right_Hyst = 0;
        }
      }
    }
    int temp = 0;
    if(Motor_Control.Left_Count <= 0) //if we have reached the end of the
    movement on left wheel
      temp |= MOTOR_L;

    if(Motor_Control.Right_Count <= 0)
      temp |= MOTOR_R;
    if(temp != 0)
      Motor_Stop(temp); //Stop the Motor

    //Delay to keep a low sample time
    delay_ms(SAMPLE_TIME);
  }
  LEDMOTOR_CLR;
}
void Motor_Stop(int Motors)
```

```
{
  if(Motors & MOTOR_L)
  {
    ML_STANDBY;
    Motor_Control.Left_State = STOP;
    pwm_stop_channels((1 << MOTOR_L)); //Start PWM Channel on M0 line
  }

  if(Motors & MOTOR_R)
  {
    MR_STANDBY;
    Motor_Control.Right_State = STOP;
    pwm_stop_channels((1 << MOTOR_R));
  }
}
int Motors_Move(int millimetres_fwd)//Move this amount forward in centimeters
{
  //Calculate number of interrupts of each wheel
  int number_interrupts;
  int distance_moved;
  if(millimetres_fwd > 0)
  {
    Motor_Control.Left_State = FORWARD;
    Motor_Control.Right_State = FORWARD;
  }
  else
  {
    millimetres_fwd = Abs(millimetres_fwd);
    Motor_Control.Left_State = BACKWARD;
    Motor_Control.Right_State = BACKWARD;
  }
  number_interrupts = (millimetres_fwd * (int)GAMMA) / (int)
    CIRCUMFERENCE_WHEEL_MM;
  distance_moved = number_interrupts * MIN_DISTANCE_RESOLUTION;
  print_dbg("\n\rNumber of interrupts to move = ");
  print_dbg_ulong(number_interrupts);

  Motor_Control.Left_Count = number_interrupts;
  Motor_Control.Right_Count = number_interrupts;
  Motor_Start(MOTOR_L | MOTOR_R);
  Motors_Execute();
  return distance_moved;
}

void Motors_Reset(void)
{
  Motor_Control.Left_State = FORWARD;
  Motor_Control.Left_Count = 1;
  Motor_Control.Right_State = FORWARD;
  Motor_Control.Right_Count = 1;
  Motor_Start(MOTOR_L | MOTOR_R);
  Motors_Execute();
}

bool Motors_Moving()
{
//  if(Motor_Control.Left_State != STOP)
//  {
```

```
//     if(Motor_Control.Right_State != STOP)
//     {
//       return true;
//     }
//     else
//       return false;
//   }
//   else
//   {
//     return false;
//   }
  if(Motor_Control.Left_State != STOP) //Left is moving
  {
    return true;
  }
  else if (Motor_Control.Right_State != STOP) //Right is moving
  {
    return true;
  }
  else
  {
    return false;
  }
}


int Motors_Rotate(int angle_degs)
{
  int interrupts_to_move = 0;
  int angle_rotated;
  //calculate interrupts to move
  interrupts_to_move = angle_degs * ROTATION_CONST;
  angle_rotated = interrupts_to_move * ROTATION_CONST_INV;

  //Both Wheels Move
  if(interrupts_to_move > 0)
  {
    Motor_Control.Left_State = FORWARD;
    Motor_Control.Right_State = BACKWARD;
  }
  else
  {
    Motor_Control.Right_State = FORWARD;
    Motor_Control.Left_State = BACKWARD;
  }
  Motor_Control.Left_Count = Abs(interrupts_to_move);
  Motor_Control.Right_Count = Abs(interrupts_to_move);
  Motor_Start(MOTOR_L | MOTOR_R);
  Motors_Execute();
  return angle_rotated;

}
```

H.1.1.8 OV7670.h

../Code/The_Columbus/ColumbusTest/src/CustomDevices/OV7670.h

```
/*
 * OV7670.h
 *
 * Created: 15/02/2013 13:12:00
 *  Author: hslovett
 */


#ifndef OV7670_H_
#define OV7670_H_
#include <asf.h>
////////////////////////////////////////////////////////////////////////////
//  Constants
////////////////////////////////////////////////////////////////////////////
#define HEIGHT        240
#define WIDTH       320
#define PIXELSIZE     2
#define SETTINGS_LENGTH   167
#define OV7670_ADDR     0x21

#define CAMERA_LEFT     1
#define CAMERA_RIGHT    2

#define CAMERA_LEFT_ERR   0x10
#define CAMERA_RIGHT_ERR  0x20

#define BMPHEADERSIZE 14
#define DIBHEADERSIZE 124 //v5
#define FILESIZE 153738
////////////////////////////////////////////////////////////////////////////
//  Globals
////////////////////////////////////////////////////////////////////////////
const char default_settings[SETTINGS_LENGTH][2];
const uint8_t DIBHead[DIBHEADERSIZE];
const uint8_t BMPHeader[BMPHEADERSIZE];
typedef struct {
  uint8_t Status;
  bool Camera_0_Found;
  bool Camera_1_Found;
  bool Camera_0_Error;
  bool Camera_1_Error;
  uint8_t VSYNC0_State;
  uint8_t VSYNC1_State;
  uint ImageCount;
  } OV7670_t ;

OV7670_t OV7670_Status;

#define IDLE      0
#define TAKE_PHOTO    1
#define TAKING_PHOTO  2
#define TAKEN_PHOTO   3
#define CAMERAS_BUSY  4

#define Image0Name  "Image_L_%d.bmp"
```

```
#define Image1Name  "Image_R_%d.bmp"
/////////////////////////////////////////////////////////////////////////
//  Methods
/////////////////////////////////////////////////////////////////////////
void OV7670_Init(void);              //Initialises Camera
void FIFO_Init();
int TakePhoto(uint8_t Cameras);
bool Photos_Ready(void);
void Store_Image_0();
void Store_Image_1();
void FIFO_Reset(uint8_t CameraID);
bool Store_Both_Images();
//void FIFO_Reset(uint8_t CameraID);
/////////////////////////////////////////////////////////////////////////
//  Pins & Macros
/////////////////////////////////////////////////////////////////////////
#define FIFO_0_RCLK   AVR32_PIN_PA19
#define FIFO_0_nRRST  AVR32_PIN_PA15
#define FIFO_0_WEN    AVR32_PIN_PA20
#define FIFO_0_WRST   AVR32_PIN_PA21
#define FIFO_0_nOE    AVR32_PIN_PA28
#define FIFO_0_VSYNC  AVR32_PIN_PA29

#define FIFO_1_RCLK   AVR32_PIN_PA23
#define FIFO_1_nRRST  AVR32_PIN_PA22
#define FIFO_1_WEN    AVR32_PIN_PA24
#define FIFO_1_WRST   AVR32_PIN_PA25
#define FIFO_1_nOE    AVR32_PIN_PA27

#define VSYNC_1_PIN         AVR32_EIC_EXTINT_1_2_PIN
#define VSYNC_1_FUNCTION      AVR32_EIC_EXTINT_1_2_FUNCTION
#define VSYNC_1_LINE        1
#define VSYNC_1_ENABLE_INTERRUPT  {eic_enable_interrupt_line(&AVR32_EIC,
    VSYNC_1_LINE);}
#define VSYNC_1_DISABLE_INTERRUPT {eic_disable_interrupt_line(&AVR32_EIC,
    VSYNC_1_LINE);}

#define VSYNC_0_PIN         AVR32_EIC_EXTINT_4_0_PIN
#define VSYNC_0_FUNCTION      AVR32_EIC_EXTINT_4_0_FUNCTION
#define VSYNC_0_LINE        4
#define VSYNC_0_ENABLE_INTERRUPT  {eic_enable_interrupt_line(&AVR32_EIC,
    VSYNC_0_LINE);}
#define VSYNC_0_DISABLE_INTERRUPT {eic_disable_interrupt_line(&AVR32_EIC,
    VSYNC_0_LINE);}


#define FIFO_0_RCLK_SET   {gpio_set_gpio_pin(FIFO_0_RCLK);}
#define FIFO_0_nRRST_SET  {gpio_set_gpio_pin(FIFO_0_nRRST);}
#define FIFO_0_WEN_SET    {gpio_set_gpio_pin(FIFO_0_WEN);}
#define FIFO_0_WRST_SET   {gpio_set_gpio_pin(FIFO_0_WRST);}
#define FIFO_0_nOE_SET    {gpio_set_gpio_pin(FIFO_0_nOE);}

#define FIFO_0_RCLK_CLR   {gpio_clr_gpio_pin(FIFO_0_RCLK);}
#define FIFO_0_nRRST_CLR  {gpio_clr_gpio_pin(FIFO_0_nRRST);}
#define FIFO_0_WEN_CLR    {gpio_clr_gpio_pin(FIFO_0_WEN);}
#define FIFO_0_WRST_CLR   {gpio_clr_gpio_pin(FIFO_0_WRST);}
#define FIFO_0_nOE_CLR    {gpio_clr_gpio_pin(FIFO_0_nOE);}
```

```
#define FIFO_1_RCLK_SET   {gpio_set_gpio_pin(FIFO_1_RCLK);}
#define FIFO_1_nRRST_SET  {gpio_set_gpio_pin(FIFO_1_nRRST);}
#define FIFO_1_WEN_SET    {gpio_set_gpio_pin(FIFO_1_WEN);}
#define FIFO_1_WRST_SET   {gpio_set_gpio_pin(FIFO_1_WRST);}
#define FIFO_1_nOE_SET    {gpio_set_gpio_pin(FIFO_1_nOE);}

#define FIFO_1_RCLK_CLR   {gpio_clr_gpio_pin(FIFO_1_RCLK);}
#define FIFO_1_nRRST_CLR  {gpio_clr_gpio_pin(FIFO_1_nRRST);}
#define FIFO_1_WEN_CLR    {gpio_clr_gpio_pin(FIFO_1_WEN);}
#define FIFO_1_WRST_CLR   {gpio_clr_gpio_pin(FIFO_1_WRST);}
#define FIFO_1_nOE_CLR    {gpio_clr_gpio_pin(FIFO_1_nOE);}


#define CAMERA_INPUT  {(uint8_t)((AVR32_GPIO.port[1].pvr) & 0xFF);}
////////////////////////////////////////////////////////////////////////////
//Camera Register Address definitions
////////////////////////////////////////////////////////////////////////////
#define OV_GAIN      0x00  //Gain Control Setting - ACG[7:0]
#define OV_BLUE      0x01  //Blue Channel Gain
#define OV_RED       0x02  //Red Channel Gain
#define OV_VREF      0x03  //Vertical Frame Control & ACG[9:8]
#define OV_COM1      0x04  //CCIR656 enable, AEC low bits (AECHH, AECH)
#define OV_BAVE      0x05  //U/B Average level - AUTO UPDATED
#define OV_GbAVE     0x06  //Y/Gb Average Level - AUTO UPDATED
#define OV_AECHH     0x07  //Exposure value [15:10] (AECH, COM1)
#define OV_RAVE      0x08  //V/R Average level - AUTO UPDATED
#define OV_COM2      0x09  //Soft Sleep, Output drive capability
#define OV_PID       0x0A  //Product ID MSB  Read only
#define OV_VER       0x0B  //Product ID LSB  Read Only
#define OV_COM3      0x0C  //Output data MSB/LSB swap + other stuff
#define OV_COM4      0x0D  //Average values - MUST BE SAME AS COM17
#define OV_COM5      0x0E  //RESERVED
#define OV_COM6      0x0F  //COM6
#define OV_AECH      0x10  //Exposure value [9:2] (see AECHH, COM1)
#define OV_CLKRC     0x11  //Internal Clock options
#define OV_COM7      0x12  //RESET, Output format
#define OV_COM8      0x13  //Common control 8
#define OV_COM9      0x14  //Automatic Gain Ceiling
#define OV_COM10     0x15  //PCLK, HREF and VSYNC options
#define OV_RSVD      0x16  //RESERVED
#define OV_HSTART    0x17  //Output format Horizontal Frame start
#define OV_HSTOP     0x18  //Output format Horizontal Frame end
#define OV_VSTRT     0x19  //Output format Vertical Frame start
#define OV_VSTOP     0x1A  //Output format Vertical Frame Stop
#define OV_PSHFT     0x1B  //Pixel Delay Select
#define OV_MIDH      0x1C  //Manufacturer ID MSB -   READ ONLY
#define OV_MIDL      0x1D  //Manufacturer ID LSB -   READ ONLY
#define OV_MVFP      0x1E  //Mirror / VFlip Enable
#define OV_LAEC      0x1F  //RESERVED
#define OV_ADCCTR0     0x20  //ADC Control
#define OV_ADCCTR1     0x21  //RESERVED
#define OV_ADCCTR2     0x22  //RESERVED
#define OV_ADCCTR3     0x23  //RESERVED
#define OV_AEW       0x24  //ACG/AEC Stable Operating Region Upper Limit
#define OV_AEB       0x25  //ACG/AEC Stable Operation Region Lower Limit
#define OV_VPT       0x26  //ACG/AEC Fast Mode Operation Region
```

```
#define OV_BBIAS    0x27  //B Channel Signal Output Bias
#define OV_GbBIAS   0x28  //Gb Channel Output Bias
#define OV_RSVD1    0x29  //RESERVED
#define OV_EXHCH    0x2A  //Dummy Pixel Insert MSB
#define OV_EXHCL    0x2B  //Dummy Pixel Insert LSB
#define OV_RBIAS    0x2C  //R Channel Signal Output Bias
#define OV_ADVFL    0x2D  //LSB of insert dummy line in vertical direction
#define OV_ADVFH    0x2E  //MSB of insert dummy line in vertical direction
#define OV_YAVE     0x2F  //Y/G Channel Average Value
#define OV_HSYST    0x30  //HSYNC Rising Edge Delay (low 8 bits)
#define OV_HSYEN    0x31  //HSYNCE Falling Edge Delay (low 8 bits)
#define OV_HREF     0x32  //HREF Control
#define OV_CHLF     0x33  //Array Current Control - RESERVED
#define OV_ARBLM    0x34  //Array Reference Control - RESERVED
#define OV_RSVD2    0x35  //RESERVED
#define OV_RSVD3    0x36  //RESERVED
#define OV_ADCCTRL    0x37  //ADC Control - RESERVED
#define OV_ACOM     0x38  //ADC and Analog Common Mode Control - RESERVED
#define OV_OFON     0x39  //ADC Offset Control
#define OV_TSLB     0x3A  //Line Buffer Test Option
#define OV_COM11    0x3B  //COM11
#define OV_COM12    0x3C  //COM12
#define OV_COM13    0x3D  //COM13
#define OV_COM14    0x3E  //COM14
#define OV_EDGE     0x3F  //Edge Detection Adjustment
#define OV_COM15    0x40  //COM15
#define OV_COM16    0x41  //COM16
#define OV_COM17    0x42  //COM17
#define OV_AWBC1    0x43
#define OV_AWBC2    0x44
#define OV_AWBC3    0x45
#define OV_AWBC4    0x46
#define OV_AWBC5    0x47
#define OV_AWBC6    0x48
#define OV_RSVD4    0x49
#define OV_RSVD5    0x40
#define OV_RSVD6    0x4A
#define OV_REG4B    0x4B
#define OV_DNSTH    0x4C
#define OV_RSVD7    0x4D
#define OV_RSVD8    0x4E
#define OV_MTX1     0x4F
#define OV_MTX2     0x50
#define OV_MTX3     0x51
#define OV_MTX4     0x52
#define OV_MTX5     0x53
#define OV_MTX6     0x54
#define OV_BRIGHT   0x55
#define OV_CONTRAS    0x56
#define OV_CONTRASCNTR  0x57
#define OV_MTXS     0x58
#define OV_RSVD9    0x59
#define OV_RSVD9_1    0x5A
#define OV_RSVD9_2    0x5B
#define OV_RSVD9_3    0x5C
#define OV_RSVD9_4    0x5D
#define OV_RSVD9_5    0x5E
#define OV_RSVD9_6    0x5F
```

```
#define OV_RSVD10    0x60
#define OV_RSVD11    0x61
#define OV_LCC1      0x62
#define OV_LCC2      0x63
#define OV_LCC3      0x64
#define OV_LCC4      0x65
#define OV_LCC5      0x66
#define OV_MANU      0x67
#define OV_MANV      0x68
#define OV_GFIX      0x69
#define OV_GGAIN     0x6A
#define OV_DBLV      0x6B
#define OV_AWBCTR3     0x6C
#define OV_AWBCTR2     0x6D
#define OV_AWBCTR1     0x6E
#define OV_AWBCTR0     0x6F
#define OV_SCALING_XSC   0x70
#define OV_SCALING_YSC   0x71
#define OV_SCALING_DCWCTR 0x72
#define OV_SCALING_PCLK_DIV 0x73
#define OV_REG74     0x74
#define OV_REG75     0x75
#define OV_REG76     0x76
#define OV_REG77     0x77
#define OV_RSVD12    0x78
#define OV_RSVD13    0x79
#define OV_GAM1      0x7A
#define OV_GAM2      0x7B
#define OV_GAM3      0x7C
#define OV_GAM4      0x7D
#define OV_GAM5      0x7E
#define OV_GAM6      0x7F
#define OV_GAM7      0x80
#define OV_GAM8      0x81
#define OV_GAM9      0x82
#define OV_GAM10     0x83
#define OV_GAM11     0x84
#define OV_GAM12     0x85
#define OV_GAM13     0x86
#define OV_GAM14     0x87
#define OV_GAM15     0x88
#define OV_GAM16     0x89
#define OV_RSVD14    0x8A
#define OV_RSVD15    0x8B
#define OV_RSVD16    0x8C
#define OV_RSVD17    0x8D
#define OV_RSVD18    0x8E
#define OV_RSVD19    0x8F
#define OV_RSVD20    0x90
#define OV_RSVD21    0x91
#define OV_DM_LNL    0x92
#define OV_DM_LNH    0x93
#define OV_LCC6      0x94
#define OV_LCC7      0x95
#define OV_RSVD22    0x96
#define OV_RSVD23    0x97
#define OV_RSVD24    0x98
#define OV_RSVD25    0x99
```

```
#define OV_RSVD26    0x9A
#define OV_RSVD27    0x9B
#define OV_RSVD28    0x9C
#define OV_BD50ST    0x9D
#define OV_BD60ST    0x9E
#define OV_HIST0     0x9F
#define OV_HIST1     0xA0
#define OV_HIST2     0xA1
#define OV_HIST3     0xA2
#define OV_HIST4     0xA3
#define OV_HIST5     0xA4
#define OV_HIST6     0xA5
#define OV_HIST7     0xA6
#define OV_HIST8     0xA7
#define OV_HIST9     0xA8
#define OV_HIST10    0xA9
#define OV_HIST11    0xAA
#define OV_HIST12    0xAB
#define OV_STR_OPT     0xAC
#define OV_STR_R     0xAD
#define OV_STR_G     0xAE
#define OV_STR_B     0xAF
#define OV_RSVD28_1    0xB0
#define OV_RSVD29    0xB1
#define OV_RSVD30    0xB2
#define OV_THL_ST    0xB3
#define OV_RSVD31    0xB4
#define OV_THL_DLT     0xB5
#define OV_RSVD32    0xB6
#define OV_RSVD33    0xB7
#define OV_RSVD34    0xB8
#define OV_RSVD35    0xB9
#define OV_RSVD36    0xBA
#define OV_RSVD37    0xBB
#define OV_RSVD38    0xBC
#define OV_RSVD39    0xBD
#define OV_AD_CHB    0xBE
#define OV_AD_CHR    0xBF
#define OV_AD_CHGb     0xC0
#define OV_AD_CHGr     0xC1
#define OV_RSVD40    0xC2
#define OV_RSVD41    0xC3
#define OV_RSVD42    0xC4
#define OV_RSVD43    0xC5
#define OV_RSVD44    0xC6
#define OV_RSVD45    0xC7
#define OV_RSVD46    0xC8
#define OV_SATCTR    0xC9



#endif /* OV7670_H_ */
```

H.1.1.9 OV7670.c

../Code/The_Columbus/ColumbusTest/src/CustomDevices/OV7670.c

```c
/*
 * OV7670.c
 *
 * Created: 15/02/2013 13:12:12
 *  Author: hslovett
 */


#include <asf.h>
#include "CustomDevices/CustomDevices.h"
#include "stdio.h"
#include "delay.h"
// Camera
// #include "CustomDevices/OV7670.h"
// I2C Mux
// #include "CustomDevices/PCA9542A.h"
// MotorDriver
// /**#include "CustomDevices/MotorDriver.h"*/
// SDCard
// #include "CustomDevices/SD_Card.h"

__attribute__((__interrupt__)) static void VSYNC0_Handler (void)
{
  //print_dbg("\n\rVSYNC0 Detected!");
  eic_clear_interrupt_line(&AVR32_EIC, VSYNC_0_LINE);
  //VSYNC_0_DISABLE_INTERRUPT;
  switch(OV7670_Status.VSYNC0_State)
  {
    case(TAKE_PHOTO):
      FIFO_0_WEN_SET;
      OV7670_Status.VSYNC0_State = TAKING_PHOTO;
      break;

    case(TAKING_PHOTO):
      FIFO_0_WEN_CLR;
      OV7670_Status.VSYNC0_State = TAKEN_PHOTO;
      break;

    case (TAKEN_PHOTO):
      FIFO_0_WEN_CLR;
      break;

    case(IDLE):
    default:
      VSYNC_0_DISABLE_INTERRUPT;
      FIFO_0_WEN_CLR;
      OV7670_Status.VSYNC0_State = IDLE;
      break;
  }
}

__attribute__((__interrupt__)) static void VSYNC1_Handler (void)
{
  //print_dbg("\n\rVSYNC1 Detected!");
  eic_clear_interrupt_line(&AVR32_EIC, VSYNC_1_LINE);
```

```
  //VSYNC_1_DISABLE_INTERRUPT;
    switch(OV7670_Status.VSYNC1_State)
    {
      case(TAKE_PHOTO):
      FIFO_1_WEN_SET;
      OV7670_Status.VSYNC1_State = TAKING_PHOTO;
      //print_dbg("\n\rCase: Take Photo;");
      break;

      case(TAKING_PHOTO):
      FIFO_1_WEN_CLR;
      OV7670_Status.VSYNC1_State = TAKEN_PHOTO;
      //print_dbg("\n\rCase: Taking Photo;");
      break;

      case (TAKEN_PHOTO):
      FIFO_1_WEN_CLR;
      //print_dbg("\n\rCase: Taken Photo;");
      break;

      case(IDLE):
      default:
      VSYNC_1_DISABLE_INTERRUPT;
      FIFO_1_WEN_CLR;
      OV7670_Status.VSYNC1_State = IDLE;
      //print_dbg("\n\rCase: Idle;");
      break;
    }
}
unsigned char Write_Reg(unsigned char Register, unsigned char Data)
{
  /*  I2C Traffic Generated:
   *  S | OV_7670 + W | A | RegID | A | Data | A | P |
   */
  uint8_t Buff[2] = {Register, Data};
  int status = twim_write(&AVR32_TWIM0, &Buff, 2, OV7670_ADDR, false);
  return status;
}
unsigned char Read_Reg(unsigned char Register, unsigned char *Data)
{
  /*  I2C Traffic Generated:
   *  S | OV_ADDR + W | A | RegID | A | P |
   *  S | OV_ADDR + R | A | Data  |~A | P |
   */
  unsigned char Buff[2] = {Register, 0};
  int status = twim_write(&AVR32_TWIM0, &Buff, 1, OV7670_ADDR, false);
  if(status != STATUS_OK)
    return status;

  status = twim_read(&AVR32_TWIM0, &Buff, 1, OV7670_ADDR, false);
  *Data = Buff[0];

  return status;

}
void OV7670_Init()
{
```

```
  //Check Cameras Exist
  PCA9542A_Chan_Sel(I2C_CHANNEL_0);
  if (twim_probe(&AVR32_TWIM0, OV7670_ADDR) == STATUS_OK)
    OV7670_Status.Camera_0_Found = true;
  else
    OV7670_Status.Camera_0_Found = false;

  PCA9542A_Chan_Sel(I2C_CHANNEL_1);
  if (twim_probe(&AVR32_TWIM0, OV7670_ADDR) == STATUS_OK)
    OV7670_Status.Camera_1_Found = true;
  else
    OV7670_Status.Camera_1_Found = false;


  //Iniialise Cameras
  if(OV7670_Status.Camera_0_Found)
  {
    PCA9542A_Chan_Sel(I2C_CHANNEL_0);
    //Reset Camera
    if(STATUS_OK != Write_Reg(OV_COM7, 0x80))
    {
      print_dbg("\n\rCamera Reset Fail");
      OV7670_Status.Camera_0_Error = true;
      OV7670_Status.Status = ERR_DEVICE;
      //return FAIL;
    }
    delay_ms(10); //wait for Camera to reset
    for (int i = 0; i < SETTINGS_LENGTH; i++)
    {
      if(STATUS_OK != Write_Reg(default_settings[i][0], default_settings[i
    ][1]))
      {
        print_dbg("\n\rCamera Initialise Fail");
        //return FAIL;
        OV7670_Status.Camera_0_Error = true;
        OV7670_Status.Status = ERR_DEVICE;
        break;
      }
      delay_ms(1);
    }
  }
  else
  {
    OV7670_Status.Status |= CAM0_NotFOUND;
  }
  if(OV7670_Status.Camera_1_Found)
  {
    PCA9542A_Chan_Sel(I2C_CHANNEL_1);

    //Reset Camera
    if(STATUS_OK != Write_Reg(OV_COM7, 0x80))
    {
      print_dbg("\n\rCamera Reset Fail");
      OV7670_Status.Camera_1_Error = true;
      OV7670_Status.Status = ERR_DEVICE;
      //return FAIL;
    }
    delay_ms(10); //wait for Camera to reset
```

```
    for (int i = 0; i < SETTINGS_LENGTH; i++)
    {
      if(STATUS_OK != Write_Reg(default_settings[i][0], default_settings[i
    ][1]))
      {
        print_dbg("\n\rCamera Initialise Fail");
        //return FAIL;
        OV7670_Status.Camera_1_Error = true;
        OV7670_Status.Status = ERR_DEVICE;
        break;
      }
      delay_ms(1);
    }
  }
  else
  {
    OV7670_Status.Status |= CAM1_NotFOUND;
  }
  PCA9542A_Chan_Sel(NO_SELECT);

  //Initialise VSYNC Interrupts
  eic_options_t eic_options;
  eic_options.eic_mode = EIC_MODE_EDGE_TRIGGERED;
  eic_options.eic_edge = EIC_EDGE_FALLING_EDGE;
  eic_options.eic_async = EIC_SYNCH_MODE;
  eic_options.eic_line = VSYNC_1_LINE;
  //eic_options.eic_line = VSYNC_0_LINE;

  Disable_global_interrupt();
  gpio_enable_module_pin(VSYNC_1_PIN, VSYNC_1_FUNCTION);
  gpio_enable_module_pin(VSYNC_0_PIN, VSYNC_0_FUNCTION);

  gpio_enable_pin_pull_up(VSYNC_1_PIN); //Enable pull up as it is a low level
    interrupt
  gpio_enable_pin_pull_up(VSYNC_0_PIN);
  //Initialise EIC
  eic_init(&AVR32_EIC, &eic_options, 1);
  eic_options.eic_line = VSYNC_0_LINE;
  eic_init(&AVR32_EIC, &eic_options, 1);

  INTC_register_interrupt(&VSYNC1_Handler, AVR32_EIC_IRQ_1, AVR32_INTC_INT0);
  INTC_register_interrupt(&VSYNC0_Handler, AVR32_EIC_IRQ_4, AVR32_INTC_INT0);
  //Enable interrupt on VSYNC1
  eic_enable_line(&AVR32_EIC, VSYNC_1_LINE);
  eic_enable_line(&AVR32_EIC, (VSYNC_0_LINE));
  VSYNC_1_ENABLE_INTERRUPT;
  VSYNC_0_ENABLE_INTERRUPT;

  FIFO_Init();
  Enable_global_interrupt();

}
void FIFO_Init()
{
  //Disable both outputs
  FIFO_0_nOE_SET
  FIFO_1_nOE_SET
```

```
  //Reset Buffer 0
  FIFO_0_WRST_CLR;
  FIFO_0_RCLK_CLR;
  FIFO_0_nRRST_SET;
  FIFO_0_WEN_CLR;
  delay_us(10);
  FIFO_0_RCLK_SET;
  delay_us(10);
  FIFO_0_RCLK_CLR;
  FIFO_0_nRRST_CLR;
  delay_us(10);
  FIFO_0_RCLK_SET;
  delay_us(10);
  FIFO_0_RCLK_CLR;
  FIFO_0_nRRST_SET;
  delay_us(10);
  FIFO_0_WRST_SET;

  //Reset Buffer 1
  FIFO_1_WRST_CLR;
  FIFO_1_RCLK_CLR;
  FIFO_1_nRRST_SET;
  FIFO_1_WEN_CLR;
  delay_us(10);
  FIFO_1_RCLK_SET;
  delay_us(10);
  FIFO_0_RCLK_CLR;
  FIFO_1_nRRST_CLR;
  delay_us(10);
  FIFO_1_RCLK_SET;
  delay_us(10);
  FIFO_1_RCLK_CLR;
  FIFO_1_nRRST_SET;
  delay_us(10);
  FIFO_1_WRST_SET;
}

void FIFO_Reset(uint8_t CameraID)
{
  FIFO_0_nOE_SET;
  FIFO_1_nOE_SET;
  if(CameraID & CAMERA_LEFT)
  {
    FIFO_0_WRST_CLR;
    FIFO_0_nRRST_CLR;
    FIFO_0_RCLK_SET;
    delay_us(10);
    FIFO_0_RCLK_CLR;
    FIFO_0_nRRST_SET;
    FIFO_0_WRST_SET;
  }
  if(CameraID & CAMERA_RIGHT)
  {
    FIFO_1_WRST_CLR;
    FIFO_1_nRRST_CLR;
    FIFO_1_RCLK_SET;
    delay_us(10);
    FIFO_1_RCLK_CLR;
```

```
    FIFO_1_nRRST_SET;
    FIFO_1_WRST_SET;
  }

}

int TakePhoto(uint8_t Cameras)
{

  //Only want to take pictures on cameras found
  if(((OV7670_Status.VSYNC0_State != IDLE) || !OV7670_Status.Camera_0_Found)
    && ((OV7670_Status.VSYNC1_State != IDLE) || !OV7670_Status.Camera_1_Found)
    )
    return CAMERAS_BUSY; //wait for cameras to be idle if they are found

  if(Cameras & CAMERA_LEFT)
    OV7670_Status.VSYNC0_State = TAKE_PHOTO;

  if(Cameras & CAMERA_RIGHT)
    OV7670_Status.VSYNC1_State = TAKE_PHOTO;
  eic_clear_interrupt_line(&AVR32_EIC, VSYNC_1_LINE);
  eic_clear_interrupt_line(&AVR32_EIC, VSYNC_0_LINE);
  VSYNC_0_ENABLE_INTERRUPT;
  VSYNC_1_ENABLE_INTERRUPT;

  return TAKING_PHOTO;

}

bool Photos_Ready(void)
{
  int status = 0;
  if(OV7670_Status.Camera_0_Found == true) //If camera is there
  {
    if(OV7670_Status.Camera_0_Error == false)//and has no errors
    {
      if(OV7670_Status.VSYNC0_State == TAKEN_PHOTO)
      {
        status |= 1; //camera0 has taken photo
      }
    }
    else
      status |= 1;
  }
  else
    status |= 1;

  if(OV7670_Status.Camera_1_Found == true) //If camera is there
  {
    if(OV7670_Status.Camera_1_Error == false)//and has no errors
    {
      if(OV7670_Status.VSYNC1_State == TAKEN_PHOTO)
      {
        status |= 1; //camera0 has taken photo
      }
    }
    else
    status |= 1;
```

```
  }
  else
    status |= 1;

  if(status)
    return true;
  else
    return false;
}


#define ImageNumberFile "ImageNum.bin"
bool Store_Both_Images()
{
  if(Photos_Ready() == false)
    return false;
  char buff[16];
  //Ideally wanted this done on flash for speed, but difficulties arose with
    using the flash memory.
  nav_filelist_reset();
  if(!nav_filelist_findname((FS_STRING)ImageNumberFile, false))//If a global
    counter file exists
  {
    nav_file_create((FS_STRING)ImageNumberFile); //Make it
    nav_setcwd((FS_STRING)ImageNumberFile, false, false);
    OV7670_Status.ImageCount= 0;
  }
  else
  {
    nav_setcwd((FS_STRING)ImageNumberFile, false, false); //Read the global
    counter file
    file_open(FOPEN_MODE_R);
    file_read_buf(buff, 16); //Read the data.
    OV7670_Status.ImageCount = atoi(buff); //Get the count
    OV7670_Status.ImageCount++; //Increment it
    file_close();
    nav_filelist_reset();
  }
  nav_setcwd((FS_STRING)ImageNumberFile, false, false); //Update the counter
    in the file
  file_open(FOPEN_MODE_W);
  sprintf(buff, "%d", OV7670_Status.ImageCount);
  file_write_buf(buff, 16);
  file_close();
  nav_filelist_reset();

  print_dbg("\n\rImage Number = ");//Print the image number to debug
  print_dbg_ulong(OV7670_Status.ImageCount);


  Store_Image_1();
  FIFO_Reset(CAMERA_RIGHT);

  Store_Image_0();
  FIFO_Reset(CAMERA_LEFT);

  OV7670_Status.VSYNC0_State = IDLE;//Reset the states to IDLE
  OV7670_Status.VSYNC1_State = IDLE;
```

```
  return true;
}

void Store_Image_1()
{
  int i, j;
  //uint8_t buffer[WIDTH * 2];
  char Filename_buff[15];
  uint8_t *Buffer_ram;
  Buffer_ram = mspace_malloc(sdram_msp, HEIGHT * WIDTH * 2);
  i = 0;
  //Use global counter
  sprintf(&Filename_buff, Image1Name, OV7670_Status.ImageCount);
  nav_filelist_reset();
  if(nav_filelist_findname((FS_STRING)Filename_buff, false))//if the file
    already exists
  {
    nav_setcwd((FS_STRING)Filename_buff, false, false);
    nav_file_del(false);//Delete it
  }
  //Image1
  //reset read pointer
  FIFO_1_nRRST_CLR;

  FIFO_1_RCLK_SET;
  delay_us(10);
  FIFO_1_RCLK_CLR;
  FIFO_1_nRRST_SET;

  //enable output
  FIFO_1_nOE_CLR;
  //  uint8_t buffer[WIDTH * 2];

  for(j = 0; j < HEIGHT * WIDTH * 2; j+= 2)
  {
    FIFO_1_RCLK_SET;
    delay_us(10);
    Buffer_ram[j+1] = ((AVR32_GPIO.port[1].pvr) & 0xFF);//CAMERA_INPUT;
    delay_us(10);
    FIFO_1_RCLK_CLR;
    delay_us(10);
    FIFO_1_RCLK_SET;
    delay_us(10);
    Buffer_ram[j] = ((AVR32_GPIO.port[1].pvr) & 0xFF);//CAMERA_INPUT;
    delay_us(10);
    FIFO_1_RCLK_CLR;
    delay_us(10);
  }

  FIFO_1_nOE_SET;//disable output
/*  file_close();*/
  SaveBitmap(Buffer_ram, WIDTH, HEIGHT, Filename_buff);
  mspace_free(sdram_msp, Buffer_ram);
}

void Store_Image_0()
{
  int i, j;
```

```
  //uint8_t buffer[WIDTH * 2];
  char Filename_buff[15];
  uint16_t *Buffer_ram;
  Buffer_ram = mspace_malloc(sdram_msp, HEIGHT * WIDTH );
  i = 0;
  sprintf(&Filename_buff, Image0Name, OV7670_Status.ImageCount);
  nav_filelist_reset();
  if(nav_filelist_findname((FS_STRING)Filename_buff, false))//if the file already exists
  {
    nav_setcwd((FS_STRING)Filename_buff, false, false);
    nav_file_del(false);//Delete it
  }
  //Image0
  //reset read pointer
  FIFO_0_nRRST_CLR;

  FIFO_0_RCLK_SET;
  delay_us(10);
  FIFO_0_RCLK_CLR;
  FIFO_0_nRRST_SET;

  //enable output
  FIFO_0_nOE_CLR;
  //  uint8_t buffer[WIDTH * 2];

  for(j = 0; j < HEIGHT * WIDTH; j++)
  {
    FIFO_0_RCLK_SET;
    delay_us(10);
    Buffer_ram[j] = ((AVR32_GPIO.port[1].pvr) & 0xFF);//CAMERA_INPUT;
    delay_us(10);
    FIFO_0_RCLK_CLR;
    delay_us(10);
    FIFO_0_RCLK_SET;
    delay_us(10);
    Buffer_ram[j] |= (((AVR32_GPIO.port[1].pvr) & 0xFF) << 8);//CAMERA_INPUT;
    delay_us(10);
    FIFO_0_RCLK_CLR;
    delay_us(10);
  }

  FIFO_0_nOE_SET;//disable output
/*  file_close();*/
  SaveBitmap(Buffer_ram, WIDTH, HEIGHT, Filename_buff);
  mspace_free(sdram_msp, Buffer_ram);
}
```

H.1.1.10 OV7670.c

../Code/The_Columbus/ColumbusTest/src/CustomDevices/OV7670_Setup.c

```
/*
 * OV7670_Setup.c
```

```
 *
 * Created: 15/02/2013 13:14:09
 *  Author: hslovett
 */

#include "CustomDevices/CustomDevices.h"

const char default_settings[SETTINGS_LENGTH][2]=
{
{OV_TSLB, 0x04},
{OV_COM15, 0xd0},//RGB565 / RGB555
{OV_COM7, 0x14},
{OV_HREF, 0x80},
{OV_HSTART, 0x16},
{OV_HSTOP, 0x04},
{OV_VSTRT, 0x02},
{OV_VSTOP, 0x7b},//0x7a,
{OV_VREF, 0x06},//0x0a,
{OV_COM3, 0x00}, //MSB and LSB swapped
{OV_COM14, 0x00},//
{OV_SCALING_XSC, 0x00},
{OV_SCALING_YSC, 0x00},
{OV_SCALING_DCWCTR, 0x11},
{OV_SCALING_PCLK_DIV, 0x00},//
{0xa2, 0x02},
{OV_CLKRC, 0x01},
{OV_GAM1, 0x20},
{OV_GAM2, 0x1c},
{OV_GAM3, 0x28},
{OV_GAM4, 0x3c},
{OV_GAM5, 0x55},
{OV_GAM6, 0x68},
{OV_GAM7, 0x76},
{OV_GAM8, 0x80},
{OV_GAM9, 0x88},
{OV_GAM10, 0x8f},
{OV_GAM11, 0x96},
{OV_GAM12, 0xa3},
{OV_GAM13, 0xaf},
{OV_GAM14, 0xc4},
{OV_GAM15, 0xd7},
{OV_GAM16, 0xe8},
{OV_COM8, 0xe0},
{OV_GAIN, 0x00},//AGC
{OV_AECH, 0x00},
{OV_COM4, 0x00},
{OV_COM9, 0x20},//0x38, limit the max gain
{OV_HIST6, 0x05},
{OV_HIST12, 0x07},
{OV_AEW, 0x75},
{OV_AEB, 0x63},
{OV_VPT, 0xA5},
{OV_HIST0, 0x78},
{OV_HIST1, 0x68},
{OV_HIST2, 0x03},//0x0b,
{OV_HIST7, 0xdf},//0xd8,
{OV_HIST8, 0xdf},//0xd8,
{OV_HIST9, 0xf0},
```

```
{OV_HIST10, 0x90},
{OV_HIST11, 0x94},
{OV_COM8, 0xe5},
{OV_COM5, 0x61},
{OV_COM6, 0x4b},
{0x16, 0x02},
{OV_MVFP, 0x27},//0x37,
{0x21, 0x02},
{0x22, 0x91},
{0x29, 0x07},
{0x33, 0x0b},
{0x35, 0x0b},
{0x37, 0x1d},
{0x38, 0x71},
{OV_OFON, 0x2a},//
{OV_COM12, 0x78},
{0x4d, 0x40},
{0x4e, 0x20},
{OV_GFIX, 0x0c},//////////////////////
{OV_DBLV, 0x60},//PLL
{OV_REG74, 0x19},
{0x8d, 0x4f},
{0x8e, 0x00},
{0x8f, 0x00},
{0x90, 0x00},
{0x91, 0x00},
{OV_DM_LNL, 0x00},//0x19,//0x66
{0x96, 0x00},
{0x9a, 0x80},
{0xb0, 0x84},
{0xb1, 0x0c},
{0xb2, 0x0e},
{OV_THL_ST, 0x82},
{0xb8, 0x0a},
{OV_AWBC1, 0x14},
{OV_AWBC2, 0xf0},
{OV_AWBC3, 0x34},
{OV_AWBC4, 0x58},
{OV_AWBC5, 0x28},
{OV_AWBC6, 0x3a},
{0x59, 0x88},
{0x5a, 0x88},
{0x5b, 0x44},
{0x5c, 0x67},
{0x5d, 0x49},
{0x5e, 0x0e},
{OV_LCC3, 0x04},
{OV_LCC4, 0x20},
{OV_LCC5, 0x05},
{OV_LCC6, 0x04},
{OV_LCC7, 0x08},
{OV_AWBCTR3, 0x0a},
{OV_AWBCTR2, 0x55},
{OV_AWBCTR1, 0x11},
{OV_AWBCTR0, 0x9f},//0x9e for advance AWB
{OV_GGAIN, 0x40},
{OV_BLUE, 0x40},
{OV_RED, 0x40},
```

```
{OV_COM8, 0xe7},
{OV_COM10, 0x02},//VSYNC negative
{OV_MTX1, 0x80},
{OV_MTX2, 0x80},
{OV_MTX3, 0x00},
{OV_MTX4, 0x22},
{OV_MTX5, 0x5e},
{OV_MTX6, 0x80},
{OV_MTXS, 0x9e},
{OV_COM16, 0x08},
{OV_EDGE, 0x00},
{OV_REG75, 0x05},
{OV_REG76, 0xe1},
{OV_DNSTH, 0x00},
{OV_REG77, 0x01},
{OV_COM13, 0xc2}, //0xc0,
{OV_REG4B, 0x09},
{OV_SATCTR, 0x60},
{OV_COM16, 0x38},
{OV_CONTRAS, 0x40},
{0x34, 0x11},
{OV_COM11, 0x02},//0x00,//0x02,
{OV_HIST5, 0x89},//0x88,
{0x96, 0x00},
{0x97, 0x30},
{0x98, 0x20},
{0x99, 0x30},
{0x9a, 0x84},
{0x9b, 0x29},
{0x9c, 0x03},
{OV_BD50ST, 0x4c},
{OV_BD60ST, 0x3f},
{0x78, 0x04},
{0x79, 0x01},//Some weird thing with reserved registers.
{0xc8, 0xf0},
{0x79, 0x0f},
{0xc8, 0x00},
{0x79, 0x10},
{0xc8, 0x7e},
{0x79, 0x0a},
{0xc8, 0x80},
{0x79, 0x0b},
{0xc8, 0x01},
{0x79, 0x0c},
{0xc8, 0x0f},
{0x79, 0x0d},
{0xc8, 0x20},
{0x79, 0x09},
{0xc8, 0x80},
{0x79, 0x02},
{0xc8, 0xc0},
{0x79, 0x03},
{0xc8, 0x40},
{0x79, 0x05},
{0xc8, 0x30},
{0x79, 0x26},
{OV_COM2, 0x03},
{OV_BRIGHT, 0x00},
```

```
{OV_CONTRAS, 0x40},
{OV_COM11, 0x42},//0x82,//0xc0,//0xc2,  //night mode

};
```

H.1.1.11 PCA9542A.h

../Code/The_Columbus/ColumbusTest/src/CustomDevices/PCA9542A.h

```
/*
 * PCA9542A.h
 *
 * Created: 15/02/2013 12:21:46
 *  Author: hslovett
 */


#ifndef PCA9542A_H_
#define PCA9542A_H_

#define A0  0
#define A1  0
#define A2  1
#define PCA9542A_ADDR (0x70 | (A2 << 2) | (A1 << 1) | A0)

#define NO_SELECT   0x00
//#define ERROR     0x01
#define I2C_CHANNEL_0 0x04
#define I2C_CHANNEL_1 0x05

//Status Codes
#define SUCCESS   0
#define DEVICE_NOT_FOUND  2

typedef struct {
  uint8_t Status;
  uint8_t ChannelSelected;
  } PCA9542A_t;

PCA9542A_t PCA9542A;
int PCA9542A_Init();
//void PCA9542A_Channel_Select(uint8_t Channel);
void PCA9542A_Chan_Sel(unsigned char Channel);
#endif /* PCA9542A_H_ */
```

H.1.1.12 PCA9542A.c

../Code/The_Columbus/ColumbusTest/src/CustomDevices/PCA9542A.c

```
/*
 * PCA9542A.c
```

```
 *
 * Created: 15/02/2013 12:21:36
 *  Author: hslovett
 */

#include <asf.h>
#include "CustomDevices/CustomDevices.h"
//Camera
/*#include "CustomDevices/OV7670.h"*/
//I2C Mux
/*#include "CustomDevices/PCA9542A.h"*/
//MotorDriver
/*#include "CustomDevices/MotorDriver.h"*/
//SDCard
/*#include "CustomDevices/SD_Card.h"*/

int PCA9542A_Init()
{
  int status = twim_probe(&AVR32_TWIM0, PCA9542A_ADDR);
  if (status != STATUS_OK)
  {
    PCA9542A.Status = DEVICE_NOT_FOUND;
    return DEVICE_NOT_FOUND;
  }
  char buff[2] = {NO_SELECT, 0};
  status = twim_write(&AVR32_TWIM0, &buff, 1, PCA9542A_ADDR, false);
  PCA9542A.Status = STATUS_OK;
  PCA9542A.ChannelSelected = NO_SELECT;
  return status;
}


void PCA9542A_Chan_Sel(unsigned char Channel)
{
  int status = 0;
  char buff[2] = {Channel, 0};
  status = twim_write(&AVR32_TWIM0, &buff, 1, PCA9542A_ADDR, false);
  if(status == STATUS_OK)
  {
    PCA9542A.ChannelSelected = Channel;
  }
  else
  {
    PCA9542A.Status = ERR_PROTOCOL;
  }
}
```

H.1.1.13 SD_Card.h

../Code/The_Columbus/ColumbusTest/src/CustomDevices/SD_Card.h

```
/*
 * SD_Card.h
 *
```

```
 * Created: 10/02/2013 17:11:51
 *  Author: hslovett
 */


#ifndef SD_CARD_H_
#define SD_CARD_H_
#include "ImageProcessor.h"
#define SIGNAL_FILE "signal.bin"
#define  TWOD_SIGNAL_FILE "signal2d.bin"

typedef struct {
  uint8_t Status;
  uint32_t Memory_size;
  } SD_Status_t;
SD_Status_t SD_Status;

void local_pdca_init(void);
void sd_mmc_resources_init(void);
static void pdca_int_handler(void);
void wait();
void Log_Write_ulong(unsigned long n);
void Log_Write(char *buff, int length);
void SaveBuff( int * WorkingBuffer , int size);
int Read2DSignal( int * WorkingBuffer );
int ReadSignal( int * WorkingBuffer );
void SaveBitmap(uint16_t *Image, int width, int height, char *FileName);
//void ReadBitmap(char *Filename);
void ReadBitmap(char *Filename, Image_t *image);
void SaveBuff_CSV(char *Filename, int *WorkingBuffer, int size);
void SaveCBuff_CSV(char *Filename, dsp16_complex_t *ComplexBuffer, int size);
void Read_CSV(char *Filename, int *WorkingBuffer, int size);
#endif /* SD_CARD_H_ */
```

H.1.1.14 SD_Card.c

../Code/The_Columbus/ColumbusTest/src/CustomDevices/SD_Card.c

```
/*
 * SD_Card.c
 *
 * Created: 10/02/2013 17:11:58
 *  Author: hslovett
 */
//Camera
/*#include "CustomDevices/OV7670.h"*/
//I2C Mux
/*#include "CustomDevices/PCA9542A.h"*/
//MotorDriver
/*#include "CustomDevices/MotorDriver.h"*/
//SDCard
/*#include "CustomDevices/SD_Card.h"*/
#include "CustomDevices/CustomDevices.h"
#include "conf_sd_mmc_spi.h"
```

```
#include <asf.h>
#include "stdlib.h"
#include "stdio.h"
// Dummy char table
const char dummy_data[] =
#include "dummy.h"
;


// PDCA Channel pointer
volatile avr32_pdca_channel_t* pdca_channelrx ;
volatile avr32_pdca_channel_t* pdca_channeltx ;
// Used to indicate the end of PDCA transfer
volatile bool end_of_transfer;
// Local RAM buffer for the example to store data received from the SD/MMC
    card
volatile char ram_buffer[1000];


void wait()
{
  volatile int i;
  for(i = 0 ; i < 5000; i++);
}
/* interrupt handler to notify if the Data reception from flash is
 * over, in this case lunch the Memory(ram_buffer) to USART transfer and
 * disable interrupt*/

static void pdca_int_handler(void)
{
  // Disable all interrupts.
  Disable_global_interrupt();

  // Disable interrupt channel.
  pdca_disable_interrupt_transfer_complete(AVR32_PDCA_CHANNEL_SPI_RX);

  sd_mmc_spi_read_close_PDCA();//unselects the SD/MMC memory.
  wait();
  // Disable unnecessary channel
  pdca_disable(AVR32_PDCA_CHANNEL_SPI_TX);
  pdca_disable(AVR32_PDCA_CHANNEL_SPI_RX);

  // Enable all interrupts.
  Enable_global_interrupt();

  end_of_transfer = true;
}

/*! \brief Initializes SD/MMC resources: GPIO, SPI and SD/MMC.
 */
void sd_mmc_resources_init(void)
{
  // GPIO pins used for SD/MMC interface
  static const gpio_map_t SD_MMC_SPI_GPIO_MAP =
  {
    {SD_MMC_SPI_SCK_PIN,  SD_MMC_SPI_SCK_FUNCTION },  // SPI Clock.
    {SD_MMC_SPI_MISO_PIN, SD_MMC_SPI_MISO_FUNCTION},  // MISO.
    {SD_MMC_SPI_MOSI_PIN, SD_MMC_SPI_MOSI_FUNCTION},  // MOSI.
```

```
    {SD_MMC_SPI_NPCS_PIN, SD_MMC_SPI_NPCS_FUNCTION}   // Chip Select NPCS.
  };

  // SPI options.
  spi_options_t spiOptions =
  {
    .reg          = SD_MMC_SPI_NPCS,
    .baudrate     = SD_MMC_SPI_MASTER_SPEED,  // Defined in conf_sd_mmc_spi.h.
    .bits         = SD_MMC_SPI_BITS,          // Defined in conf_sd_mmc_spi.h.
    .spck_delay   = 0,
    .trans_delay  = 0,
    .stay_act     = 1,
    .spi_mode     = 0,
    .modfdis      = 1
  };

  // Assign I/Os to SPI.
  gpio_enable_module(SD_MMC_SPI_GPIO_MAP,
                     sizeof(SD_MMC_SPI_GPIO_MAP) / sizeof(SD_MMC_SPI_GPIO_MAP[0]));

  // Initialize as master.
  spi_initMaster(SD_MMC_SPI, &spiOptions);

  // Set SPI selection mode: variable_ps, pcs_decode, delay.
  spi_selectionMode(SD_MMC_SPI, 0, 0, 0);

  // Enable SPI module.
  spi_enable(SD_MMC_SPI);

  // Initialize SD/MMC driver with SPI clock (PBA).
  sd_mmc_spi_init(spiOptions, PBA_HZ);
}

/*! \brief Initialize PDCA (Peripheral DMA Controller A) resources for the SPI transfer and start a dummy transfer
 */
void local_pdca_init(void)
{
  // this PDCA channel is used for data reception from the SPI
  pdca_channel_options_t pdca_options_SPI_RX ={ // pdca channel options

    .addr = ram_buffer,
    // memory address. We take here the address of the string dummy_data. This string is located in the file dummy.h

    .size = 512,                              // transfer counter: here the size of the string
    .r_addr = NULL,                           // next memory address after 1st transfer complete
    .r_size = 0,                              // next transfer counter not used here
    .pid = AVR32_PDCA_CHANNEL_USED_RX,        // select peripheral ID - data are on reception from SPI1 RX line
    .transfer_size = PDCA_TRANSFER_SIZE_BYTE  // select size of the transfer: 8,16,32 bits
  };
```

```
  // this channel is used to activate the clock of the SPI by sending a dummy variables
  pdca_channel_options_t pdca_options_SPI_TX ={ // pdca channel options

    .addr = (void *)&dummy_data,              // memory address.
                                              // We take here the address of the string dummy_data.
                                              // This string is located in the file dummy.h
    .size = 512,                              // transfer counter: here the size of the string
    .r_addr = NULL,                           // next memory address after 1st transfer complete
    .r_size = 0,                              // next transfer counter not used here
    .pid = AVR32_PDCA_CHANNEL_USED_TX,        // select peripheral ID - data are on reception from SPI1 RX line
    .transfer_size = PDCA_TRANSFER_SIZE_BYTE  // select size of the transfer: 8,16,32 bits
  };

  // Init PDCA transmission channel
  pdca_init_channel(AVR32_PDCA_CHANNEL_SPI_TX, &pdca_options_SPI_TX);

  // Init PDCA Reception channel
  pdca_init_channel(AVR32_PDCA_CHANNEL_SPI_RX, &pdca_options_SPI_RX);

  //! \brief Enable pdca transfer interrupt when completed
  INTC_register_interrupt(&pdca_int_handler, AVR32_PDCA_IRQ_0, AVR32_INTC_INT1);  // pdca_channel_spi1_RX = 0

}

#define BUFFER_FILENAME   "Buffer.bin"
#define BUFFERCSV_FILENAME    "Buffer.csv"
void SaveBuff( int * WorkingBuffer , int size)
{
  //If the file exists, delete it
  if(nav_filelist_findname((FS_STRING)BUFFER_FILENAME, false))
  {
    nav_setcwd((FS_STRING)BUFFER_FILENAME, false, false);
    nav_file_del(false);
  }
  nav_file_create((FS_STRING)BUFFER_FILENAME);
  nav_setcwd((FS_STRING)BUFFER_FILENAME, false, true);
  file_open(FOPEN_MODE_APPEND);
  file_write_buf(WorkingBuffer, size * sizeof(WorkingBuffer));
  file_close();
}
void SaveBuff_CSV(char *Filename, int *WorkingBuffer, int size)
{
  int i, j;
  char Buff[16];
  //If the file exists, delete it
  nav_filelist_reset();
  if(nav_filelist_findname((FS_STRING)Filename, false))
  {
    nav_setcwd((FS_STRING)Filename, false, false);
```

```
      nav_file_del(false);
    }
    nav_file_create((FS_STRING)Filename);
    nav_setcwd((FS_STRING)Filename, false, true);
    file_open(FOPEN_MODE_W);
    for(i = 0; i < size; i++)
    {
      sprintf(Buff, "%d,", WorkingBuffer[i]);
      //itoa(WorkingBuffer[i], Buff, 10);
      j = 0;
      while(Buff[j++] != 0);//count the size of data to be written
      atoi(Buff);
      file_write_buf(Buff, j-1);
      //file_write_buf(",", 1);
    }

    file_close();
  }
  void SaveCBuff_CSV(char *Filename, dsp16_complex_t *ComplexBuffer, int size)
  {
    int i, j;
    char Buff[16];
    //If the file exists, delete it
    nav_filelist_reset();
    if(nav_filelist_findname((FS_STRING)Filename, false))
    {
      nav_setcwd((FS_STRING)Filename, false, false);
      nav_file_del(false);
    }
    nav_file_create((FS_STRING)Filename);
    nav_setcwd((FS_STRING)Filename, false, true);
    file_open(FOPEN_MODE_W);
    for(i = 0; i < size; i++)
    {
      if(ComplexBuffer[i].imag >= 0)
        sprintf(Buff, "%d+%dj,", ComplexBuffer[i].real, ComplexBuffer[i].imag);
      else
        sprintf(Buff, "%d%dj,", ComplexBuffer[i].real, ComplexBuffer[i].imag);
      //itoa(WorkingBuffer[i], Buff, 10);
      j = 0;
      while(Buff[j++] != 0);//count the size of data to be written
      atoi(Buff);
      file_write_buf(Buff, j-1);
      //file_write_buf(",", 1);
    }

    file_close();
  }
  void Read_CSV(char *Filename, int *WorkingBuffer, int size)
  {
    char Buff[32];
    int i, j;
    char c;
    nav_filelist_reset();
    if(WorkingBuffer == NULL){
      print_dbg("\n\rRead_CSV: Buffer not initialised");
      return;
    }
```

```
  //Check file Exists
  if(nav_filelist_findname((FS_STRING)Filename, false) == false){
    print_dbg("\n\rRead_CSV : File doesn't exist;");
    return;
  }
  nav_setcwd((FS_STRING)Filename, false, true);
  file_open(FOPEN_MODE_R);
  for(i = 0; i < size; i++)
  {
    if(file_eof())
      break;
    c = 0;
    //j = 0;
    for(j = 0; j < 32; j++)
      Buff[j] = 0; //clear the buffer
    j = 0;
    while(c != ',')
    {
      if(file_eof())
        break;
      c = file_getc();
      if(c == ',')
        break;
      Buff[j++] = c; //load string into buffer
    }
    WorkingBuffer[i] = atoi(Buff); //Convert to int and put into buffer
  }
  file_close();
  return;
}
void Log_Write(char *buff, int length)
{
  nav_setcwd((FS_STRING)LOG_FILE, true, false);
  file_open(FOPEN_MODE_APPEND);
  if(length == -1)
    length = sizeof(buff);
  file_write_buf(buff, length);
  file_close();
}
void Log_Write_ulong(unsigned long n)
{
  char tmp[11];
  int i = sizeof(tmp) - 1;

  // Convert the given number to an ASCII decimal representation.
  tmp[i] = '\0';
  do
  {
  tmp[--i] = '0' + n % 10;
  n /= 10;
  } while (n);

  // Transmit the resulting string with the given USART.
  Log_Write(tmp + i, -1);
}

int ReadSignal( int * WorkingBuffer )
{
```

```
  bool status_b;
  int Status, temp;
  char c = 0;
  if(Columbus_Status.SD_Card->Status != STATUS_OK)
    return ERR_IO_ERROR;
  nav_filelist_reset();
  nav_setcwd((FS_STRING)SIGNAL_FILE, false, false);
  status_b = file_open(FOPEN_MODE_R);
  if(status_b == false)
  {
    print_dbg("File Open Error");
    return ERR_IO_ERROR;
  }


  //Status = file_read_buf(WorkingBuffer, 16);
  for(Status = 0; Status < FFT_SIZE; Status++)
  {
//    print_dbg("\n\r Read from file: ");
    c = 0;
    temp = 0;
    temp |= file_getc() << 24;
    temp |= file_getc() << 16;
    temp |= file_getc() << 8;
    temp |= file_getc();

//    print_dbg_char(c);

    WorkingBuffer[Status] = temp;
//    print_dbg("  Working Buff = ");
//    print_dbg_char(WorkingBuffer[Status]);
  }
  file_close();
  return STATUS_OK;
}

int Read2DSignal( int * WorkingBuffer )
{
  bool status_b;
  int Status, temp;
  char c = 0;
  if(Columbus_Status.SD_Card->Status != STATUS_OK)
  return ERR_IO_ERROR;
  nav_filelist_reset();
  nav_setcwd((FS_STRING)TWOD_SIGNAL_FILE, false, false);
  status_b = file_open(FOPEN_MODE_R);
  if(status_b == false)
  {
    print_dbg("File Open Error");
    return ERR_IO_ERROR;
  }


  //Status = file_read_buf(WorkingBuffer, 16);
  for(Status = 0; Status < FFT_SIZE * FFT_SIZE; Status++)
  {
    //    print_dbg("\n\r Read from file: ");
    c = 0;
```

```
    temp = 0;
    temp |= file_getc() << 24;
    temp |= file_getc() << 16;
    temp |= file_getc() << 8;
    temp |= file_getc();

    //    print_dbg_char(c);

    WorkingBuffer[Status] = temp;
    //    print_dbg("  Working Buff = ");
    //    print_dbg_char(WorkingBuffer[Status]);
  }
  file_close();
  return STATUS_OK;
}

void SaveBitmap(uint16_t *Image, int width, int height, char *FileName)
{
  int i, j, k;
  uint8_t *Buffer;

  nav_filelist_reset();
  if(nav_filelist_findname((FS_STRING)FileName, false))
  {
    nav_setcwd((FS_STRING)FileName, true, false);
    nav_file_del(false);
  }
  nav_file_create((FS_STRING)FileName);
  file_open(FOPEN_MODE_W);
  //write a modified bitmap header
  //Calculate which is the biggest:
  i = width * 2;
  if(height > i)
    i = height;
  if(DIBHEADERSIZE > i)
    i = DIBHEADERSIZE;

  Buffer = malloc(i);

  for(i = 0; i < BMPHEADERSIZE; i++)//copy all the header
  {
    Buffer[i] = BMPHeader[i];
  }
  //edit the size field
  j = width * height * 2 + BMPHEADERSIZE + DIBHEADERSIZE;
  for(i = 0; i < 4; i ++)
  {
    Buffer[i + 2] = (uint8_t)(j >> 8*i);
  }

  file_write_buf(Buffer, BMPHEADERSIZE);

  //DIB Header
  for(i = 0; i < DIBHEADERSIZE; i ++)
  {
    Buffer[i] = DIBHead[i];
  }
  Buffer[4] = (uint8_t)(width & 0xFF);
```

```
  Buffer[5] = (uint8_t)((width >> 8) & 0xFF);
  Buffer[6] = (uint8_t)((width >> 16) & 0xFF);
  Buffer[7] = (uint8_t)((width >> 24) & 0xFF);

  Buffer[8] = (uint8_t)(height & 0xFF);
  Buffer[9] = (uint8_t)((height >> 8) & 0xFF);
  Buffer[10] = (uint8_t)((height >> 16) & 0xFF);
  Buffer[11] = (uint8_t)((height >> 24) & 0xFF);

  file_write_buf(Buffer, DIBHEADERSIZE);

  for(i = 0; i < height ; i++ )
  {
    for(j = 0; j < width ; j++)
    {
      //Copy the data across.

      /*Buffer[j] = Image[i*width + j];*/
      Buffer[(2 * j) + 1] = (uint8_t)(Image[i*width + j]);
      Buffer[(2 * j)] = (uint8_t)(Image[i*width + j] >> 8);
    }
    if(file_write_buf(Buffer, width * 2) != (width * 2))
    {
      print_dbg("\n\rFile write error.");
    }

//    j = width % 4;
//    if(j != 0)
//    {//Padding is needed to make things 4 byte aligned
//      file_write_buf(Buffer, j);
//    }
  }



  free(Buffer);
  file_close();
}

#define BMP_HEADER_FILESIZE_OFFSET    2
#define BMP_HEADER_OFFSETTOARRAY_OFFSET 10
#define DIB_V5_WIDTH_OFFSET       4
#define DIB_V5_HEIGHT_OFFSET      8
#define DIB_V5_BITCOUNT_OFFSET     14
#define DIB_V5_IMAGESIZE_OFFSET     20

int ReadBigEndian(uint8_t *Buffer, int Offset, uint size)
{
  int retVal, i;
  retVal = 0; //initialise value
  for(i = 0; i < size; i++)
  {
    retVal |= Buffer[Offset + i] << (i * 8);
  }
  return (Buffer[Offset]) | (Buffer[Offset + 1] << 8) | (Buffer[Offset + 2] <<
     16) | (Buffer[Offset + 3] << 24);
}
void ReadBitmap(char *Filename, Image_t *image)
```

```
{
//  Image_t image;
  int i, j, FileSize, OffsetToArray, temp, BitCount, ImageSize;
  uint8_t Buffer[128];
  nav_filelist_reset();
  if(nav_filelist_findname((FS_STRING)Filename, false) == false)//if the file doesn't exist
  {
    print_dbg("\n\rFile ");
    print_dbg(Filename);
    print_dbg("\n\r does not exist;");
    return;
  }
  nav_setcwd((FS_STRING)Filename, false, false);
  file_open(FOPEN_MODE_R);
  //Read Header
  file_read_buf(Buffer, BMPHEADERSIZE);
  //Check for BM to confirm it is a Bitmap
  if((Buffer[0] != 'B') || (Buffer[1] != 'M'))
  {
    print_dbg("\n\rBitmap Parse Fail 'BM';");
    return;
  }
  //Extract file size and offset to pixel array
  FileSize = ReadBigEndian(Buffer, BMP_HEADER_FILESIZE_OFFSET, 4);
  OffsetToArray = ReadBigEndian(Buffer, BMP_HEADER_OFFSETTOARRAY_OFFSET, 4);

  file_read_buf(Buffer, DIBHEADERSIZE);
  temp = ReadBigEndian(Buffer, 0, 4);
  if(temp != 0x7C) //check it is a V5 BMP DIB Header
  {
    print_dbg("\n\rBMP Parse: DIB Header not V5;");
    return;
  }
  image->Width= ReadBigEndian(Buffer, DIB_V5_WIDTH_OFFSET, 4);
  image->Height = ReadBigEndian(Buffer, DIB_V5_HEIGHT_OFFSET, 4);
  BitCount = ReadBigEndian(Buffer, DIB_V5_BITCOUNT_OFFSET, 2);
  ImageSize = ReadBigEndian(Buffer, DIB_V5_IMAGESIZE_OFFSET, 4);
  print_dbg("\n\rBitmap Width = ");
  print_dbg_ulong(image->Width);
  print_dbg("\n\rBitmap Height = ");
  print_dbg_ulong(image->Height);
  print_dbg("\n\rBitmap File Size = ");
  print_dbg_ulong(FileSize);
  print_dbg("\n\rBitmap Offset to Array = ");
  print_dbg_ulong(OffsetToArray);
  print_dbg("\n\rBitmap Image Bitcount = ");
  print_dbg_ulong(BitCount);
  print_dbg("\n\rBitmap Image Size = ");
  print_dbg_ulong(ImageSize);

  file_seek(OffsetToArray, FS_SEEK_SET);
  j = 0;
  image->ImagePtr = mspace_malloc(sdram_msp, image->Height * image->Width);
  for(i = 0; i < ImageSize; i += 2)
  {
    image->ImagePtr[j++] = (file_getc()<<8) | (file_getc());
  }
```

```
  file_close();
  nav_filelist_reset();
  return;
}
```

H.1.1.15 TWI.c

../Code/The_Columbus/ColumbusTest/src/CustomDevices/TWI.c

```
/*
 * TWI.c
 *
 * Created: 27/02/2013 10:51:19
 *  Author: hslovett
 */

#include "CustomDevices/CustomDevices.h"
#include <asf.h>

#define TARGET_ADDRESS      0x0            //! Target's TWI address
#define TARGET_ADDR_LGT     3               //! Internal Address length
#define VIRTUALMEM_ADDR     0x123456        //! Internal Address
#define TWIM_MASTER_SPEED   50000           //! Speed of TWI


void twim_init (void)
{
  int8_t status;
  /**
  * \internal
  * PIN 2 & 3 in Header J24 can be used in EVK1104
  * PIN 1 & 2 in Header J44 can be used in UC3C_EK
  * \endinternal
  */
  const gpio_map_t TWIM_GPIO_MAP = {
  {AVR32_TWIMS0_TWCK_0_0_PIN, AVR32_TWIMS0_TWCK_0_0_FUNCTION},
  {AVR32_TWIMS0_TWD_0_0_PIN, AVR32_TWIMS0_TWD_0_0_FUNCTION}
  };

  // Set TWIM options
  const twi_options_t TWIM_OPTIONS = {
    .pba_hz = FOSC0,
    .speed = TWIM_MASTER_SPEED,
    .chip = TARGET_ADDRESS,
    .smbus = false,
  };
  // TWIM gpio pins configuration
  gpio_enable_module (TWIM_GPIO_MAP,
      sizeof (TWIM_GPIO_MAP) / sizeof (TWIM_GPIO_MAP[0]));

  // Initialize as master.
  status = twim_master_init (&AVR32_TWIM0, &TWIM_OPTIONS);

}
```

H.2 MATLAB Code

H.2.1 Image Matching Algorithms

H.2.1.1 Match.m

../MATLAB/Match.m

```matlab
function [ NCC MatchDiff ] = Match( left, right )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
%close all;
% Global Constants
BoxSize = 50

% Display Images
f = figure;
subplot(2,2,1);
imshow(left);
title('Left Image');
subplot(2,2,2);
imshow(right);
title('Right Image');
% Get Template
figure;
imshow(right);
title('Choose location to Cross Correlate with the Left Image');
rSubCoord = ginput(1);
[rSubCoord(2), rSubCoord(1)];
rSubCoord = round(rSubCoord);
Rx = rSubCoord(1);
Ry = rSubCoord(2);
close;
leftG = rgb2gray(left);
rightG = rgb2gray(right);
% Extract Template
RightTemplate = rightG(Ry-(BoxSize/2):Ry+(BoxSize/2), Rx-(BoxSize/2):Rx+(
    BoxSize/2));

%Cross Correlate
NCC = normxcorr2(RightTemplate, leftG);

%Display
figure(f+1);
%subplot(2,2,3);
surf(NCC);
hold on;
shading interp;

%Find Maximum
[~, l] = max(NCC(:));
%Return Match Coordinates
[~, Lx] = ind2sub(size(NCC),l);
Lx = Lx-BoxSize/2;
```

```
fprintf('Right Template at %d\n', Rx);
fprintf('Left Match at %d\n', Lx);

MatchDiff = abs(Lx - Rx);
end
```

H.2.1.2 MatchAll.m

../MATLAB/MatchAll.m

```
for i = 1:10
    l = sprintf('Range_Test_Images/Duck_L_%d00.bmp', i);
    r = sprintf('Range_Test_Images/Duck_R_%d00.bmp', i);

    left = imread(l);
    right = imread(r);

    Matches(i) = Match(left,right);

end
```