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). The original project brief can be seen in Appendix A. 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 and 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. Microsoft's Xbox Kinect (Microsoft, 2012) uses stereo vision to locate a game player in order to use their movements to control the game. Mrovlje and Vrančić (2008) used stereo vision to be able to locate the distance to a marker.

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 was mounted on the base seen in Figure 1.1 and used two OmniVision OV7670 cameras delivering colour QVGA format images.

The final robot 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

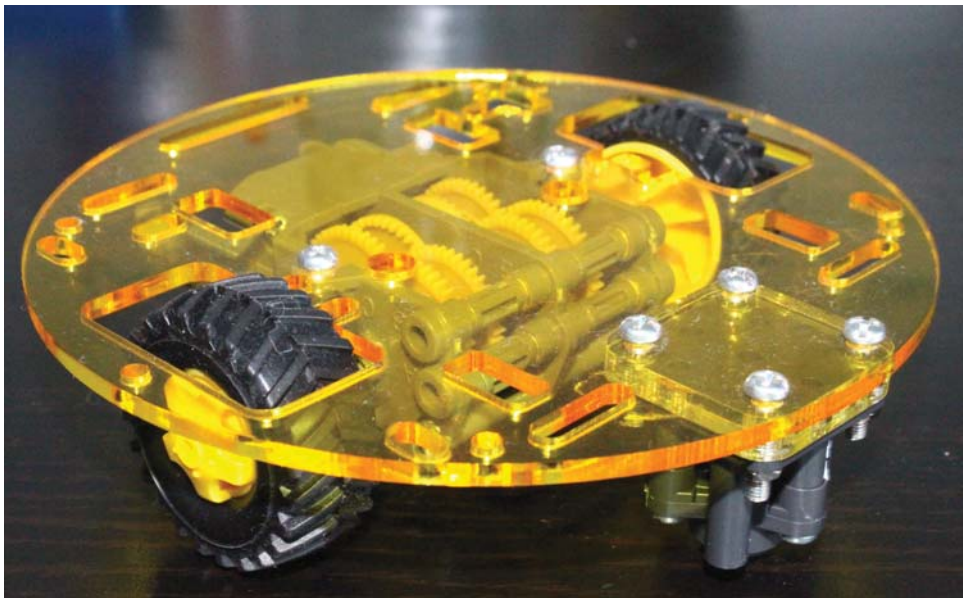


Figure 1.1: The base of the robot

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

This project set ambitious targets. All sections of the design that needed addressing are seen in Table 1.1. The project was developed using a spiral model to achieve the design aspects, starting with the hardware and firmware which were developed concurrently. This was done to obtain basic functionality first before concentrating on more complex aspects. Higher level algorithms were also prototyped during initial hardware development. However, the software was inherently dependent on the hardware. Therefore the hardware aspect was given priority.

A number of risks were present during the project. These were assessed and steps were taken to reduce the chance of them occurring. These can be seen in Table 1.2.

At the beginning of the project, the Gantt chart in Figure B.1 was created. This was then reviewed after the interim report hand-in and another Gantt chart was made, seen in Figure B.2, to show how time was allocated for the second semester. Figure B.3 documents how time was actually spent during the project.

Table 1.1: Hardware and software aspects of the design

Hardware	**Software**
Single Camera	Firmware
Dual Camera	Matching Algorithms
SD Card	Range Finding
Motor System	Mapping
PCB Design	Searching
PCB Test	Testing

Table 1.2: A list of risks and the prevention steps taken to reduce their impact

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 backups of all work on Devtrack Git repository and Dropbox.
Damage to the robot through physical force or electrical connection	Medium	Ensure robot is transported in a box. Be careful not to connect the supply voltage to the $3V3$ rail to avoid damaging components.

Figure B.3 shows that more time was spent on testing than originally anticipated. Also, while developing individual sections, small errors occurred that were time-consuming to correct, causing the time allocated to be insufficient. The motor driving required a large change when porting to the AT32, meaning time to be spent developing this further.

1.2 Contributions

The main part of this project was the hardware design and the embedded firmware development. MATLAB skills had to be learnt for the prototyping of high-level

algorithms for image processing. The skills obtained in MATLAB also included producing graphs and 3D plots as well as general use of the software. Programming for a 32 bit Atmel microcontroller also had to be learnt during the project. This project supplied a practical use for theory learnt in lectures of the Fourier transform and cross correlation.

Chapter 2

Research

The research for this project was split into three sections:

1. Hardware
2. Software, broken down into:
   (a) Firmware
   (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 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 developed by ARM Holdings. ARM devices come in a many varieties: ARM9, ARM7, Strong ARM, and ARM Cortex, amongst others. 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 Harvard RISC architecture and tend to be 32-bit with a high clock speed. ARM microcontrollers have on-chip support for SPI, I^2C , PWM and 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 was 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. Many implementations of firmware for this camera exist across different devices. The camera operation is discussed in Section 3.1.

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). UC3C is the AT32UC3C0512C, XMega is the ATXMega256A3BU and the ATMega is the ATmega644P

Attribute	**Stellaris**	**UC3C**	**XMega**	**ATMega**
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[i]	2.7-5.5
Communication Interfaces	SPI, I^2C, SSI, MAC, CAN, EPI[ii], 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

[i] Clock frequency for voltages 1.6 - 2.7 is 12MHz. [ii] EPI - External Peripheral Interface, equivalent to EBI

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. This 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 Figure C.1. The camera has a small hardware modification, due to a PCB fault. Pin 8 of the buffer was lifted and connected to pin 6 of the header, which was 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.

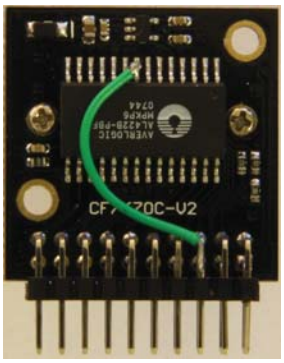


Figure 3.1: Reverse side of the OV7670 camera showing the modification

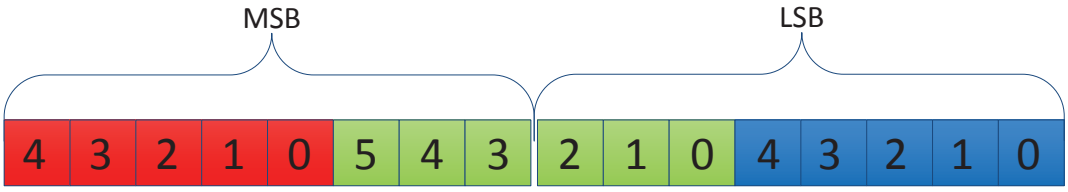


Figure 3.2: RGB565 pixel format

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 (2012) and the two protocols 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 use 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 65,536 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 and easily converted to grey scale.

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

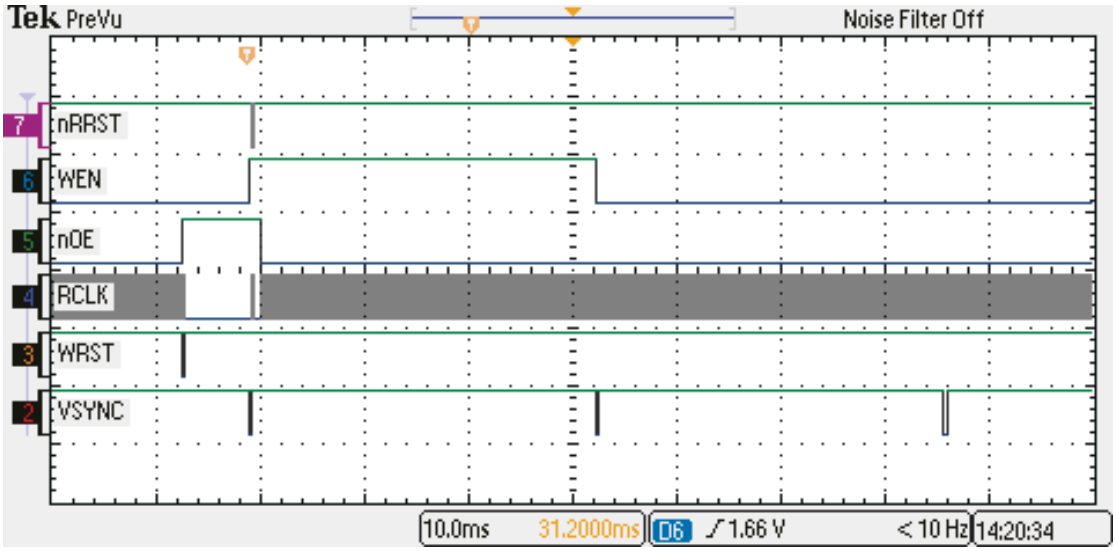


Figure 3.3: Signals generated to control the OV7670 capture and read

The camera must use a high speed clock in order to ensure the pixels obtained are from the same time frame. This makes it difficult for an ATMega (typically clocked at 8-12MHz) to be able to respond to the camera quickly 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 output data is valid. The pixel data is then clocked out on every rising edge of PCLK. To control the buffer, WEN is logically NAND with the HREF signal. When both are high, the WEN to the buffer is active and the data will be clocked into the buffer by the camera.

In order to store a full image into the buffer, 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 FIFO buffer will contain all the valid pixel data.

To obtain the image data from the FIFO 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 8 bits at a time.

After the data has been read, the buffer is reset by asserting the read reset (RRST) and write reset (WRST) signals for at least one clock cycle of RCLK or PCLK respectively. The entire operation can be seen in Figure 3.3 and a hierarchical diagram of this is seen in Figure 3.4.

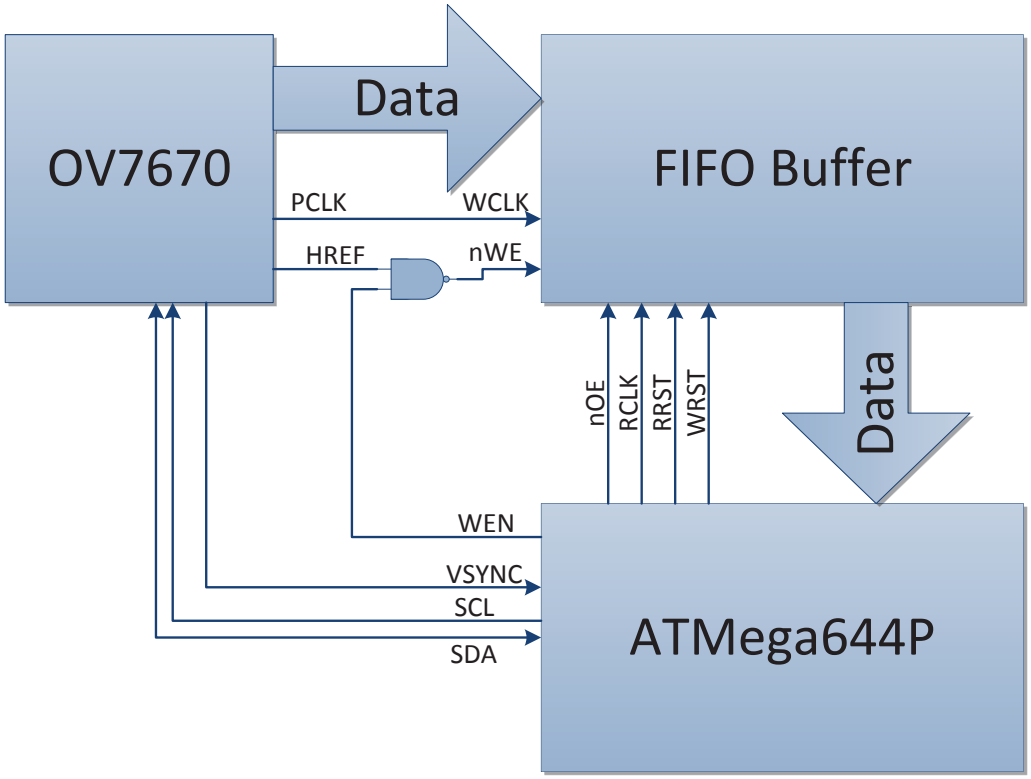


Figure 3.4: Hierarchical diagram of the single camera operation showing signals between the camera, FIFO buffer and the microcontroller

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 was 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 must be added to pull up the two extra buses and the three interrupt lines which aren't used.

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.

The 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 was 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.

3.2 SD Card

An SD card was chosen due to its small size, low cost and a large data storage. The cards work using an SPI bus which can also be used for other devices within

Table 3.1: Comparison of different image formats available (Fulton, 2010)

	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

the system. 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 H shows the make up of a bitmap file that was used.

Storing the image as a bitmap enables the file to be opened on a range of operating systems. This aids debugging and allows the prototyping of image algorithms in a more powerful environment. Figure 3.5 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

Figure 3.5: An example image taken using the OV7670 and stored as a Bitmap on the SD card

reading the bitmaps. This could be avoided by opening and resaving the images as a bitmap in MS Paint.

3.3 The Prototypes

The first prototype made can be seen in Figure 3.6. This obtained two images from both cameras, and stored them to the SD Card. The cameras shared the data bus and an I^2C multiplexer was used. An ATMega168 microcontroller was added to the system as no pins remained for debugging on the *Il Matto* board. The pinout for the ATMega644P 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 camera circuit worked.

A second prototype was made to stream the images to the TFT screen designed for the *Il Matto*. This can be seen in Figure 3.7. This provided the ability to check the cameras worked, and 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.

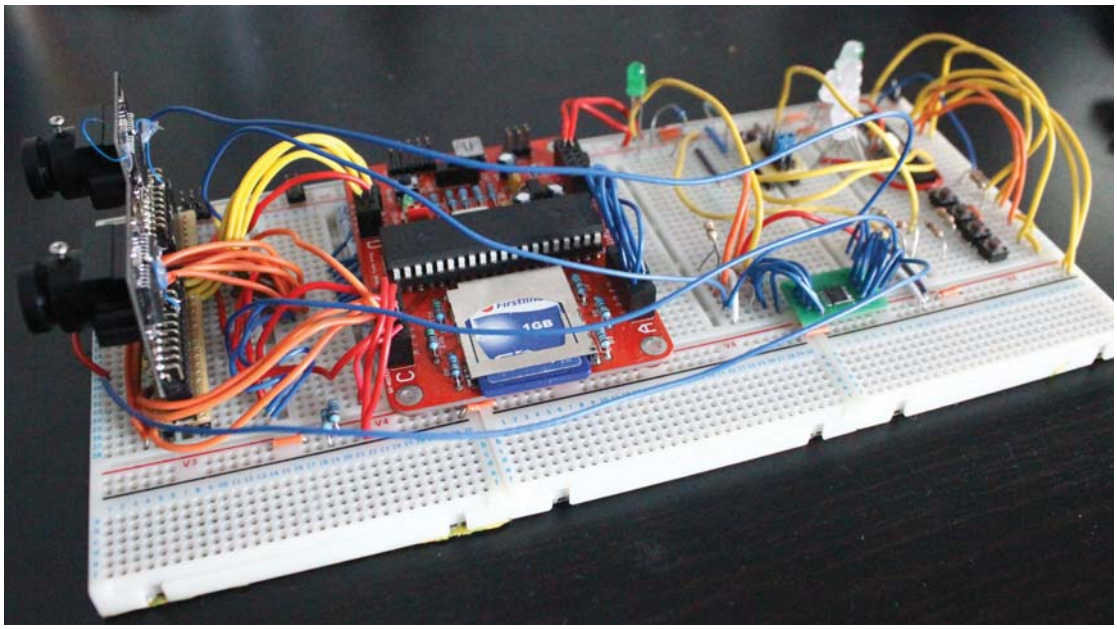


Figure 3.6: Prototype of dual camera operation

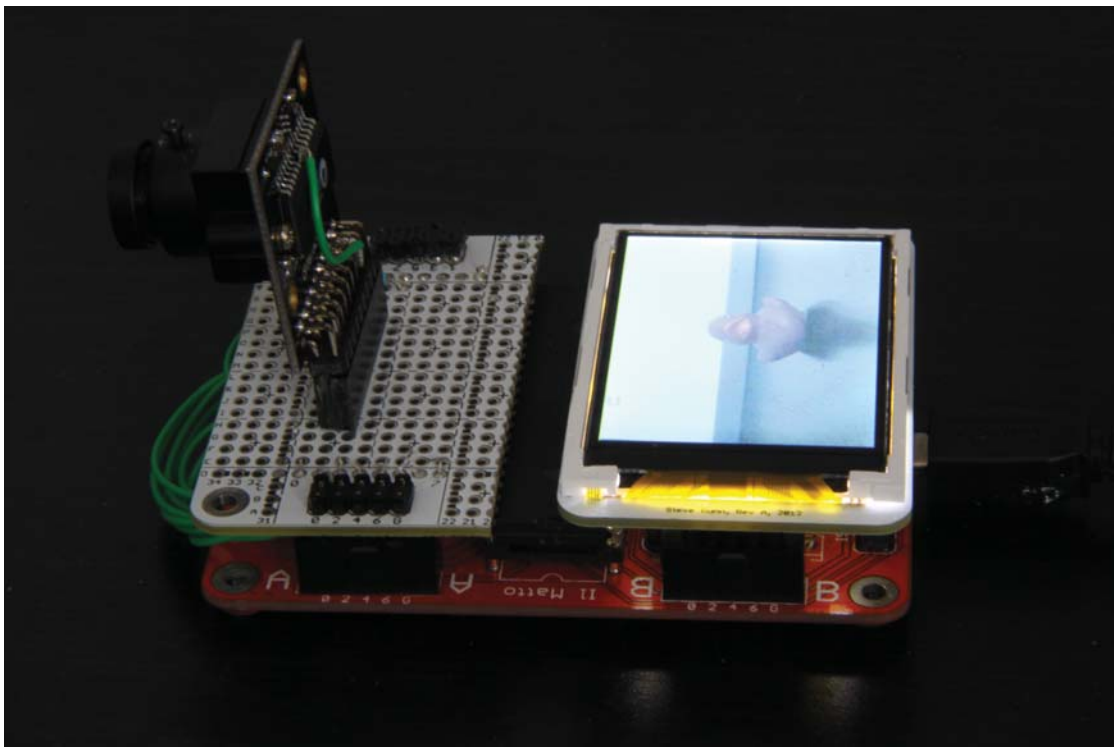


Figure 3.7: Prototype of streaming camera images to TFT screen

Table 3.2: Pin connections of the ATMega644P for dual camera operation.

Pin	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

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 and speed can be calculated given the wheel size.

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 the transistor amplifier used can be seen in Figure 3.8, 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 3.9 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 3.9. A lot of noise exists due to imperfect white tabs. However, the dips are prominent and can be detected with the correct threshold voltage.

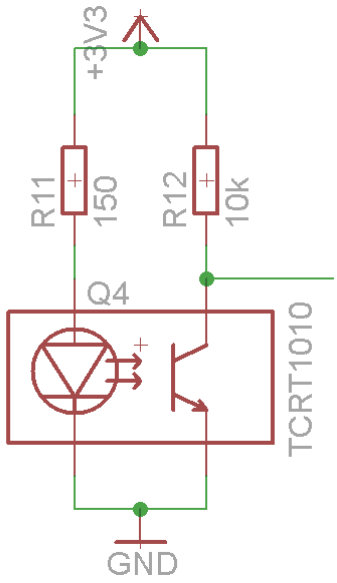


Figure 3.8: Circuit diagram of optosensor amplifier

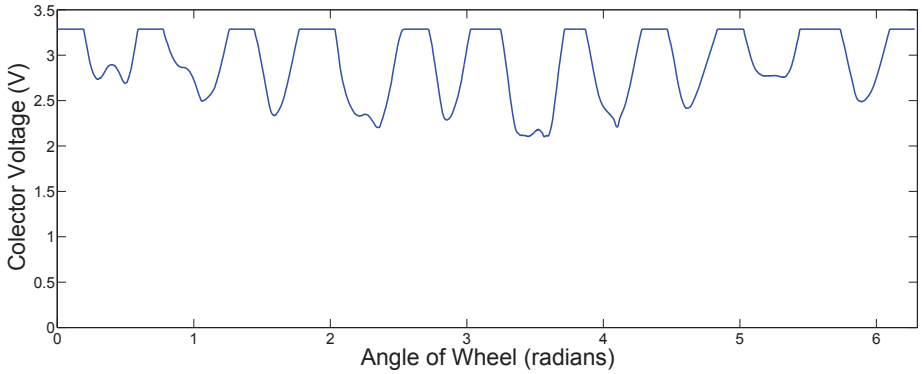


Figure 3.9: Graph of wheel angle against the voltage read by the AVR

3.4.2 Firmware Development

To enable the robot to move a set distance, the number of wheel rotations must be counted. This is achieved by incrementing a count every time a tab is 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 left for further work.

Table 3.3: 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 function 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 3.11(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).

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

Combining Equations (3.1) and (3.2) gives:

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

Figure 3.11 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 representing whether the collector voltage was higher or lower than the threshold voltage. The detection of a tab was then simpler. This method was chosen 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

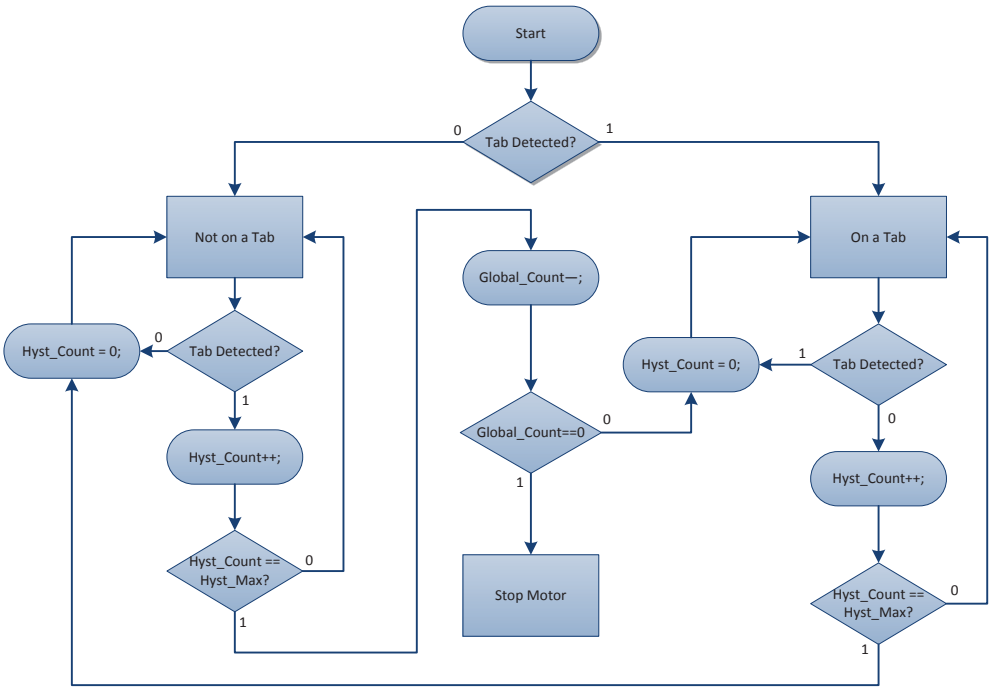
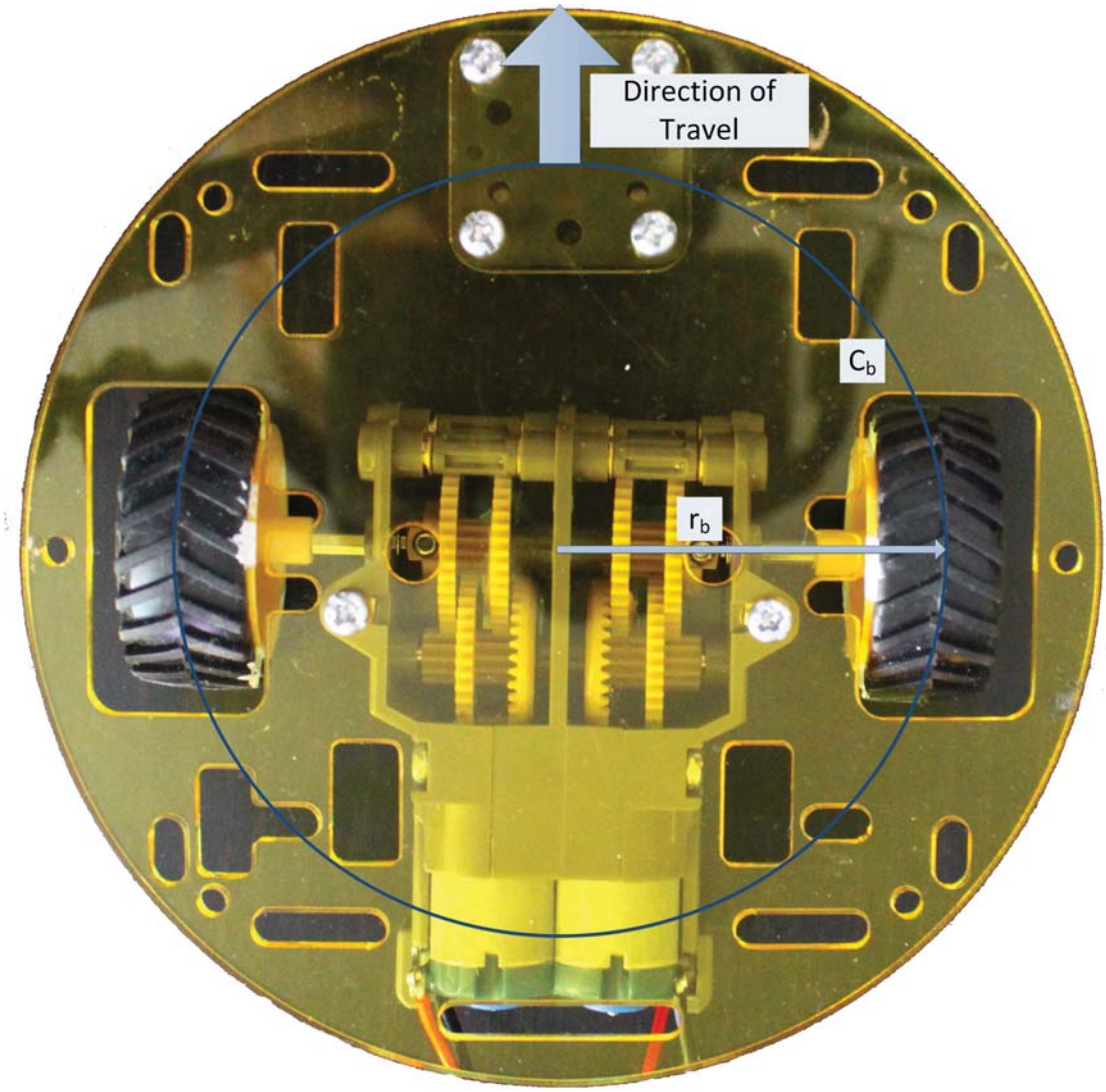


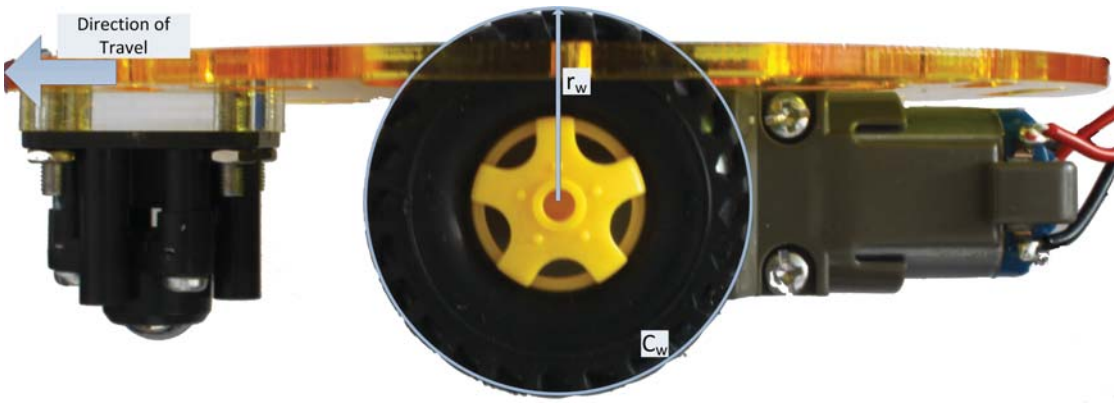
Figure 3.10: A state machine showing the operation of the *Motor_Execute* method

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 3.10. Starting in the '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 and increase the certainty that a tab is, or is not, 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.

In code, the *Motor_Init* method must be called before *Motor_Execute* can be performed. 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.



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



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

Figure 3.11: Dimensions of interest for robot movement

Table 3.4: Results of motor distance test

Distance Specified (mm)	**Number of Counts Calculated**	**Predicted Distance (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

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 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.

However, the robot did not move in a straight line, but with a slight arc. Speed tests were performed 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 caused by this is small enough to neglect.

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

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 that a greater distance resolution could be obtained by decreasing the wheel size or increasing γ, and a greater rotational resolution could be obtained by using the above, 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 counts, namely 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. 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 than to 1cm, this method has proved to be cheap and successful. The robot is able to move a distance with reasonable accuracy, albeit to a fairly low resolution.

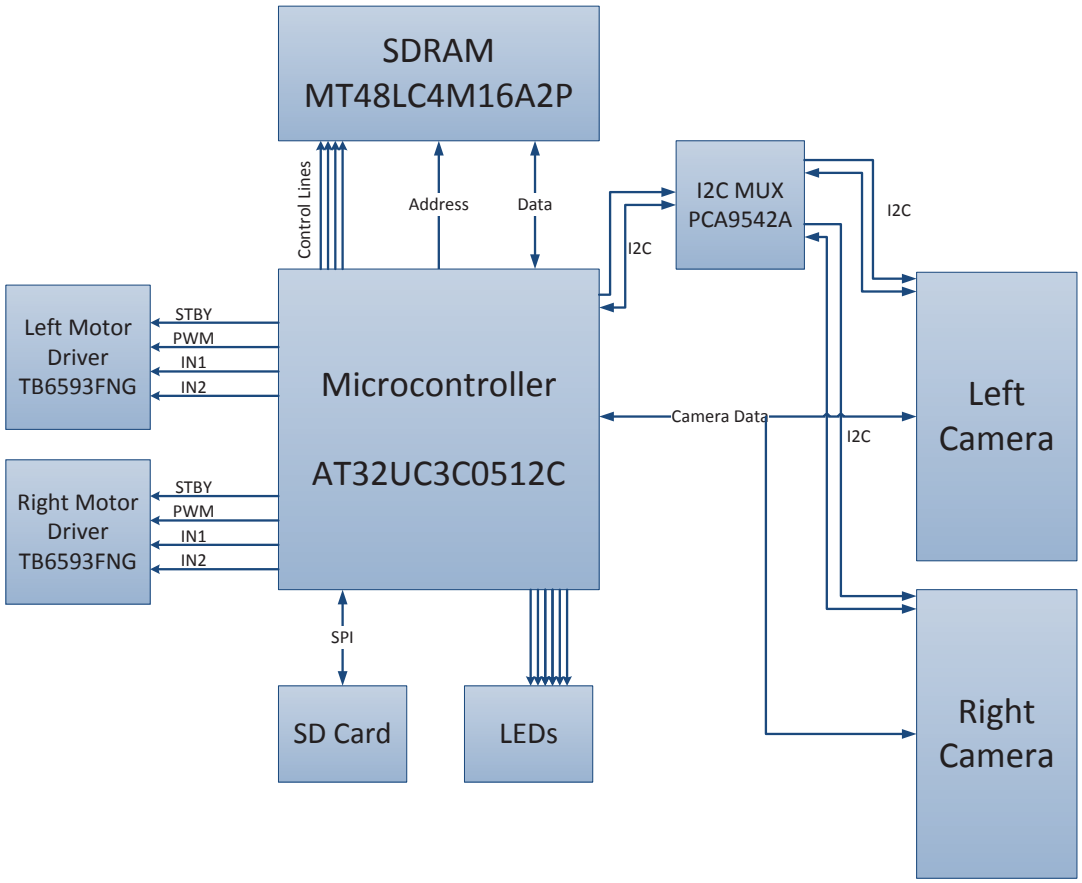


Figure 3.12: A hierarchical diagram of the robot

3.5 PCB Development

3.5.1 Circuit Design

Figure 3.12 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 Appendix C. 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).

3.5.2 PCB Design

The PCB was designed using EAGLE CAD Software. A four layer board was decided upon to reduce the number of tracks, and more easily supply power and

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

Pin	Port A	Port B	Port C	Port 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-ADDR1
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	-

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 that the track lengths matched to each other.

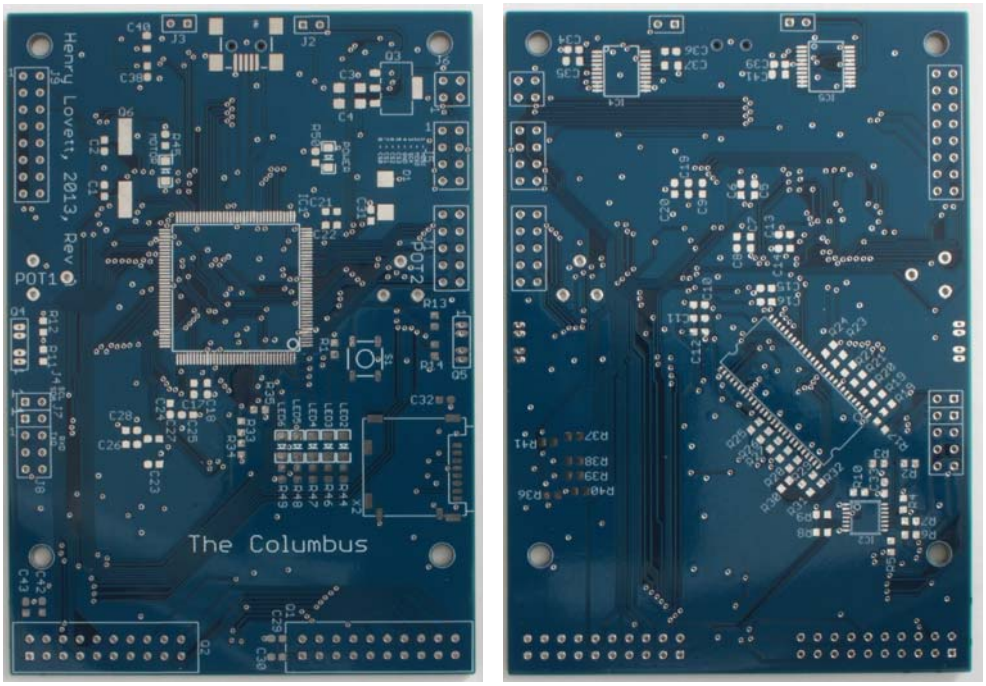
Tracks were routed in order of priority, starting with the UC3C, SDRAM and cameras. All other devices were then routed (I^2C multiplexer, SD card, motor drivers etc). As a precaution, spare pins from the UC3C were routed to headers (J8 and J9), so that additions could be done if a pinout or connection was found to be incorrect. USART, I^2C and SPI connections were routed to headers J7, J4 and J5 respectively so that 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 inch spaced for easy connections and a mini B USB socket was added. USB could be used to power the robot or to program the AVR via the bootloader. Future work could make the robot act as a USB device.

The layout of components was important. The cameras needed to be as far apart as possible and be positioned at the front of the PCB. The motor drivers were situated toward the back of the PCB and headers were added to connect to the motors. The optosensors were positioned as such so they could be mounted directly on to the PCB and be in the correct position to sense the wheels. Mounting holes were also added on to 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 E.

The Eagle CAD Diagram of the PCB can be seen in Appendix D. The PCB was manufactured by PCB Cart (2013). Five PCBs cost £205 to manufacture and ship. A photo of the PCB can be seen in Figure 3.13.

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. The



(a) Top of the PCB (b) Bottom of the PCB

Figure 3.13: PCB with no components mounted

robot was named after Christopher Columbus who explored and navigated parts of the American continents which were unknown at the time.

3.5.3 PCB Testing

A program was written to test all the devices on the PCB. Tests were done to exercise all features and are explained in the subsequent sections. All code can be seen in Appendix F

3.5.3.1 USART 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 F.1 shows the test code for the USART protocol.

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 consists 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.

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 that 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 in the ASF (Atmel Corporation, 2009). The code can be seen in Listing F.4. 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 integer, *'noErrors'*, is used to count errors.

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 acknowledge a probe. A probe is set up to write to the address. If a device exists on the line, it should acknowledge (ACK) its own address (Phillips,

2012). The probe consists of a set-up to write to the address. Figure 3.14 shows the I^2C traffic generated for a probe, with both cases of the device responding with an ACK, or negative acknowledging (NACK). 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 addresses expected are 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 F.5 shows the test code for the I^2C bus and Listing 3.1 shows the result from the full bus scan with channel 0 selected. The existence of both cameras is checked.

Listing 3.1: Result of I^2C bus scan with Channel 0 of the I^2C multiplexer selected

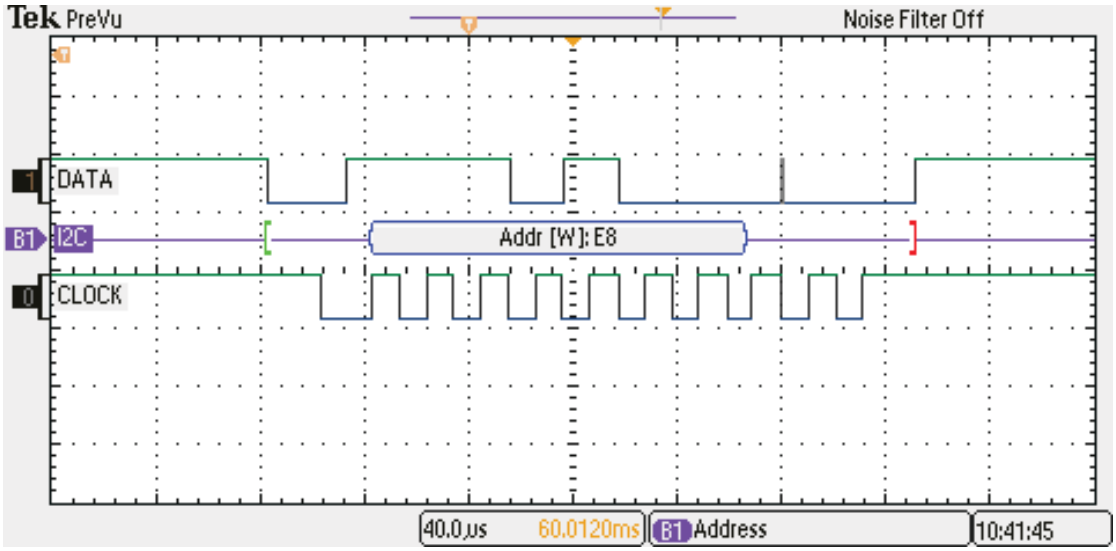
```
Scanning Channel 0
h 0 1 2 3 4 5 6 7 8 9 A B C D E F
0 - - - - - - - - - - - - - - - -
1 - - - - - - - - - - - - - - - -
2 - A - - - - - - - - - - - - - -
3 - - - - - - - - - - - - - - - -
4 - - - - - - - - - - - - - - - -
5 - - - - - - - - - - - - - - - -
6 - - - - - - - - - - - - - - - -
7 - -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

3.5.3.6 Camera Test

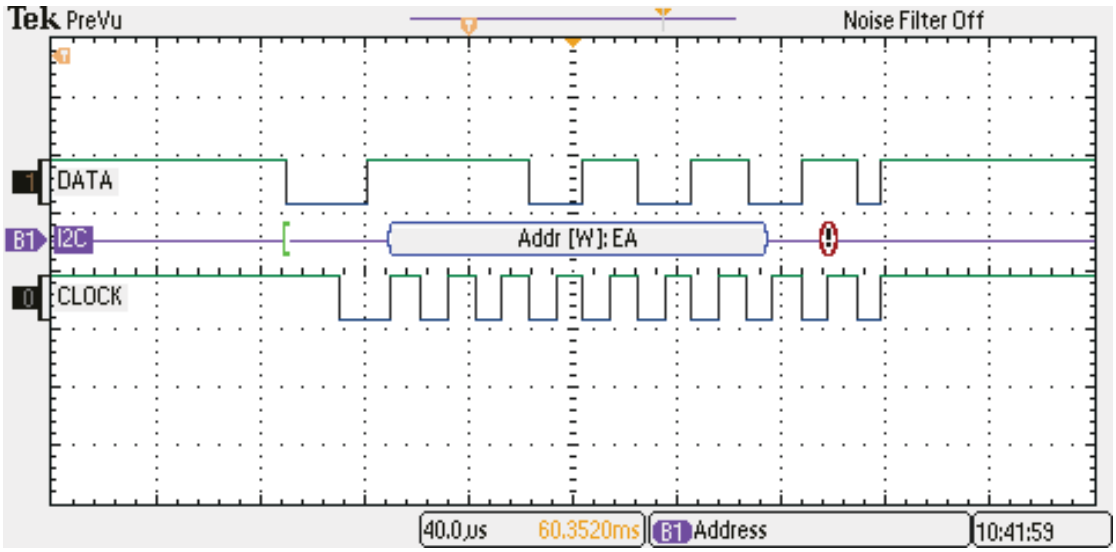
This test consists of initialising both cameras and checking it succeeds. Two photos are then taken and stored to the SD card. Success or failure of the initialisation and capture is sent to the debug terminal. An image from each camera should exist on the SD card. Listing F.6 shows the code to conduct this test.

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 F.7. The robot should move no further than 1cm to reach a white tab and the motors should drive forward. This test is useful as it ensures the motors are connected the correctly and that the threshold is set to an appropriate level.



(a) ACK



(b) NACK

Figure 3.14: The I^2C traffic of a probe. An ACK shows a device exists. *'!'* indicates a NACK was returned, lack of *'!'* means ACK. Green bracket shows a Start and red bracket indicates a Stop.

3.5.4 PCB Faults

During the build and test of the PCB, a number of faults were found. All the faults are documented below of the problem and a solution to avoid these from occurring.

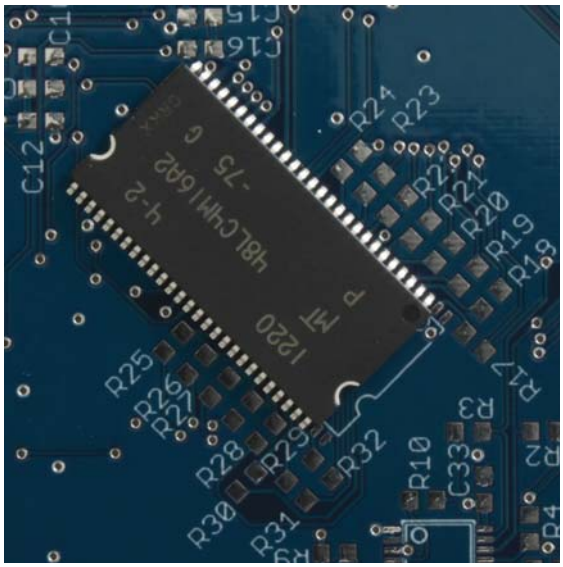


Figure 3.15: SDRAM chip shown against its footprint

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 implications other than the connection being weaker than it should be.

3.5.4.2 SDRAM Footprint

The SDRAM footprint was made exactly to the specification of the pad size and locations with no consideration for soldering. This meant the chip fitted exactly on to the footprint and made soldering difficult. It put the device at risk as more heat had to be used than would usually be necessary. Figure 3.15 shows the SDRAM chip slightly offset against the footprint. Though this made building difficult and resulted in more soldering errors, it had no impact on the operation of the device. To avoid reproducing the problem, future PCB designs should use existing footprints where available, and consider the design from a practical assembly point of view.

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 but 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 vias 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 3.17(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 consistently 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: 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 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. 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 header. The operation was then easily obtained and the VSYNC line triggered correctly.

This issue would have been avoided with more understanding of the device beforehand and by checking the datasheet. The patch can also be seen in Figure 3.17(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 pins for each motor output.The pin assignment was mixed up when created and connected the two outputs together. Figure 3.16 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. No impact to the operation of the drivers has been seen, but the patch may hinder the device's 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:

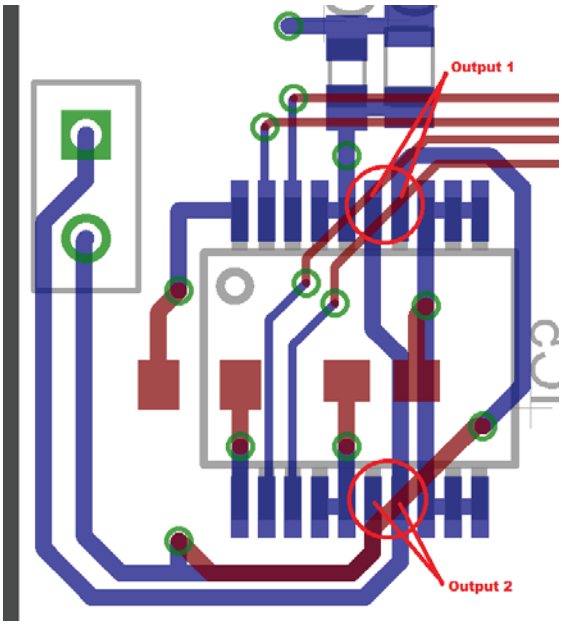


Figure 3.16: Motor driver error - outputs incorrectly connected

- 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 patched. By utilising the soldermask more, errors would be prevented during building.

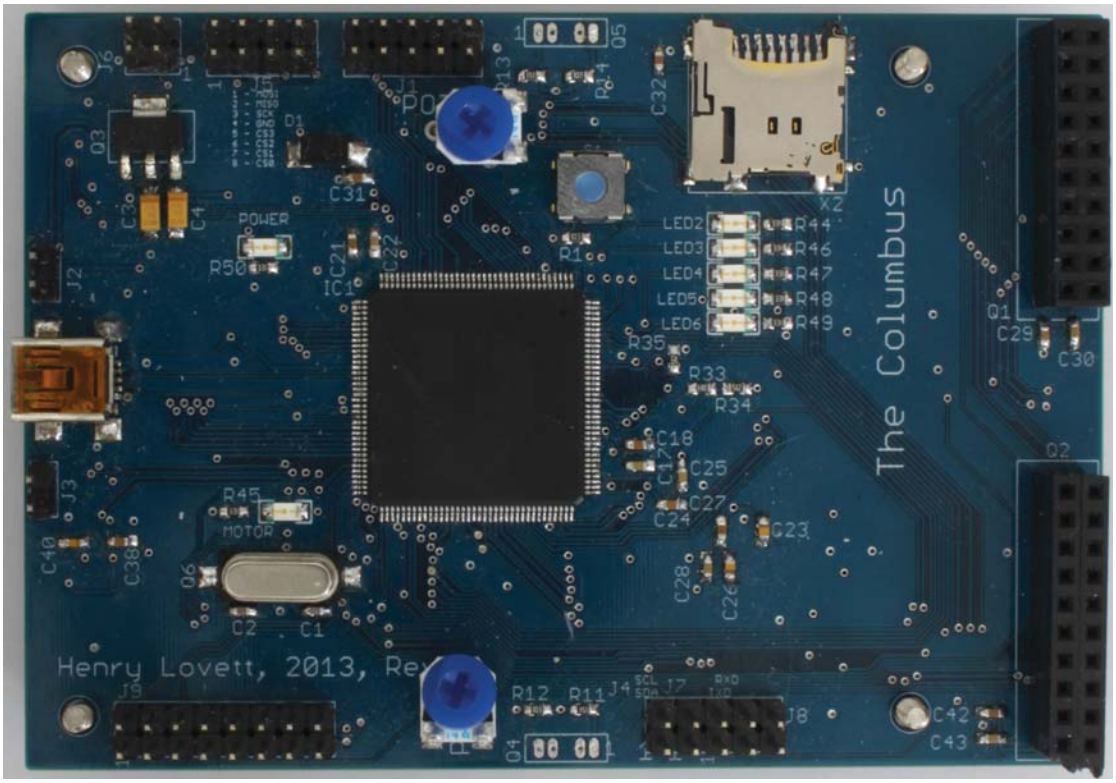
For future PCBs, more care will be taken in circuit design, with prototyping of circuits with the hardware that will be used. This will highlight any pin-specific operations (e.g. the non maskable interrupt) and reduce debugging post-production. The effectiveness of a soldermask is also apparent, so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

The PCB itself was a success. It was a complex PCB with a high risk of potential errors. All devices are functional (with a few small modifications) on the PCB so

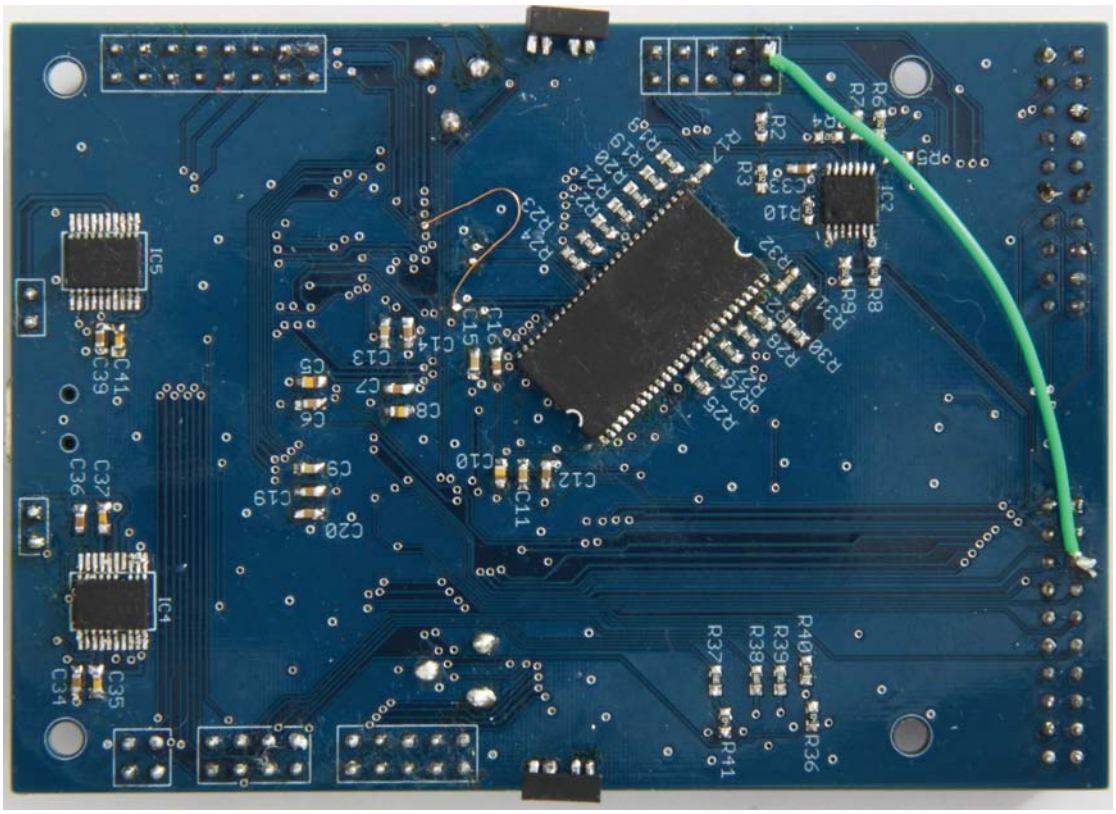
firmware development could continue with all hardware able to be used. Figure 3.17 shows the fully assembled PCB with the modifications needed.

3.6 Conclusions

Overall, the hardware design and firmware was 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, and memory of both SD card (up to 2GB size) and external 4MB SDRAM. This provides a good platform for a manoeuvrable, stereoscopic vision robot to be developed.



(a) Top view of built PCB



(b) Bottom view of built PCB with patches

Figure 3.17: Pictures of the built PCB

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 (SAD) (Hamzah et al., 2010), the sum of squared differences (SSD)(Mrovlje and Vrančić, 2008) and normalised cross correlation (NCC)(Zhao et al., 2006). Each of these methods will be explained and tested in order to compare them. All testing will use images seen in Figure 4.1. Each test uses the same size template ($50px \times 50px$) 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 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 calculation of the square root of a number in order to find the standard deviation. Floating point arithmetic is extensively used, which takes much more time to execute on a microcontroller than pure integer arithmetic.

4.1.4 Comparison

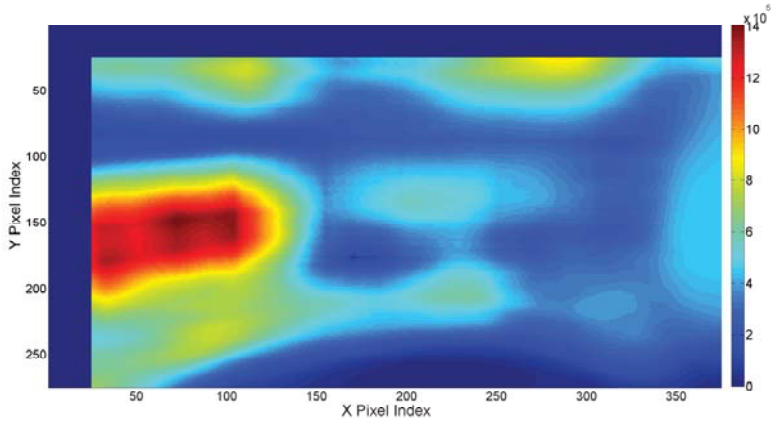
To compare these equations, a $50px \times 50px$ template taken from the right picture was compared with the left image over the entire valid range. The coordinates on the graph give the centre pixel of the calculation.

Each graph shows the correct area being identified as a match, but this also highlights the downfalls of the SAD and SSD methods. 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).

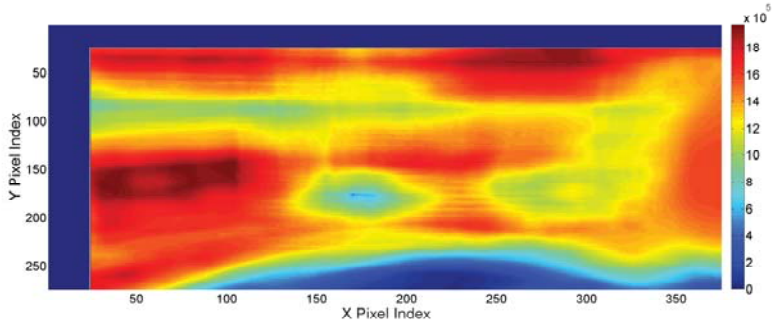
SAD results in Figure 4.2(a) show large areas of matching. A minimum occurs around the location expected$(170, 175)$ of a value of 5.66×10^4. However, the dark area beneath the desk is a false detection. The SAD algorithm detects a greater comparison with a low value of 3370 at $(227, 275)$ which causes a false match.

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.

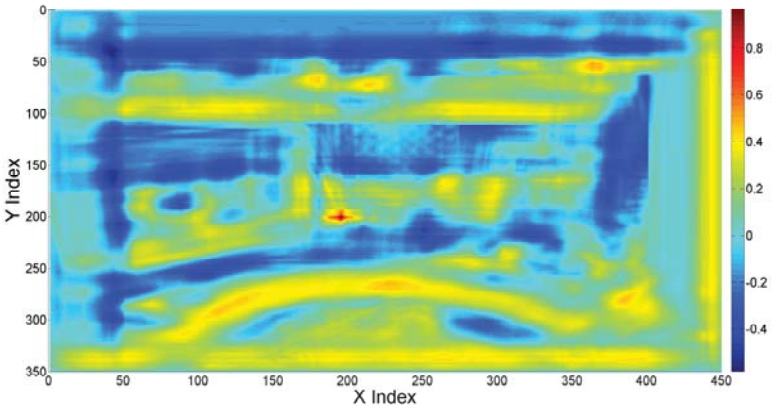
The NCC results are visible in Figure 4.2(c). A match can be seen at coordinate $(195, 201)$ with a peak value of 0.9654. The coordinate is different to the previous results because the cross correlation works over the boundary of the image creating more results. The dimensions of the image are $300px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



(a) S.A.D results - blue shows areas of matching



(b) S.S.D. results - blue shows areas of matching



(c) N.C.C. results - red shows areas of matching

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 is apparent 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 suitable for microcontrollers 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 I contains the derivations for the follow scenarios:

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

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

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 distance to an object.

4.2.3 Field of View

The field of view of the camera is an important variable that must be measured. 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. Equation (4.9), derived from the set-up in Figure 4.3, was then used to calculate the field of view. This was done multiple times for accuracy. 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 $\varphi_0 = 0.6249^c$.

$$\varphi_0 = 2\arctan\left(\frac{L}{2D}\right) \tag{4.9}$$

4.2.4 Testing

MATLAB was used to test the range finding. It is unable to automatically detect an object so the user must select the template when prompted.

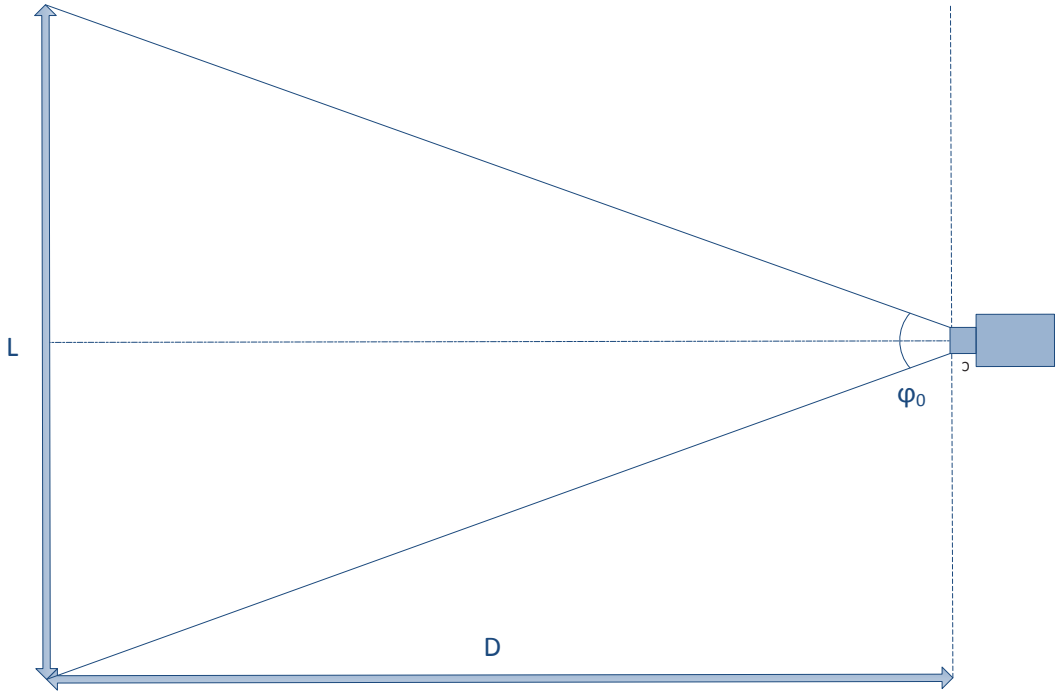


Figure 4.3: Diagram of the set-up to measure the field of view of the camera

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

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 duck's 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 4.4 and an example NCC result can be seen in Figure 4.5.

Table 4.2 shows the distances tested with the distance calculated using the above method. The ranges calculated were inaccurate. Matching can only be achieved to the accuracy of a few pixels without a more complex design, and therefore can

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

(a) Left image (b) Right image

Figure 4.4: Stereo pair of images of a rubber duck on a reel of solder

introduce a small error when matching and cause a large distance error in the calculation.

4.2.5 The Effect of Resolution and Field of View

By simplifying Equation (4.4) to (4.10) shows that the function is at a maximum when $\Delta_x = 1$ and is the maximum distance able to be calculated. The maximum error is found using Equation (4.11) and shows that the error increases with resolution.

$$D(\Delta_x) = \frac{B.x_0}{2\tan\left(\frac{\varphi_0}{2}\right).\Delta_x} \tag{4.10}$$

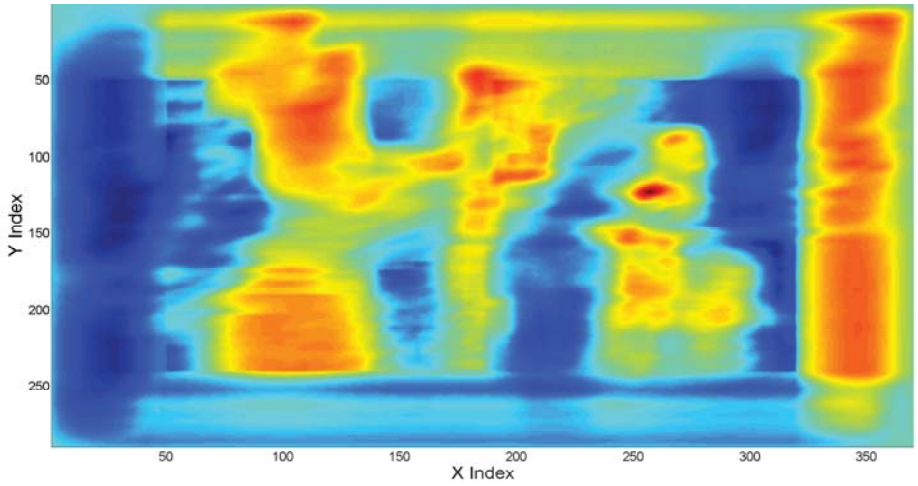


Figure 4.5: NCC results from matching using the duck's head from the right image as the template to the left image

$$Err_{max} = D(1) - D(2) = \frac{B.x_0}{2\tan\left(\frac{\varphi_0}{2}\right)} - \frac{B.x_0}{2\tan\left(\frac{\varphi_0}{2}\right).2} = \frac{B.x_0}{4\tan\left(\frac{\varphi_0}{2}\right)} \tag{4.11}$$

This is because as x_0 increases, the range of $(x_1 - x_2)$ also increases. The minimum detectable distance, Equation (4.12), is not related to resolution, but depends on B and φ_0.

$$D(x_0) = \frac{B}{2\tan\left(\frac{\varphi_0}{2}\right)} \tag{4.12}$$

Figure 4.6 shows the relationship between maximum distance, x_0 and φ_0. By increasing the horizontal resolution, the maximum distance increases, but also increases the error. Increasing φ_0 will lower the maximum distance, but the error will decrease and can be seen in Figure 4.7. A compromise must be made between maximum distance and resolution.

4.2.6 Conclusion

Extensive testing of the effect of separation has been done before (Mrovlje and Vrančić, 2008). Figure 4.8 shows that increasing B will give you a larger range of detectable distances, but with a larger maximum error. Resolution also increases the error and maximum detectable distance. In reality, cameras have not increased only by resolution, but the lens have become wider angled as well.

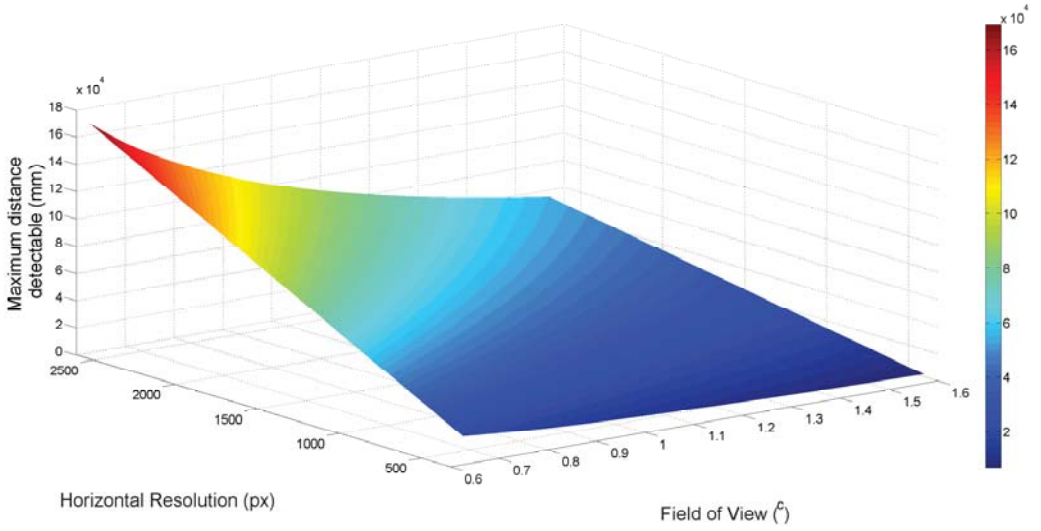


Figure 4.6: A surface plot of the maximum range able to be found with varying φ_0 and x_0. $B = 42mm$, $\Delta_x = 1$

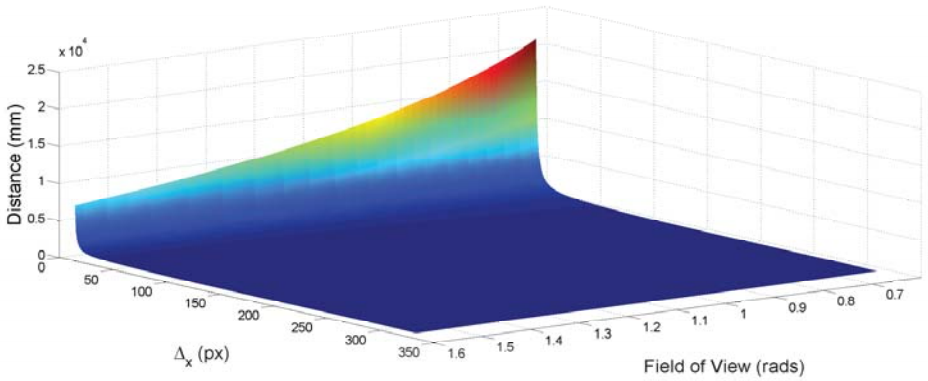


Figure 4.7: A surface plot of the range of distances able to be calculated with a range of field of views. $B = 42mm$, $x_0 = 320$.

In order to improve the ability of these cameras for range finding, adding a wide angle lens will reduce the error at the expense of reducing the maximum range. Alternatively, the robot could use an algorithm to view the object at different perspectives to estimate the distance to it.

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 by using cameras as described, ranges cannot be accurately measured. The separation of objects in the images is a reciprocal function and the data gathered in the test matches this characteristic, see Figure 4.9. This means the system can perceive depth from the separation of the objects, but not accurately calculate the distance.

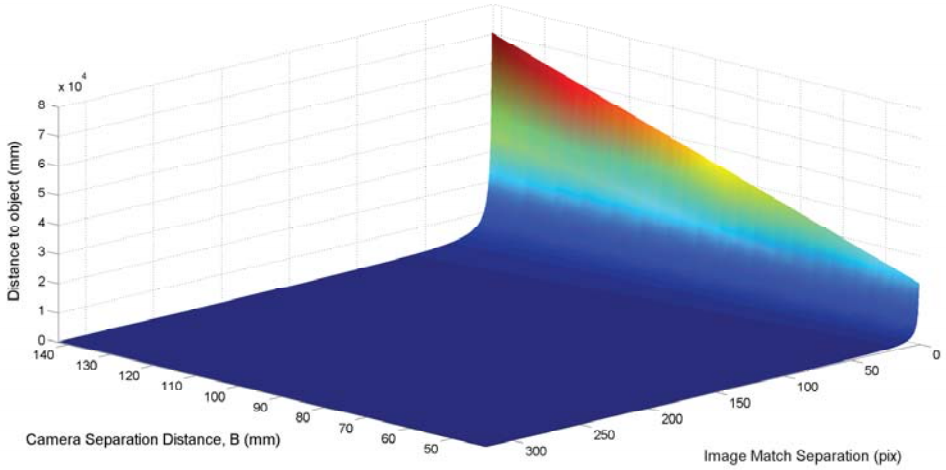


Figure 4.8: 3D surface plot graph showing range of distances that can be calculated over a range of camera separation, B, values

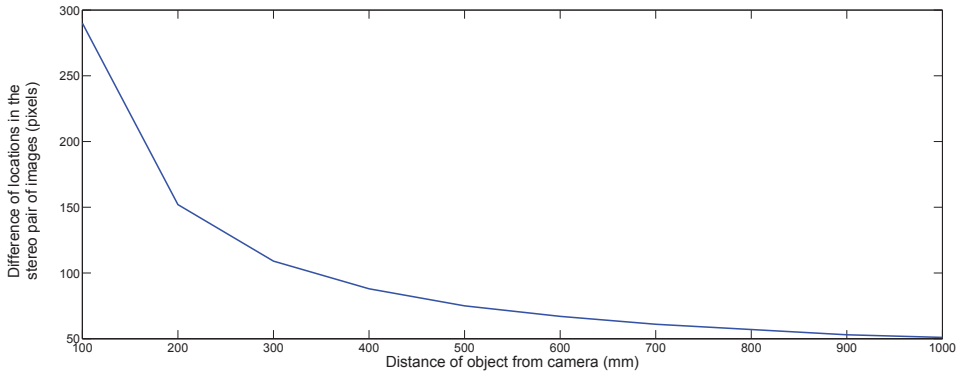
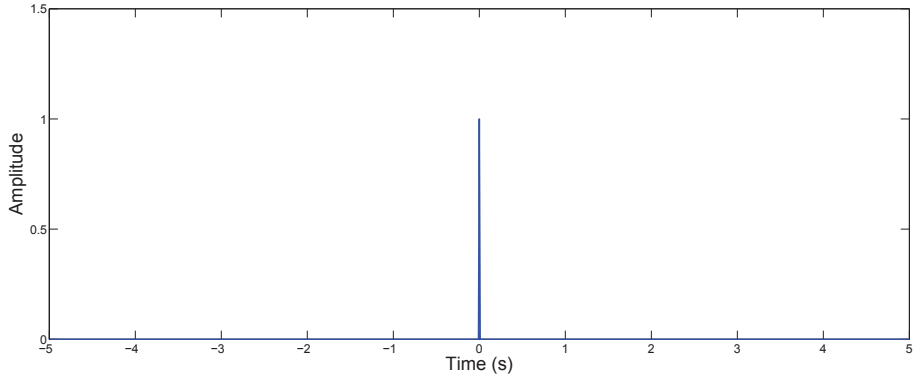


Figure 4.9: Graph showing distance of the object against the difference in the match locations

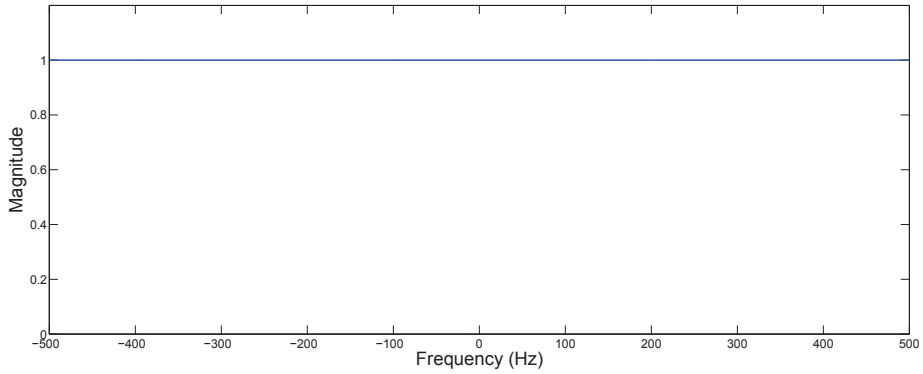
4.3 Fourier Transform

4.3.1 Background Research and the FFT

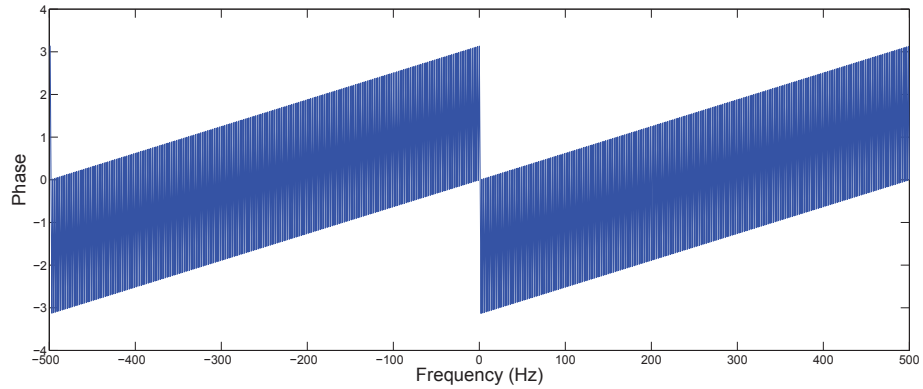
The Fourier transform is a common tool in signal processing used for filer design, system analysis and image processing, as well as other applications. It transforms a time-based signal to the frequency domain, showing the frequency components contained in the signal as a complex number. This is often displayed as magnitude and phase. The Fourier transform is defined in Equation (4.13) and two examples of signals and their Fourier transforms are shown in Figures 4.10 and 4.11.



(a) A graph showing a Dirac function



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

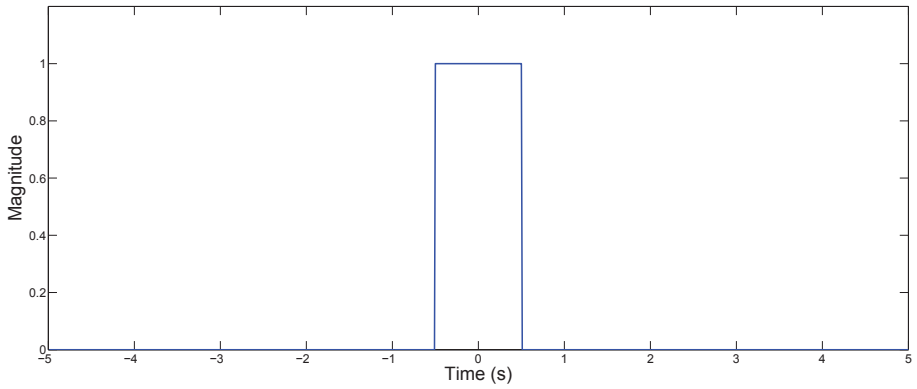


(c) A graph showing the phase of the Fourier transform of the Dirac function

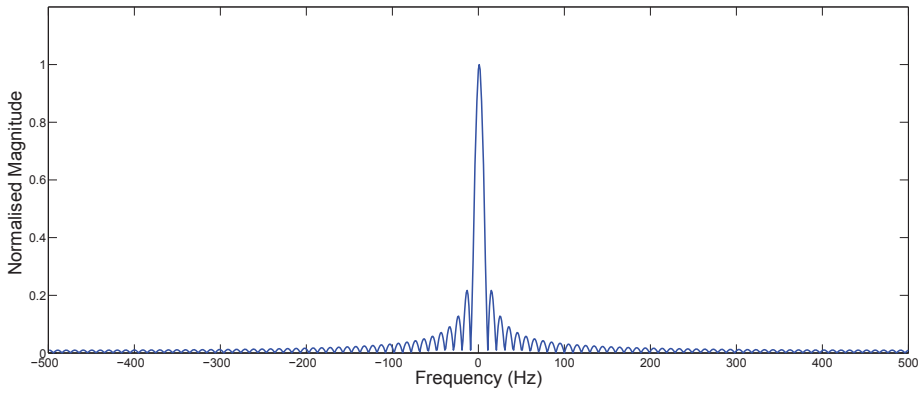
Figure 4.10: A Dirac function, and the phase and magnitude of its Fourier transform

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-\jmath 2\pi f t} dt \tag{4.13}$$

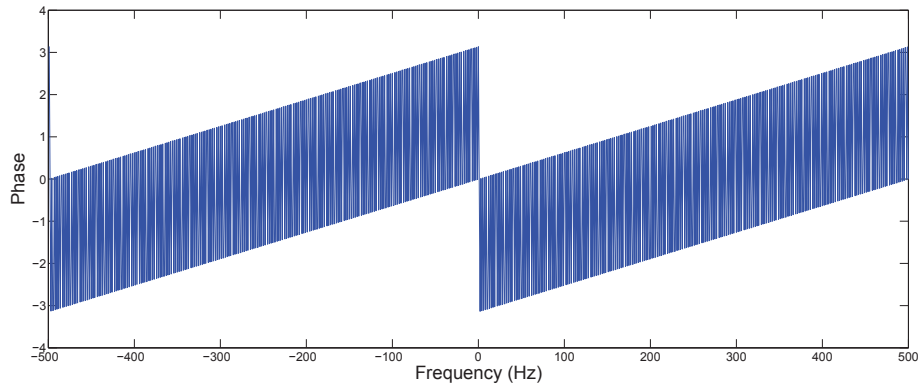
The equation for the Fourier transform in Equation (4.13) is for continuous time.



(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 4.11: A rectangular pulse, and the phase and magnitude of its Fourier transform

A discrete Fourier transform (DFT) exists for finite, equally spaced samples. This is commonly used in digital systems and is defined in Equation (4.14). 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. The FFT is most suitable for use on microcontrollers due its speed of calculation and the availability of code.

$$X[k] = \sum_{0}^{N-1} x[n] e^{-\jmath \Omega_0 k n} \tag{4.14}$$

Where Ω_0 is the sample frequency

A property of the Fourier transform, which is of interest, is the convolution theorem which states that convolution in time is multiplication in frequency and is defined mathematically in Equation (4.15). Cross correlation is defined in Equation (4.16) and related to convolution by Equation (4.16). With images, $f(t)$ is a real signal, its conjugate is exactly the same, $f(t) \equiv f^*(t)$, given that $f(t) \in \Re$. Fourier transforms can be used to calculate cross correlation more efficiently, by multiplying the Fourier transform of an image by the reversed template.

$$\int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau = f(t) * g(t) = X(f) \cdot Y(f) \tag{4.15}$$

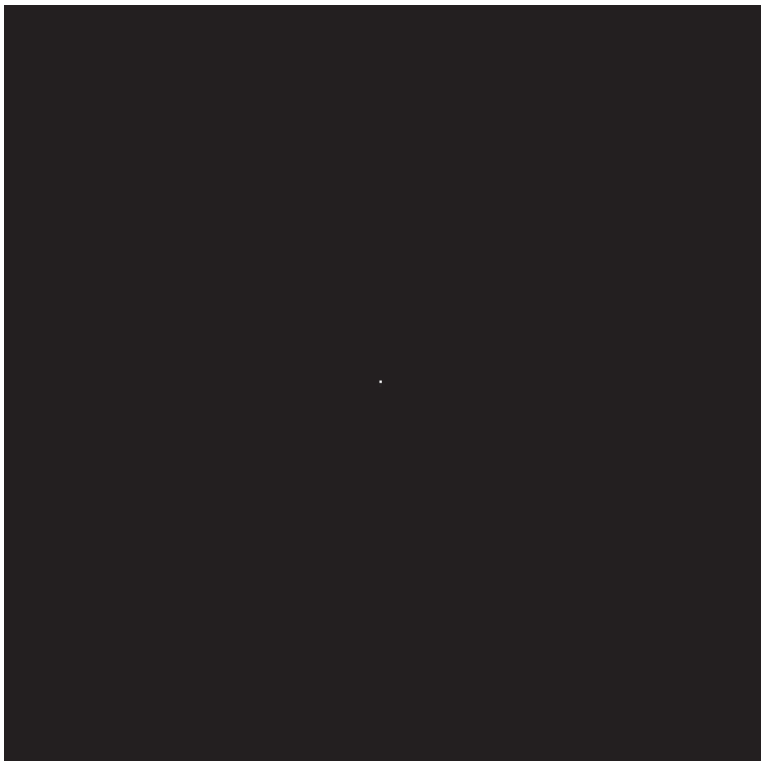
$$\int_{-\infty}^{\infty} f^*(\tau) g(t+\tau) d\tau = f(t) \star g(t) = f'(-t) * g(t) = X(-f) \cdot Y(f) \tag{4.16}$$

4.3.2 Two Dimensional Fast Fourier Transform

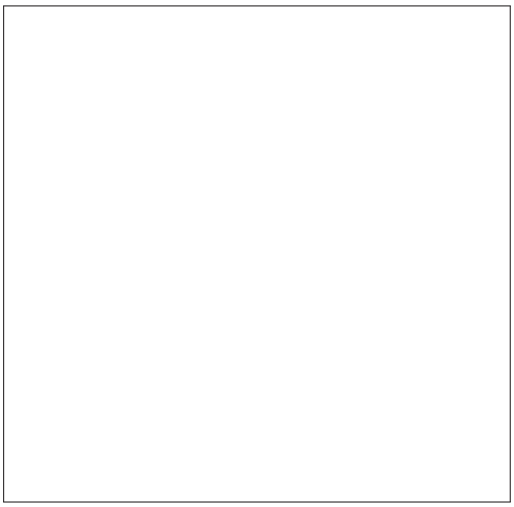
A two dimensional (2D) Fourier transform exists for analysing 2D signals, such as an image. The Fourier Transform is shown in Equation (4.17) and the discrete version is shown in Equation (4.18)

$$F(u,v) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) e^{-2\pi\jmath(xu+yv)} dx dy \tag{4.17}$$

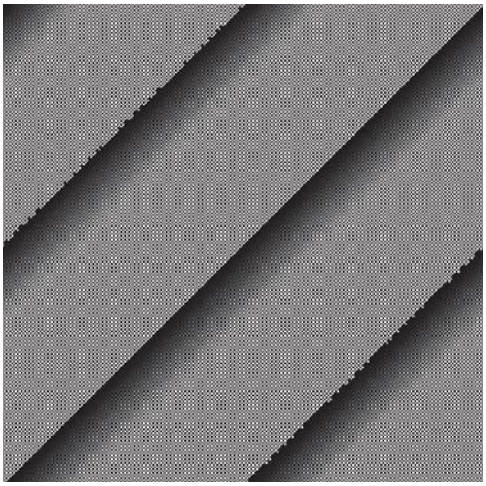
$$F(u,v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) e^{-\frac{2\pi\jmath(xu+yv)}{N}} \qquad x,y,u,v \in \{0 \ldots N-1\} \tag{4.18}$$



(a) An image of a 2D Dirac function



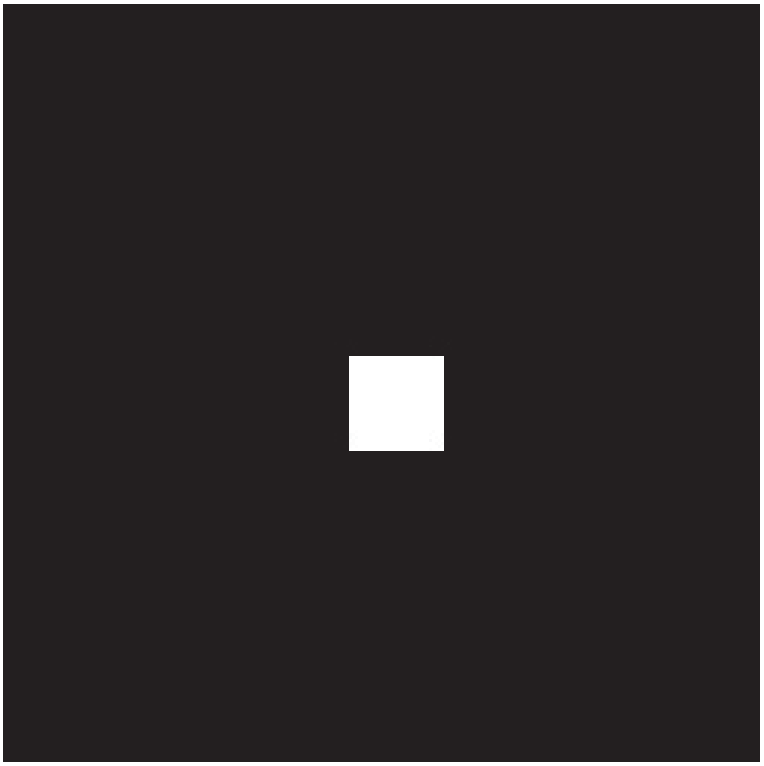
(b) An image of the magnitude of the Fourier transform of the 2D Dirac function



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

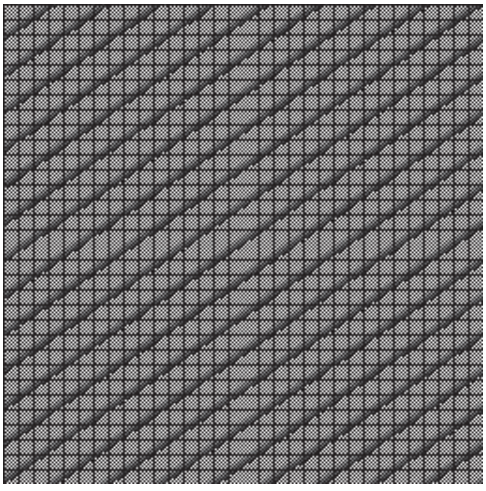
Figure 4.12: A 2D Dirac signal, and the phase and magnitude of its Fourier transform

Figures 4.12 and 4.13 show the 2D equivalent test signals of Figures 4.10(a) and 4.11(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 4.10(b) and 4.12(b)) are both constant values and the rectangular pulses both have a modulus sinc function magnitude (Figures 4.11(a) and 4.13(a)).



(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 4.13: A 2D rectangular pulse signal, and the phase and magnitude of its Fourier transform

The 2D Fourier transform can also be optimised to a FFT algorithm in a similar way to the 1D case. However, this algorithm, sometimes referred to as the Butterfly transform, can only be applied to images with equal dimensions that are a power of 2, without extra effort (Nixon and Aguado, 2012). 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.

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 in fixed point notation and with a length of an even power of 2. This gives a usable dimension of $256px \times 256px$ 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.

A function was made, called *FFT2DCOMPLEX*, to realise the 2D FFT 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 are returned in fixed point notation. A 16 bit representation was chosen over 32 bit due to more functions being 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.

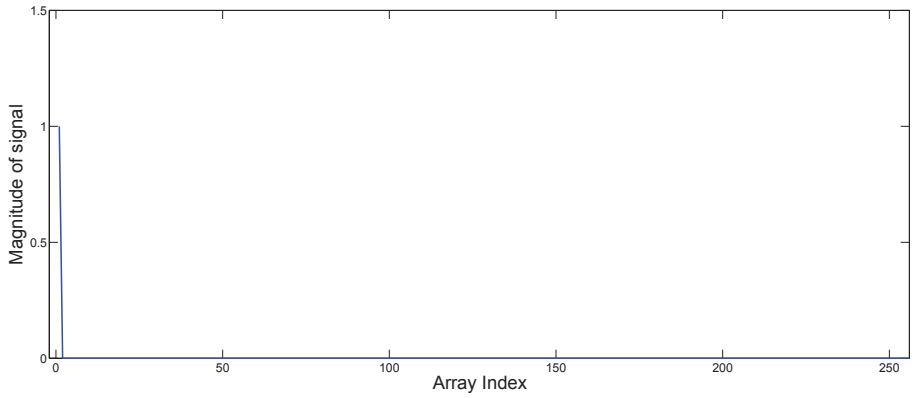
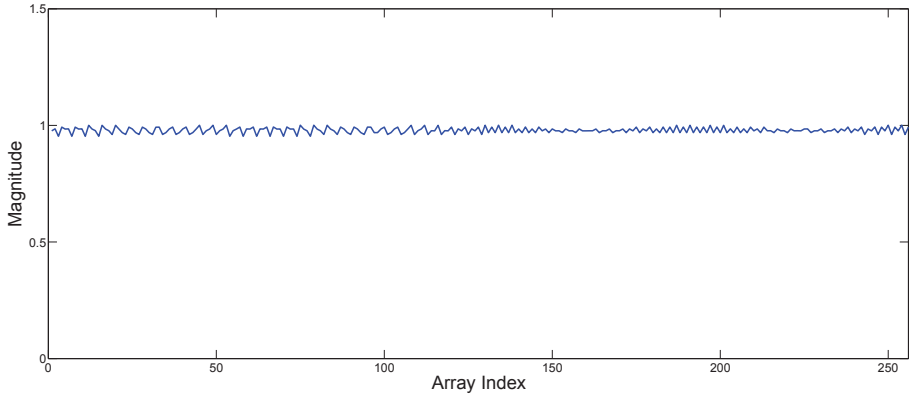


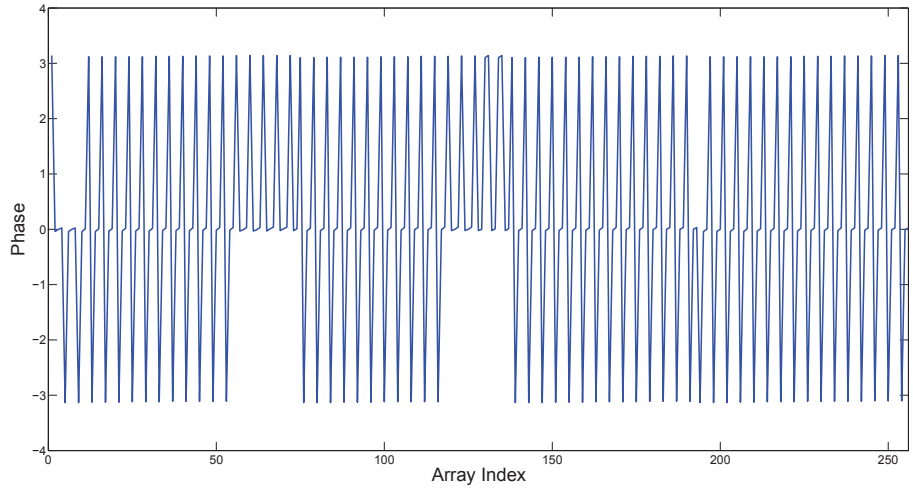
Figure 4.14: Input Dirac signal for AVR fast Fourier transform

Figure 4.14 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 CSV file and read into MATLAB. Figure 4.15 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 4.10(b), they are relatively similar. The phase, however, seems to be very different. Figure 4.10(c) shows what was expected, but the two phase results appear to be quite different. This could be due to MATLAB having more accurate algorithms and a more accurate representation than 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 4.16 is obtained. This, though is not a magnitude of exactly 1.0 as expected, is completely flat and it is computed from the same transformed data. This suggests that there is some internal compensation in the algorithms. The actual value in Figure 4.16 is 0.9897 to 4 decimal places giving an overall error of 1.03%.

Figures 4.17, 4.18 and 4.19 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 that the magnitude calculated from the complex output (Figure 4.18) is different to the result when the magnitude is calculated on the AVR (Figure 4.19). There are also differences in the result from the AVR and the result from MATLAB in Figure 4.11, which can, again, be attributed to the algorithms.



(a) Magnitude of the complex output from the AVR



(b) Phase of the complex output from the AVR

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

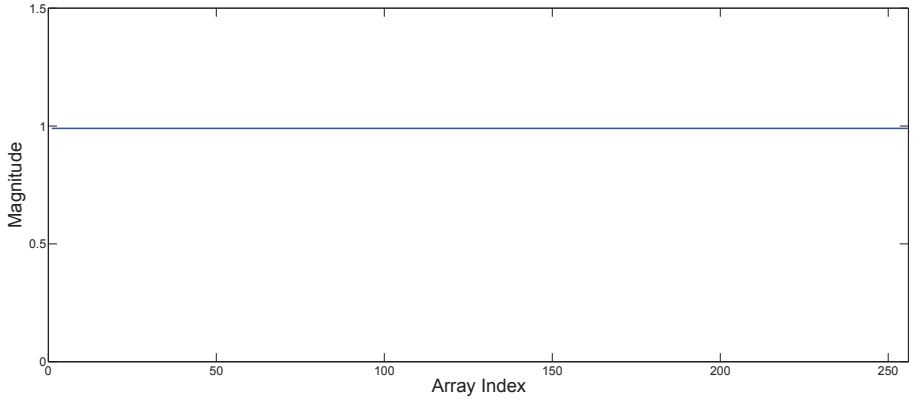


Figure 4.16: Magnitude calculated by the AVR of the Fourier transform of a Dirac function

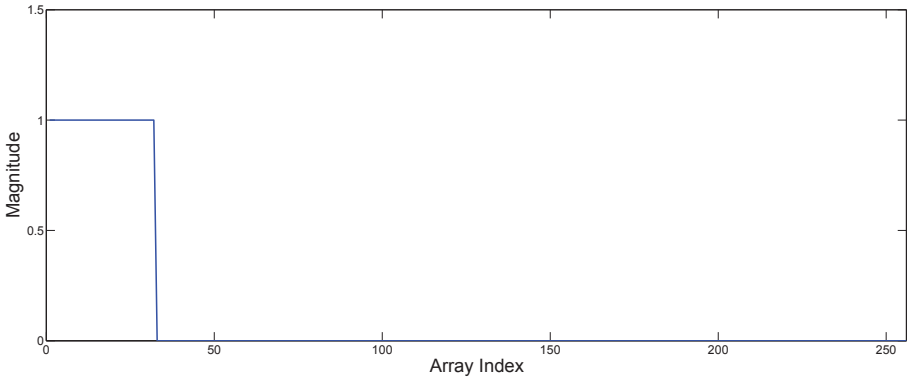
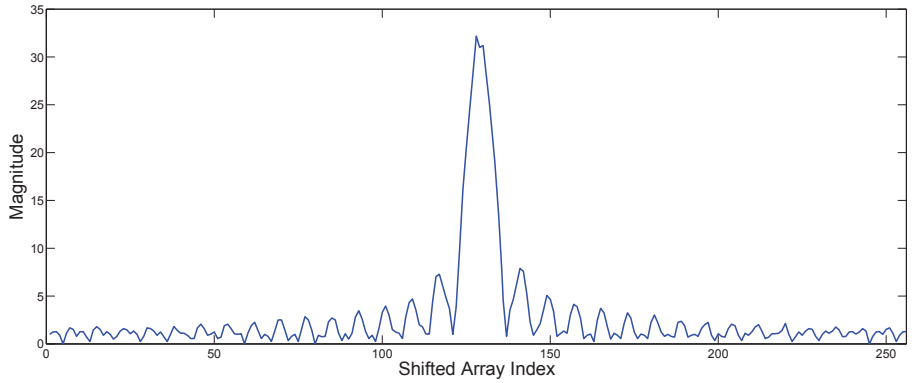
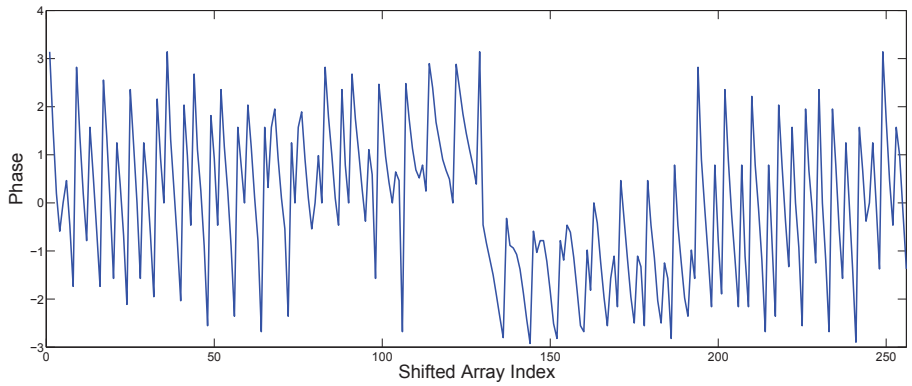


Figure 4.17: Input rectangular pulse for AVR fast Fourier transform



(a) Magnitude of the complex output from the AVR



(b) Phase of the complex output from the AVR

Figure 4.18: Output phase and magnitude of the complex output from AVR fast Fourier transform of a rectangular pulse

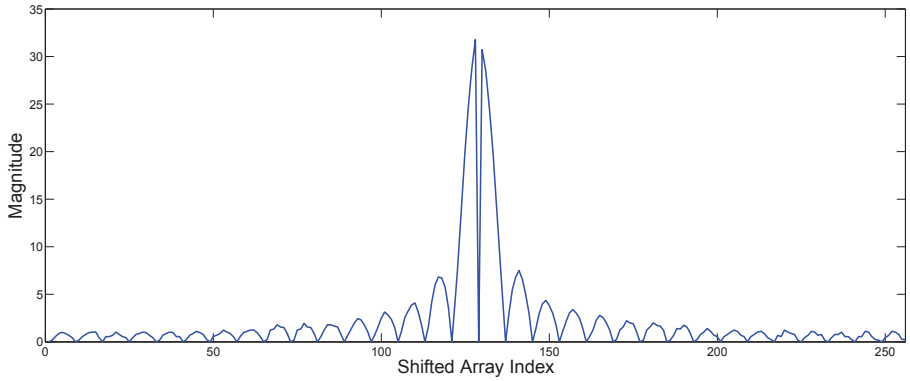


Figure 4.19: Magnitude calculated by the AVR of the Fourier transform of a rectangular pulse

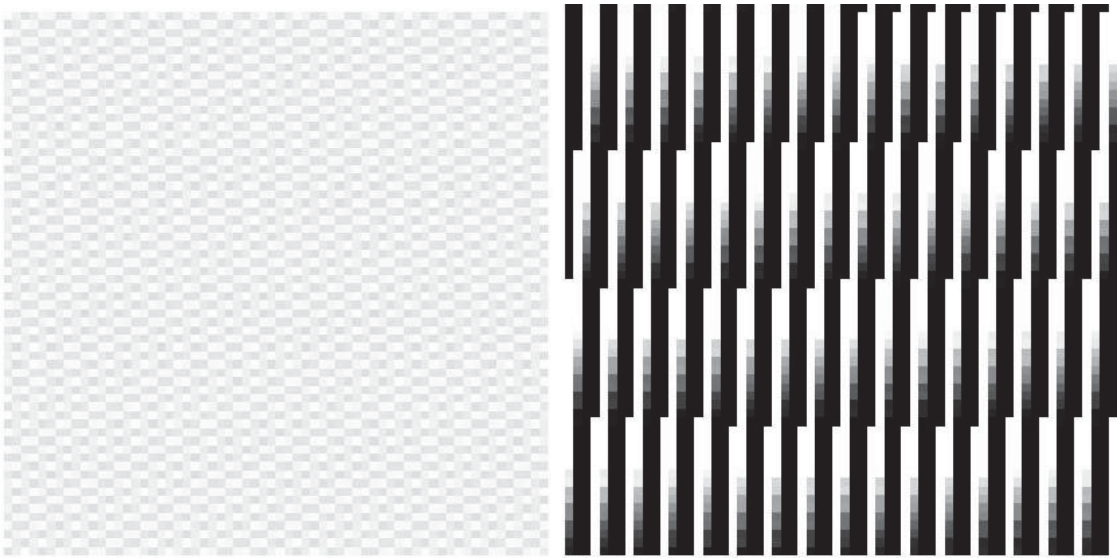
4.3.4.2 2D FFT Test

Two test signals were used to test the 2D FFT on the AVR; a Dirac signal and a square wave, seen in Figures 4.12(a) and 4.13(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 were normalised to omit the effects of the fixed point notation and the data shifted so that frequency 0 was in the centre. These tests were done with a 64×64 2D data as with a 256×256 array the AVR runs out of internal RAM to calculate the transform.

The result of the Dirac test is shown in Figure 4.20. 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 4.16. The phase has similar issues as 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 4.21(a) is very similar, with a distinct peak in the centre (frequency 0) and a sinc function extending vertically and horizontally from this. Again, the phase (Figure 4.21(b)) seems to differ a lot from the expected result in Figure 4.13(c).

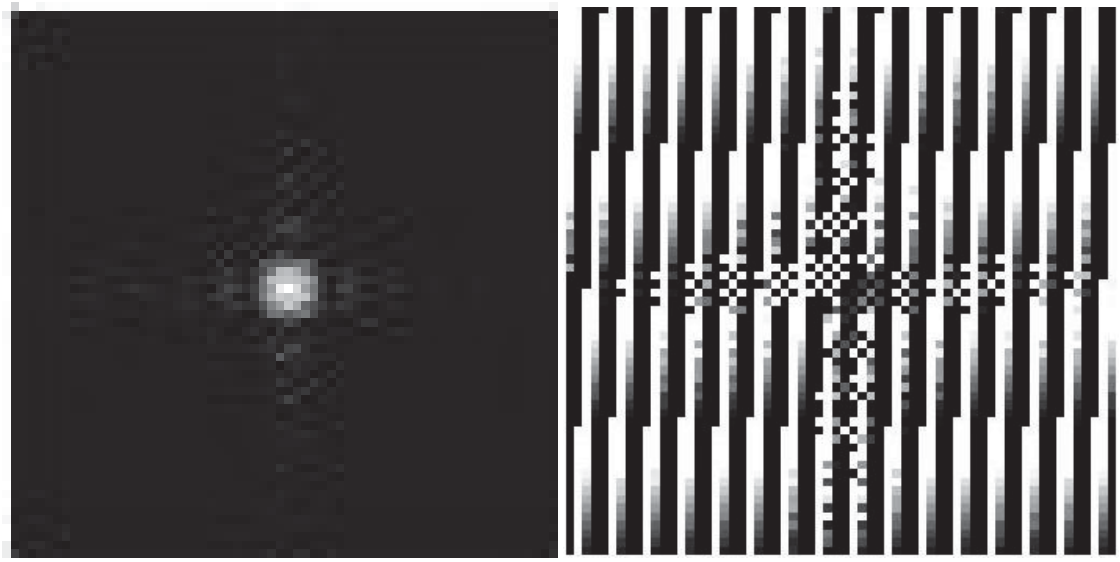
4.3.5 Conclusion

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



(a) Normalised magnitude (b) Phase

Figure 4.20: Images of the phase and magnitude of the complex data returned from the 2D FFT on the AVR of a 2D Dirac function



(a) Normalised magnitude (b) Phase

Figure 4.21: Images of the phase and magnitude of the complex data returned from the 2D FFT on the AVR of a 2D square function

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	-

reduced by increasing the internal clock speed on the AVR, potentially taking it up to 33MHz, and therefore halving the time to compute. A $64px \times 64px$ image, however, is not practical for the application. Larger transforms can be done, but further development is required to utilise the external RAM more efficiently.

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 compromises of 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 has been shown to work to 4.5% accuracy over a $300mm$ distance.

Stereo image pairs can be captured and stored on an SD card in a FAT32 file system. The SD card is also used for transferring images to and reading log files on a computer. Images are stored in QVGA format ($320px \times 240px$) 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 (DMA) 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. A wide angle lens could be added to the system to more accurately measure distances but at the expense of reducing the maximum range.

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 proved to be fairly accurate in testing.

All aspects implemented on the robot have been shown to be functional. A faster processor would have been more suitable 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 reduced the amount of hardware design needed to be done. Existing image processing libraries could then have been used to gain more functionality.

Though some aspects of the specification were not fulfilled completely, the progress made during this project, and the challenges that have been overcome, provide a solid platform for further development.

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 a computer, and search algorithms can be implemented alongside distance calculations to make the robot aware of its surroundings.

5.1 System Operation and Performance

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 execute. Debug mode can be entered by the relevant

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)

command 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 system is able to move reasonably accurately over a $30cm$ distance. The robot can move in a straight line or rotate on the spot. Photos are captured in colour with a resolution of $320px \times 240px$ and stored to an SD card.

The final robot can be seen in Figure 5.1.

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

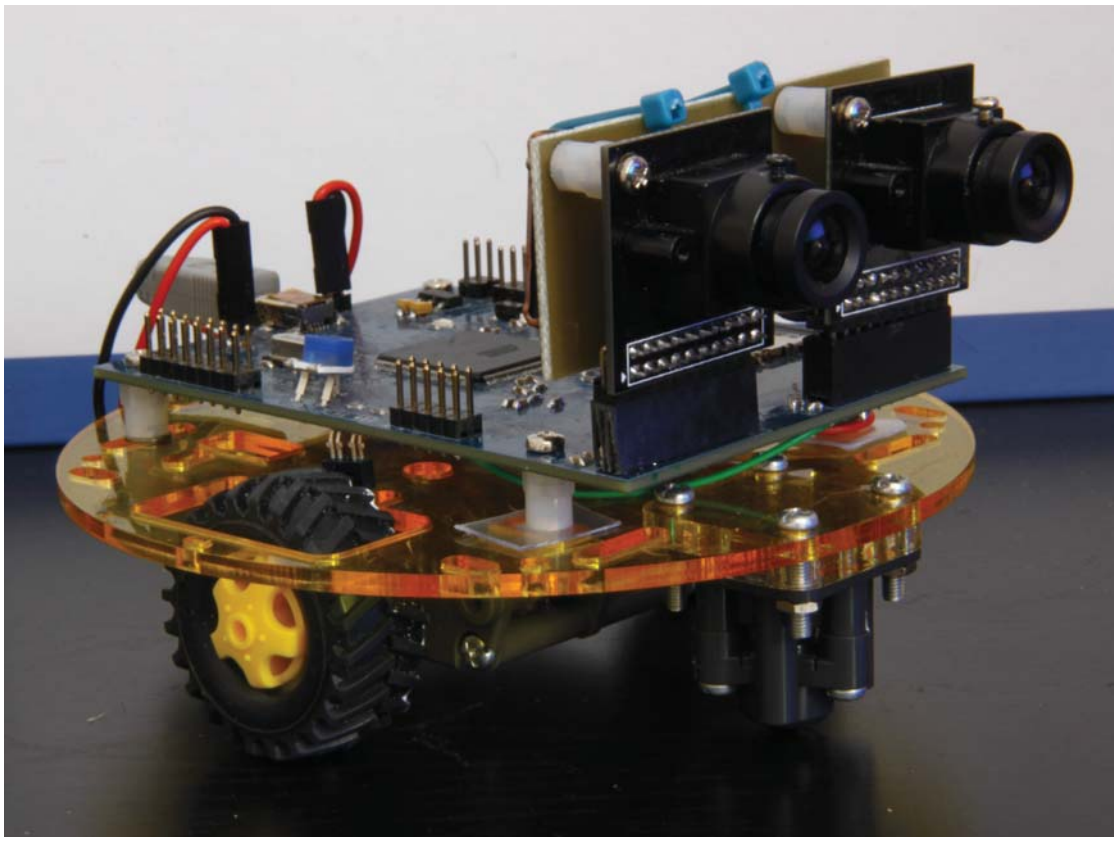


Figure 5.1: The completed robot

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

LED						Meaning
MOTOR	**2**	**3**	**4**	**5**	**6**	
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