

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

8th 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**

This report describes the research, design, build and test of a stereoscopic robot using two Omnivision OV7670 cameras and an Atmel AT32UC3C0512C 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 PCB that was designed is also tested.

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 roam unintelligently and capture photos or be used as a command line terminal to control by a user.

Contents

List of Symbols	xv
Acknowledgements	xvii
1 Introduction	1
1.1 Project Management	2
2 Research	5
2.1 Hardware Research	5
2.1.1 Microcontrollers	5
2.2 Firmware	6
2.2.1 Camera	6
2.2.2 Atmel Software Framework	7
3 Hardware and Firmware Development	9
3.1 Camera	9
3.1.1 Single Camera Operation	10
3.1.2 Dual Camera Operation	12
3.2 SD Card	13
3.2.1 Storing Images	13
3.2.2 User Interface	14
3.3 Motor Driver Development	16
3.3.1 Hardware	16
3.3.2 Firmware Development	17
3.3.3 Testing	20
3.3.4 Conclusion	23
3.4 PCB Development	23
3.4.1 Circuit Design	23
3.4.2 PCB Design	24
3.4.3 PCB Testing	27
3.4.3.1 UART Test	28
3.4.3.2 SD Card Test	28
3.4.3.3 LED Test	30
3.4.3.4 SDRAM Test	30
3.4.3.5 I^2C Test	31

3.4.3.6	Camera Test	33
3.4.3.7	Motor Driver Test	33
3.4.4	PCB Faults	34
3.4.4.1	TCRT1010 Footprint	34
3.4.4.2	SDRAM Footprint	34
3.4.4.3	SDRAM Chip Select	34
3.4.4.4	SDRAM Data Line Resistors	35
3.4.4.5	Camera Interrupt Line	36
3.4.4.6	Motor Driver Pinout	37
3.4.5	PCB Conclusions	37
3.5	Conclusions	38
4	Investigation into Vision Algorithms	41
4.1	Matching Algorithms	41
4.1.1	Sum of Absolute Differences	42
4.1.2	Sum of Squared Differences	42
4.1.3	NCC	42
4.1.4	Comparison	43
4.1.5	Conclusion	45
4.2	Range Finding	45
4.2.1	Derivations	45
4.2.2	Summary	45
4.2.3	Field of View	46
4.2.4	Testing	46
4.2.5	Conclusions	48
4.3	Fourier Transform	49
4.3.1	Background Research and the FFT	49
4.3.2	Two Dimensional Fast Fourier Transform	53
4.3.3	Implementing the FFT	56
4.3.4	Testing of the FFT on AVR	56
4.3.4.1	1D FFT Test	56
4.3.4.2	2D FFT Test	59
5	Conclusions and Further Work	63
5.1	System Operation	64
A	Gantt Chart	69
B	Circuit Diagrams	71
B.1	OV7670 Breakout Board Schematic	71
B.2	Il Matto and Dual Camera Schematic	71
B.3	The Columbus Circuit Diagram	71
C	PCB Design	81
C.1	PCB Top Side	81

C.2 PCB Bottom Side	81
D Costings and Components	85
E Contents of Files	87
F Bitmap File Format	89
F.1 Bitmap File Format	89
G Range Finding Derivations	93
G.1 Object is between the Cameras	93
G.2 Object is to the same side in each camera	95
G.3 Object is in front of a camera	96
H Source Code	99
H.1 C Code for AVR	99
H.1.1 The Columbus Source Code	99
H.1.1.1 main.c	99
H.1.1.2 Bitmap.c	109
H.1.1.3 CustomDevices.h	110
H.1.1.4 ImageProcessor.h	112
H.1.1.5 ImageProcessor.c	112
H.1.1.6 MotorDriver.h	118
H.1.1.7 MotorDriver.c	120
H.1.1.8 OV7670.h	127
H.1.1.9 OV7670.c	133
H.1.1.10 OV7670.c	144
H.1.1.11 PCA9542A.h	148
H.1.1.12 PCA9542A.c	148
H.1.1.13 SD_Card.h	149
H.1.1.14 SD_Card.c	150
H.1.1.15 TWI.c	160
H.2 MATLAB Code	161
H.2.1 Image Matching Algorithms	161
H.2.1.1 Match.m	161
H.2.1.2 MatchAll.m	162
References	163

List of Figures

1.1	The base of the robot	2
3.1	RGB565 pixel format	10
3.2	Signals generated to control the OV7670 capture and read	11
3.3	An Example Image taken using the OV7670 and stored as a Bitmap on the SD Card	14
3.4	Prototype of Dual Camera operation.	15
3.5	Prototype of streaming camera images to TFT screen.	16
3.6	Circuit diagram of Optosensor	17
3.7	Graph of Wheel Angle against the Voltage read by the AVR	17
3.8	A State Machine showing the operation of the <i>Motor_Execute</i> method	20
3.9	Dimensions of Interest for Robot Movement	21
3.10	A plot of Expected Distance against the measured data. Line indicates the average of the data at each point.	22
3.11	A hierarchical diagram of the robot	24
3.12	PCB with no components. Left: Top View. Right: Bottom View . .	27
3.13	SDRAM Chip shown against its footprint.	35
3.14	Motor Driver error. Outputs incorrectly connected	37
3.15	Pictures of the built PCB.	39
4.1	Stereoscopic Test Images from MATLAB Examples	41
4.2	Result Graphs of Comparison Algorithms	44
4.3	Stereo pair of images of a rubber duck on a reel of solder	48
4.4	NCC results from matching using the ducks head from the right image as the template to the left image	48
4.5	3D Surface plot graph showing range of distances that can be calculated over a range of camera separation, B, values.	49
4.6	Graph showing Distance of the object against the difference in the match locations.	49
4.7	A Dirac signal and the phase and magnitude of its Fourier Transform	51
4.8	A 2D Rectangular pulse and the phase and magnitude of its Fourier Transform	52
4.9	A 2D Dirac signal and the phase and magnitude of its Fourier Transform	54
4.10	A 2D Rectangular Pulse signal and the phase and magnitude of its Fourier transform	55

4.11	Input Dirac Signal for AVR fast Fourier transform	57
4.12	Output phase and magnitude of the complex output from AVR fast Fourier transform of a Dirac function	57
4.13	Magnitude calculated by the AVR of the Fourier transform of a Dirac function	58
4.14	Input Rectangular Pulse for AVR fast Fourier transform	58
4.15	Output phase and magnitude of the complex output from AVR fast Fourier transform of a Rectangular Pulse	58
4.16	Magnitude calculated by the AVR of the Fourier transform of a Rectangular Pulse	59
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	60
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	61
A.1	Gantt Chart of how time will be spent in the areas of the project .	70
B.1	The circuit diagram for the OV7670 breakout board	72
B.2	The circuit diagram for Dual Cameras using the Il Matto Board .	73
B.3	The Columbus Circuit Diagram Page 1	74
B.4	The Columbus Circuit Diagram Page 2	75
B.5	The Columbus Circuit Diagram Page 3	76
B.6	The Columbus Circuit Diagram Page 4	77
B.7	The Columbus Circuit Diagram Page 5	78
B.8	The Columbus Circuit Diagram Page 6	79
C.1	The Top side of the CAD Design of the PCB	82
C.2	The Bottom side of the CAD Design of the PCB	83
G.1	Problem 1 - Object is between the Cameras	94
G.2	Problem 2 - Object is to the same side in both cameras	96
G.3	Problem 3 - Object is directly in front of a camera	97

List of Tables

1.1	A list of risks and the prevention steps taken to reduce their impact	3
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)	7
3.1	A table comparing different image formats available (Fulton, 2010)	13
3.2	Pin Connections of the ATMega644P for Dual Camera Operation.	15
3.3	Results of Motor Distance Test	22
3.4	Results of motor speed test	22
3.5	The Pinout of the AVR for the circuit. ‘-’ means unavailable and blank means unused	25
3.6	A table showing examples of the incorrect data returned from the SDRAM	36
4.1	Table of results to calculate the field of view of the camera	47
4.2	Results of range finding test	47
4.3	Number of clock cycles taken to calculate the Transform of 64 or 256 long data set	60
5.1	Table showing the Auto Run Commands Implemented	65
5.2	Table showing the available Debug Commands	66
5.3	Table show the meaning of the LED lights. F - Flashing, X - Don’t Care	67
D.1	A table of all components used and their costs.	85
D.2	All components and their values (if applicable)	86
F.1	Format of a Bitmap file with values used, to write an image from the camera to an SD Card	89

Listings

3.1	UART Test Code	28
3.2	UART Test Code	28
3.3	SDRAM Test Code	30
3.4	I^2C Test Code	31
3.5	Result of I^2C bus scan with Channel 0 of the I^2C MUX selected	32
3.6	Camera Test Code	33
3.7	Motor Test Code	33

List of Symbols

Abbreviations

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

Images

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

Motors

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 searched an area and return an occupancy map ([Thrun, 2003](#)). However, due to time constraints, the vision part of the project was not implemented. The end robot is able to capture stereo image pairs, move with reasonable accuracy and do some image processing. The theory for distance measuring is discussed and prototyped in MATLAB but is not implemented on the AVR.

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.

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 will use the base seen in figure [1.1](#) and use two OmniVision OV7670 cameras delivering QVGA format images.

The final robot designed could be used for a variety of applications. As a mapping robot, the device could be used by estate agents to measure room sizes. The robot



Figure 1.1: The base of 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 are looked at and are summarised in table 1.1. A Gantt chart of how time should be spent can be seen in figure A.1.

The project will be designed in stages - first, gaining operation of all the basic sections; movement, image capturing, image detection algorithms etc. These will then be brought together once tested to create the final product.

Risk	Severity	Prevention
Parts 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) Algorithms

Hardware and firmware research will be discussed in this section. Vision algorithms are looked at in detail in chapter [4](#).

2.1 Hardware Research

2.1.1 Microcontrollers

The robot is to be designed with a budget of £80 (not including P.C.B.). The choice of microcontroller will be an important one, as a compromise between cost, power and usability must be made. There are two main brands of microcontrollers present in the consumer market: ARM and Atmel 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, SDRAM 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. They range from 8-bit, low clock speed devices for the hobbyist (ATMega and ATTiny series), to an improved 8-bit variant (XMega), and a 32-bit design (AT32UC3). XMegas and AVR32s tend to have higher clock speeds than the ATMegas. The AVR core also has a Harvard RISC architecture, and is mainly 8-bit. Atmel devices often have on board peripherals such as I^2C (called TWI on AVRs), SPI and ADCs, as well as a number of different memories: Flash, EEPROM and SDRAM. 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 SDRAM and Flash. All devices looked at use 3.3V supply and have basic communication protocols (SPI, I^2C and USART). Overall, the AT32UC3C0512C is the best choice with a high throughput, a floating point unit and a vast amount of GPIO 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 camera by OmniVision. Steve Gunn provided source code for use on the Il Matto development board which uses an ATMega644P and also has an onboard SD Card reader. The original code 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.

	ARM Stelllaris	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 ² C , SSI, MAC, CAN, EPI, USB, US- ART, 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 TWI, SPI and timers. These application notes are aimed at older devices like the ATTiny and ATMega and are generally written for IAR Embedded Workbench compiler, as opposed to the AVRGCC compiler used within Atmel Studio.

Chapter 3

Hardware and Firmware Development

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

The following section is broken down into the following parts:

- [3.1 Camera Code](#)
- [3.2 SD Card](#)
- [3.3 Motor Development](#)
- [3.4 PCB Development](#)

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 needed and a 24MHz clock mounted. The schematic for the device can be seen in appendix [B](#).

Original code for the camera operation was given by Steve Gunn, which was used to gain the operation required. This code streamed continuous video to a TFT

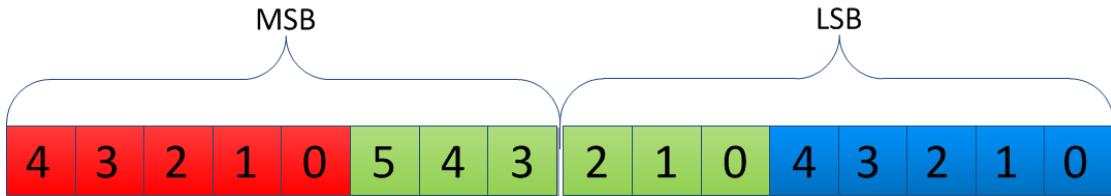


Figure 3.1: RGB565 pixel format

screen. 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 are compatible. The original code 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 AVR s)¹. This communication bus is used to set up the control registers of the OV7670 to enable operation in the correct format.

RGB565 is a 16 bit pixel representation where bits 0:4 represent the blue intensity, 5:10 is the green intensity and 11:15 represent the red intensity (see figure 3.1). 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 to store images in. It is easily converted to grey scale and is a widely used format.

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 AVR to be able to respond to the camera quick enough (ATMegas typically clocked at 8-12MHz). 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 full frame of data, and HREF is high when the data being output is valid. The pixel 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.

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 ”

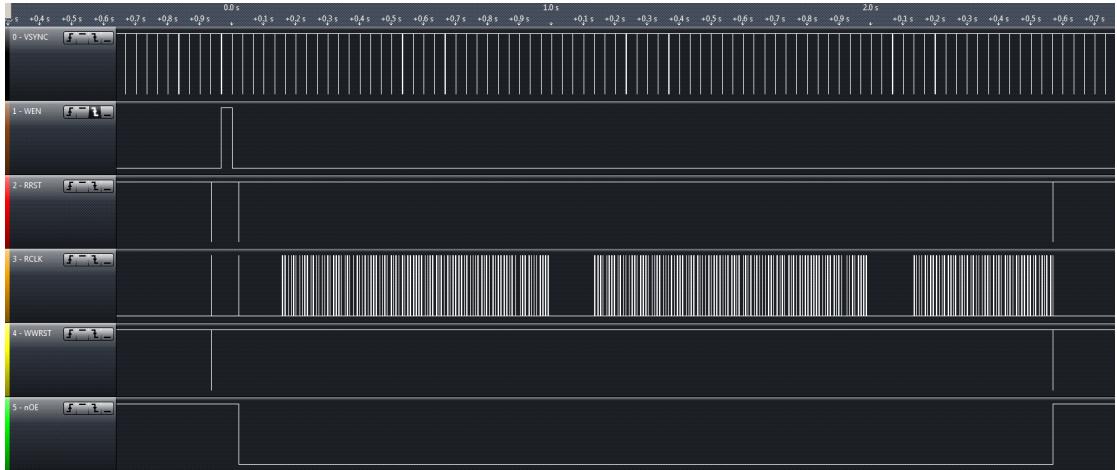


Figure 3.2: Signals generated to control the OV7670 capture and read

In order to acquire a full frame, the first VSYNC pin is set up to interrupt the AVR to enable WEN. The camera will output an entire frame of pixel data and store it into the buffer. When the second VSYNC is received, the WEN signal is disabled, stopping any more data being stored. At this point, the FIFO buffer contains all the image data.

To obtain the data from the buffer, the AVR sets output enable and pulses the read clock. Valid data is available on the input port while RCLK is high. All the data is then read in half a pixel at a time. The entire operation can be seen in figure 3.2.

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 straight to a desktop computer via a COM Port using USART. A simple desktop program was written in C# to receive and store all the data, and to make a Bitmap image from the data. 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 is used as FAT file system so that data can be looked at by a user on a computer. Text log files are 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 deals with the protocol bit by bit in software. This takes up memory and is not reusable for other operations.

An I^2C multiplexer sits on the bus and has multiple output buses. The master can then address the multiplexer and select whether to pass the bus to bus 0, bus 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, but not as much as using SCCB. The main disadvantage to the I^2C MUX is the extra hardware needed; firstly the MUX itself, but also 7 extra resistors to pull up the two extra buses and the three interrupt lines must be added.

Overall, the disadvantages posed by using a MUX are small, so a multiplexer will be 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.

Two ISRs are used to control the relevant WEN signal when VSYNC is triggered. When taking a photo, both frames are taken at a time period close together to capture the same scenario. The data for both images are read back individually by the AVR.

Operation to read an image is identical to using one camera. However, an ID number is passed through the functions to make a decision on the pins to use to read the buffer and to enable the output. Care was taken to avoid bus contention,

	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](#))

but no checking procedure is explicitly in place. Both images are then read back from the buffers and stored to memory.

3.2 SD Card

To use the SD card, the FATFS library ([Electronic Lives Manufacturing, 2012](#)) was used. The library supplies all the functions for writing a FAT File System in the files *ff.c*, *ff.h*, *ffconf.h*, *diskio.c*, *diskio.h* and *integer.h*. The *diskio.h* functions control what device is being used - SD/MMC Card, USB drive etc. The *ff.h* header contains all the functions to write to in a FAT File system.

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 be used for other devices within the system as well.

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 some 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



Figure 3.3: An Example Image taken using the OV7670 and stored as a Bitmap on the SD Card

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 in a more powerful environment. Figure 3.3 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.

3.2.2 User Interface

The ATMega 664P pinout for the dual camera operation can be seen in table 3.2. Due to a lack of available GPIO pins, an ATMega168 was added on the I^2C bus to act as a port extender. The ATMega168 accepts a read or write command. A write places the written data on Port D and a read returns any button pressed that occurred on Port C. When a button is pressed, this is stored in the ATMega168 until a read has been done. This is so the master (644P) does not miss any button presses while busy doing lengthy operations such as writing an image. The code is based on Application Note AVR311 ([Atmel Corporation, 2007](#)), written for IAR

	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.

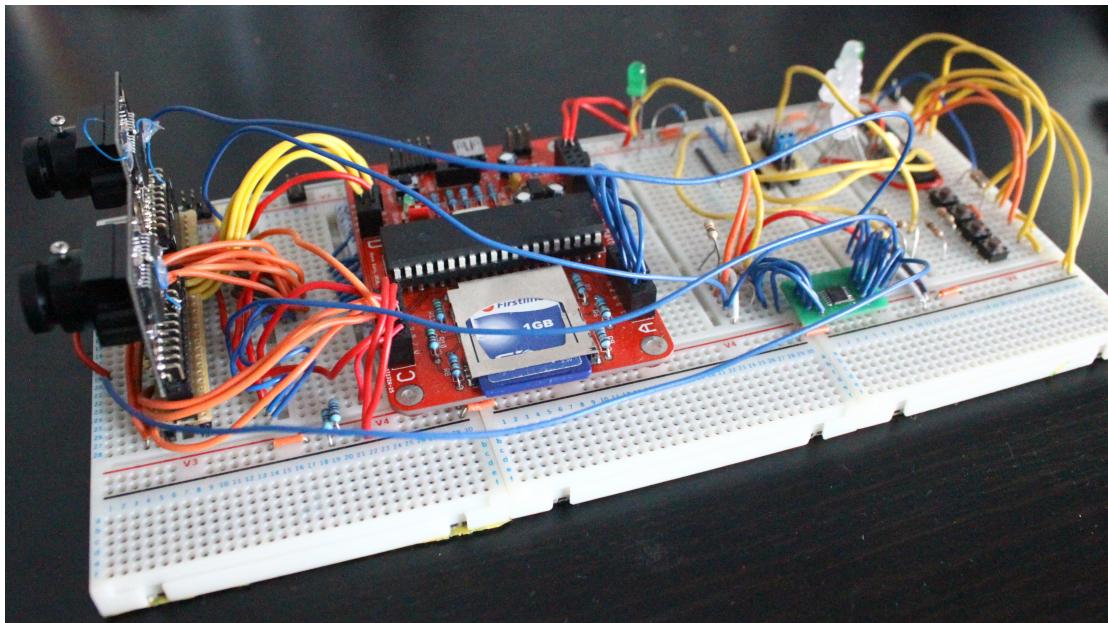


Figure 3.4: Prototype of Dual Camera operation.

Compiler. This code was altered to compile with GCC under Atmel Studio. AVRs contain a hardware based I^2C protocol that is interrupt driven. The ISR of the TWI vector is a state machine which loads the data to send, stores received data, responds to acknowledges and address calls and deals with bus errors that can occur.

The entire prototype for the dual camera operation can be seen in figure 3.4. This circuit was developed to test the cameras and is used to develop the PCB in section 3.4. Another prototype was also made to stream the camera images to a TFT screen. This helped with testing cameras and focusing them. Figure 3.5 shows the set up during operation.

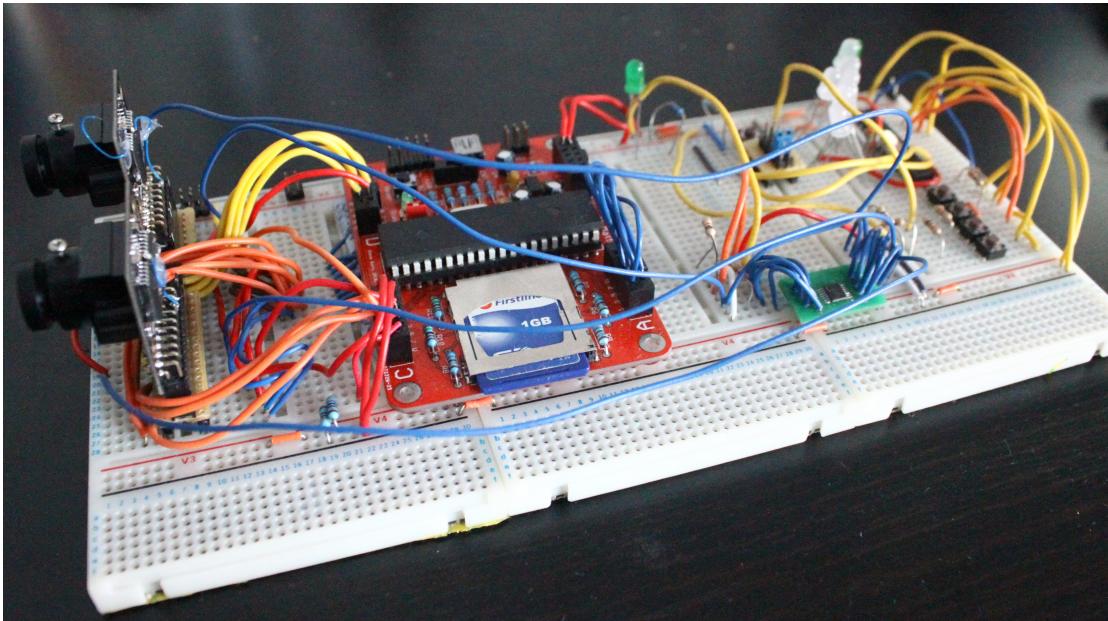


Figure 3.5: Prototype of streaming camera images to TFT screen.

3.3 Motor Driver Development

3.3.1 Hardware

Tachometers are devices used to measure rotational speed of a shaft. Tachometers 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 detected by the sensor is measured and, by knowing the size of the wheel, speed can be calculated.

Here an optosensor, the TCRT1010 made by [Vishay Semiconductors \(2012\)](#), is used to measure rotations of the wheel and used to 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 3.6 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 tippex marks, "tabs", were applied to the wheels at regular intervals. These tabs reflected IR, resulting in a lower collector voltage and thereby giving a way to detect wheel rotation, but not direction. Figure 3.7 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,

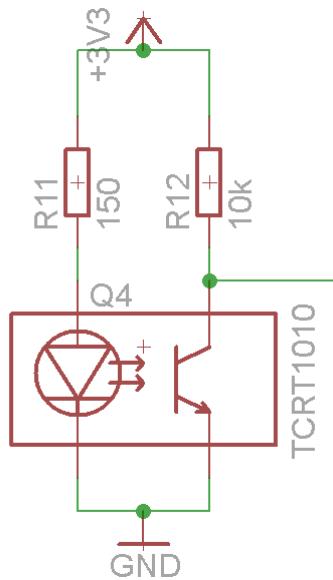


Figure 3.6: Circuit diagram of Optosensor

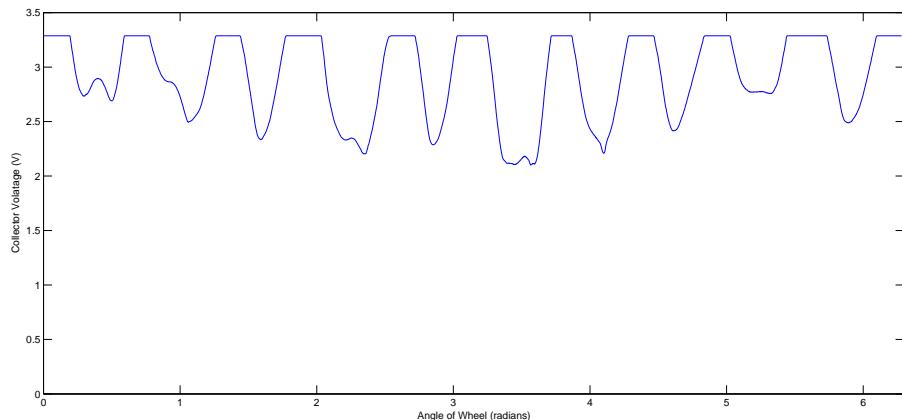


Figure 3.7: Graph of Wheel Angle against the Voltage read by the AVR

and ten dips in the voltage can be seen in figure 3.7. There is a lot of noise that exists on this line due to the non-uniform white tabs applied. However, the dips are prominent and can be detected with the correct threshold voltage.

3.3.2 Firmware Development

The basic outline of the firmware is a set up to calculate how many counts are needed to move the distance, set the inputs to the motor drivers and use PWM to control the speed. The number of times the tabs pass the sensor are counted and when the count reaches the correct number, the motors are stopped. The robot

can move in either a straight line or rotate on the spot. More complex movements, for example arcs, would require more accurate tachometers and are not discussed here.

Moving in a straight line takes a parameter of how far to move as a signed integer and calculates the total number of counts that need to occur. This can be calculated using equation (3.1). This value is put in a global variable so that other methods have scope to it.

$$\text{Counts} = \delta \times \frac{\gamma}{C_w} \quad (3.1)$$

For rotation, the radius from the centre of the robot to the wheels needs to be known, see figure 3.9(a). The circumference through the wheels is then easily calculated and the angle of rotation is then a ratio. The distance to move is calculated by equation (3.2) and the total number of interrupts can be calculated using equation (3.1). To rotate clockwise, the left motor is driven forward and the right is driven backwards. To rotate anti-clockwise, the directions are reversed.

$$\delta_R = A \times \frac{C_b}{360} \quad (3.2)$$

Combining equations (3.1) and (3.2) gives:

$$\text{Interrupts} = A \times \frac{\gamma}{C_w} \times \frac{C_b}{360} \quad (3.3)$$

Where A is the angle to rotate in degrees, γ is the counts per revolution of the wheel, C_w is the circumference of the wheel and $C_b = 2\pi \times r_b$ and r_b is the distance from the centre of the robot to the centre of the wheel (see figure 3.9).

As the voltage swing on the collector of the phototransistor does not reach near 0V, the AVR cannot detect this as a logical 0. An external amplifier could be used to generate a full swing voltage. However, the AVR has ADCs and analogue comparators.

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 is 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. This method has the advantage of returning a boolean value of if the collector voltage is higher or lower than the threshold voltage. The detection is easier to detect than reading ADC values and the code will be simpler to implement. 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. After the set up is complete, the method *Motors_Execute* is run, containing the state machine to control the motors. The state machine has two states ‘On a Tab’ and ‘Not on a Tab’. When the method enters, the initial state is read from the comparators. 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 3.8. Starting in ‘Not on a Tab’ state, a tab must be detected for Hyst_Max 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 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 motor speed was controlled by PWM. The code sets up a low duty cycle PWM signal to drive the motors slowly. This causes any overshoot that could happen to be low and removes the need for a speed controller.

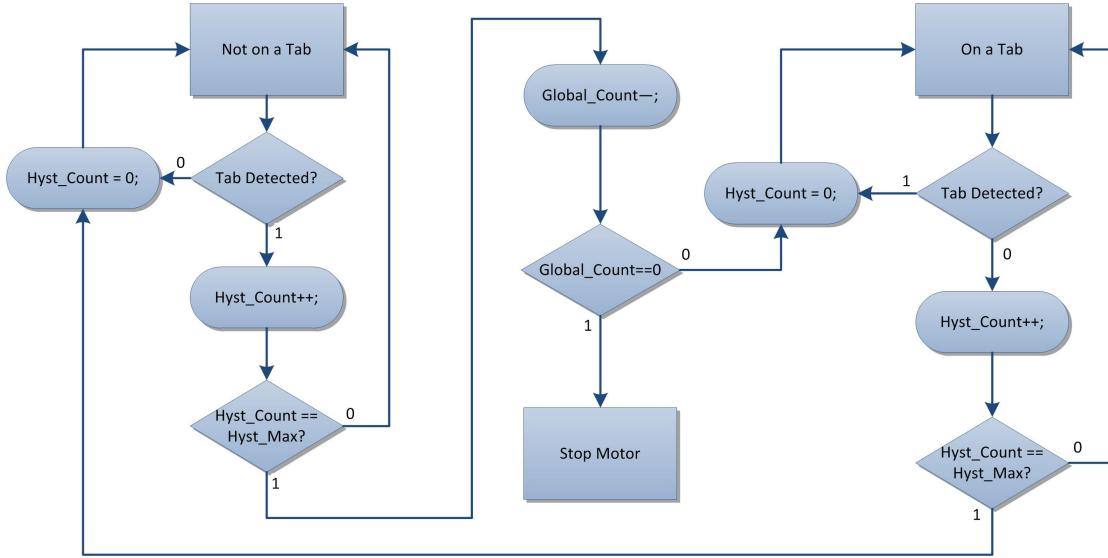


Figure 3.8: A State Machine showing the operation of the *Motor_Execute* method

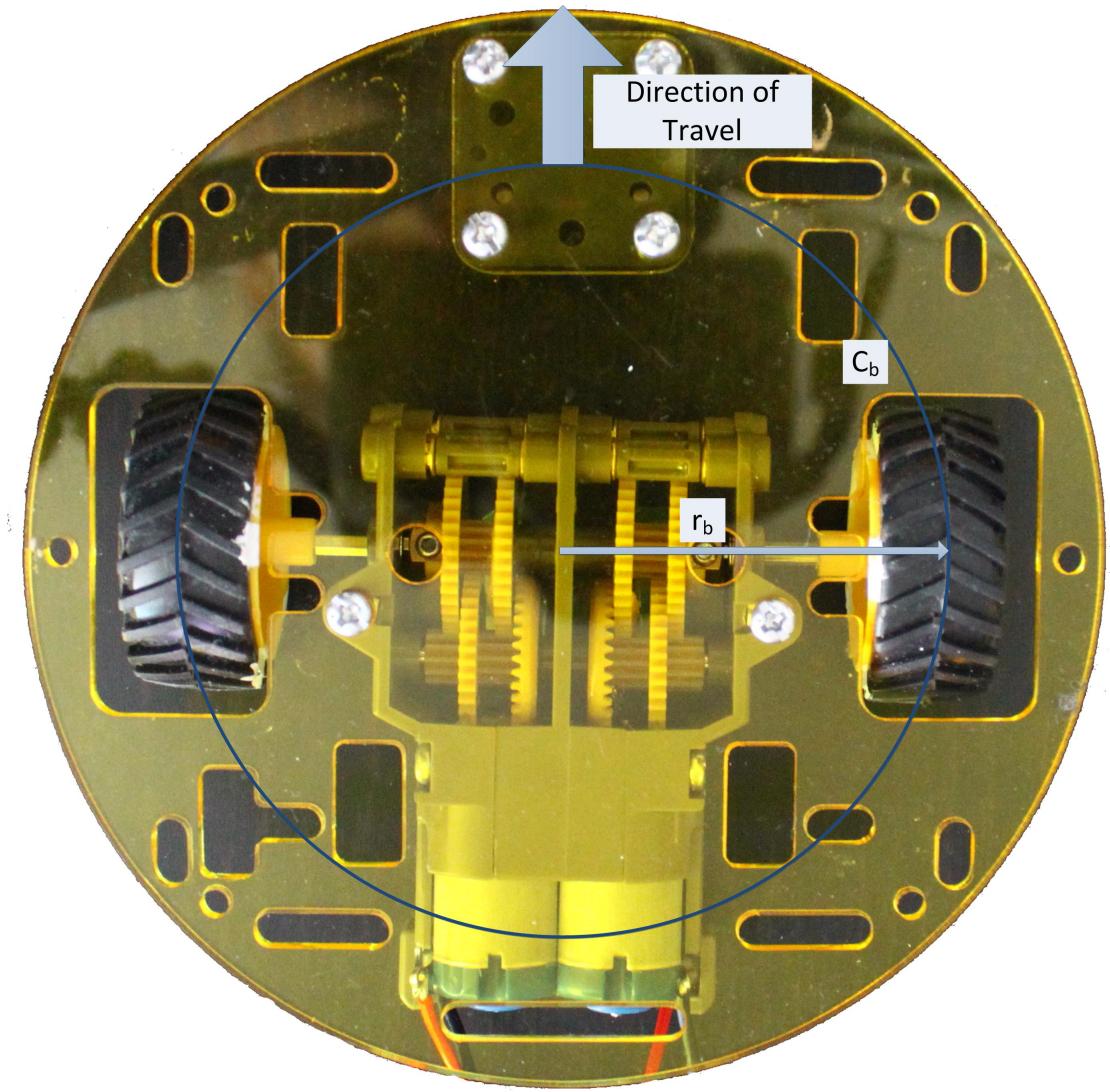
The final code can be seen in appendix H. *Motor_Init* method must be called before operation can occur. This sets up the PWM and analogue comparators. Methods *Motors_Move* and *Motors_Rotate* are the methods that can be called to move in a straight line or rotate on the spot. They both take an input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

3.3.3 Testing

Test Rotation

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 3.3 shows the results of this test and figure 3.10. The results show that the maximum error observed is 4.52%, which related to 4.5mm. This error is acceptable as a half centimetre error over 10 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



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



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

Figure 3.9: Dimensions of Interest for Robot Movement

Table 3.3: Results of Motor Distance Test

Distance Specified (mm)	Number of Counts Returned	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

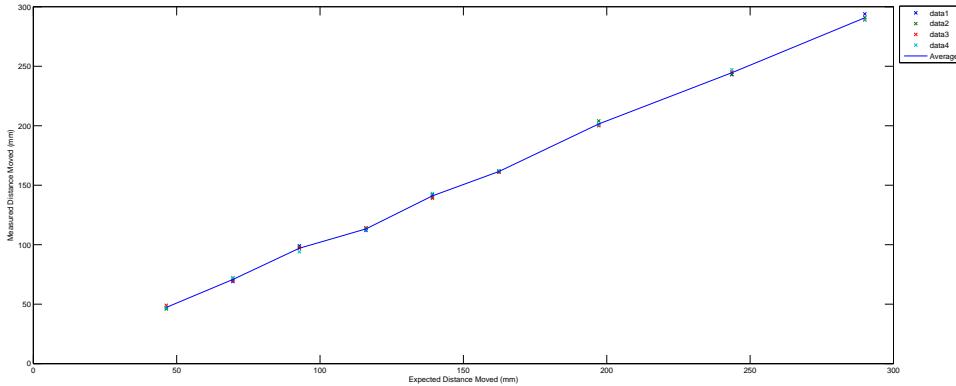


Figure 3.10: A plot of Expected Distance against the measured data. Line indicates the average of the data at each point.

Table 3.4: 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

seen in table 3.4. 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.

3.3.4 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 greater distance resolution can be obtained by decreasing the wheel size or increasing γ and a greater rotational resolution can be obtained by the same as distance, or by increasing the distance the wheels are from the centre of the robot. The ratio $C_w : \gamma$ should be as large as possible to obtain the best resolution for movement.

$$\Delta_\delta = \frac{C_w}{\gamma} = 12mm \quad (3.4)$$

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

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 proportionally 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 than 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.

3.4 PCB Development

3.4.1 Circuit Design

Figure 3.11 shows a basic hierarchy of the robot. Each pin on the AT32UC3C0512C can have one of up to six special functions, as well as being a GPIO pin. Table

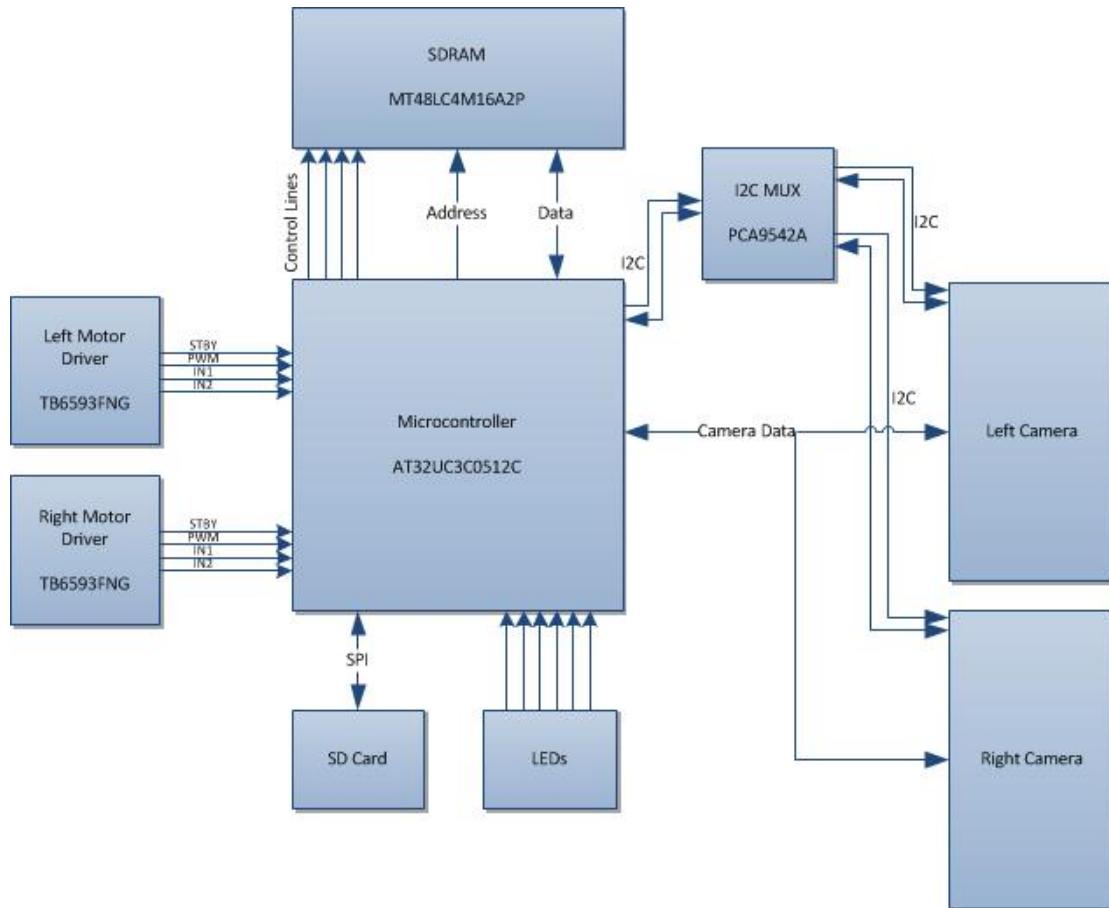


Figure 3.11: A hierarchical diagram of the robot

[3.5](#) 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](#)).

3.4.2 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 3V3 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

Table 3.5: The Pinout of the AVR for the circuit. ‘-’ means unavailable and blank means unused

Pin	Port			
	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	-

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 MUX, SD card, motor drivers etc). As a precaution, spare pins from the UC3C were routed to a header (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 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 for either power or so that the robot can 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 [Cart \(2013\)](#). The PCB cost £205 to manufacture and ship. A photo of the PCB can be seen in figure 3.12.

Considerations - Power consumption of devices not exceeding VReg

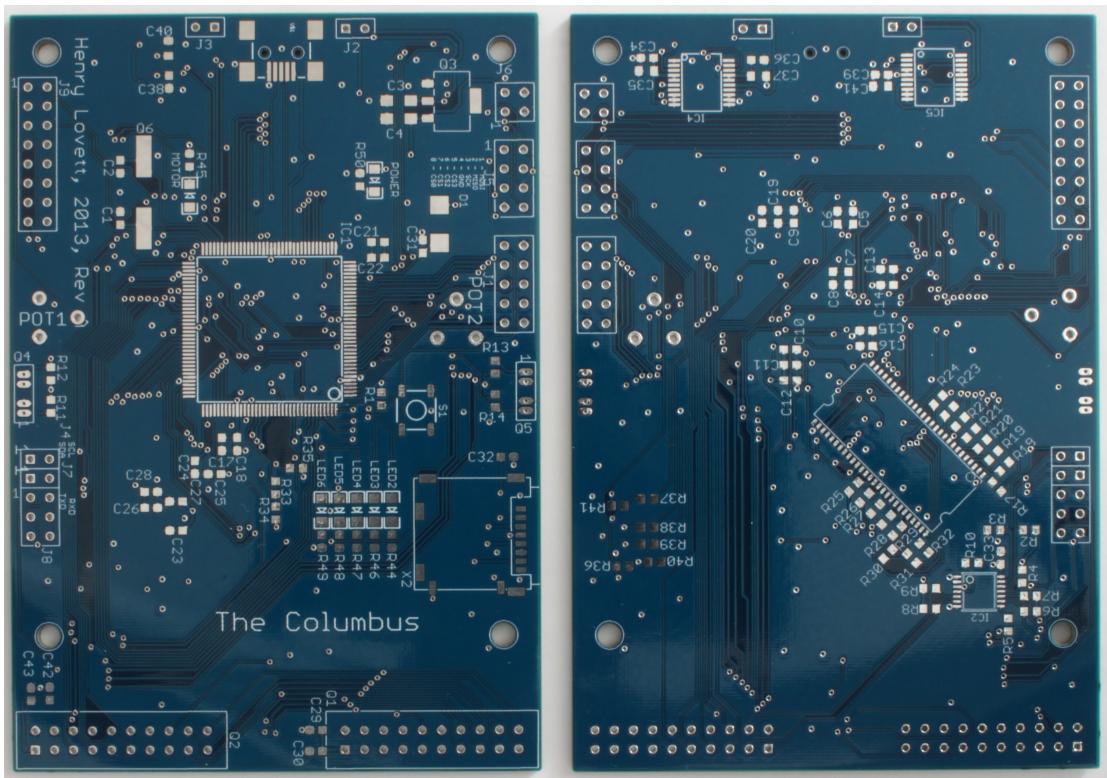


Figure 3.12: PCB with no components. Left: Top View. Right: Bottom View

3.4.3 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:

[3.4.3.1](#) UART Send and Receive

[3.4.3.2](#) SD Card

[3.4.3.3](#) LEDs

[3.4.3.4](#) SDRAM

[3.4.3.5](#) I^2C

[3.4.3.6](#) Camera

[3.4.3.7](#) Motor

3.4.3.1 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

```

1 ///////////////////////////////////////////////////////////////////
2 // UART Test ///////////////////////////////////////////////////////////////////
3 ///////////////////////////////////////////////////////////////////
4 uint8_t ch;
5 while (true) {
6     ch = usart_getchar(DBG_USART); // get one input character
7     if (ch) {
8         print_dbg(ch); // echo to output
9     }
10    if(ch == 13)
11        break;
12 }
```

3.4.3.2 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.

Listing 3.2: UART Test Code

```

1 ///////////////////////////////////////////////////////////////////
2 // SD Card Test ///////////////////////////////////////////////////////////////////
3 ///////////////////////////////////////////////////////////////////
4 print_dbg("\n\n\rSD Card Memory Test:\n\r");
5 // Test if the memory is ready - using the control access memory abstraction
6 // layer (/SERVICES/MEMORY/CTRL_ACCESS/)
7 if (mem_test_unit_ready(LUN_ID_SD_MMC_SPI_MEM) == CTRL_GOOD)
8 {
9     // Get and display the capacity
10    mem_read_capacity(LUN_ID_SD_MMC_SPI_MEM, &VarTemp);
11    print_dbg("OK:\t");
```

```
11     //printf_ulong((VarTemp + 1) >> (20 - FS_SHIFT_B_TO_SECTOR));
12     i = ((VarTemp + 1) >> (20 - FS_SHIFT_B_TO_SECTOR));
13     print_dbg_ulong(i);
14     print_dbg("MB\r\n");
15     print_dbg("SD Card Okay.\n\r");
16 }
17 else
18 {
19     // Display an error message
20     print_dbg("Not initialized: Check if memory is ready...\r\n");
21 }
22 nav_reset();
23 // Use the last drive available as default.
24 nav_drive_set(nav_drive_nb() - 1);
25 // Mount it.
26 nav_partition_mount();
27 nav_filelist_reset();
28 if(nav_filelist_findname((FS_STRING)LOG_FILE, false))
29 {
30     print_dbg("\n\rLog File Already Exists\n\rAttempting to delete...");
31     nav_setcwd((FS_STRING)LOG_FILE, true, false);
32     nav_file_del(false);

34     if(nav_filelist_findname((FS_STRING)LOG_FILE, false))
35         print_dbg("\n\rLog File Still Exists...");
36     else
37         print_dbg("\n\rLog File Deleted!");
38 }
39 print_dbg("\n\rCreating Log File.");

41 if(nav_file_create((FS_STRING)LOG_FILE) == true)
42     print_dbg("\n\rSuccess!");
43 else
44     print_dbg("\n\rNot worked...");
45 print_dbg("\n\rWriting to log file.");
46 Log_Write("Columbus Tester:\n\r", 18);
47 nav_filelist_reset();
48 nav_setcwd((FS_STRING)LOG_FILE, true, false);
49 file_open(FOPEN_MODE_R); //Open File
50 file_read_buf(Buffer, 8);
51 noErrors = 0;
52 if(Buffer[0] != 'C')
53     noErrors++;
54 if(Buffer[1] != 'o')
55     noErrors++;
56 if(Buffer[2] != 'l')
57     noErrors++;
58 if(Buffer[3] != 'u')
59     noErrors++;
60 if(Buffer[4] != 'm')
61     noErrors++;
62 if(Buffer[5] != 'b')
63     noErrors++;
64 if(Buffer[6] != 'u')
65     noErrors++;
66 if(Buffer[7] != 's')
67     noErrors++;
68 file_close();
```

```

69     if(noErrors == 0)
70         print_dbg("\n\rSD Card Read Successful\n\r");
71     else
72         print_dbg("\n\rSD Card Read Fail\n\r");
73     noErrors = 0;

```

This exercises all basic File I/O functions, creating, reading and writing and checks them on the device.

3.4.3.3 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.

3.4.3.4 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

```

1 ///////////////////////////////////////////////////////////////////
2 // SDRAM Test ///////////////////////////////////////////////////////////////////
3 ///////////////////////////////////////////////////////////////////
4 print_dbg("\n\rSDRAM Test:");
5 sdram_size = SDRAM_SIZE >> 2;
6 print_dbg("\n\rSDRAM size: ");
7 print_dbg_ulong(SDRAM_SIZE >> 20);
8 print_dbg(" MB\r\n");
9 // Determine the increment of SDRAM word address requiring an update of the
10 // printed progression status.
11 progress_inc = (sdram_size + 50) / 100;
12 // Fill the SDRAM with the test pattern.
13 for (i = 0, j = 0; i < sdram_size; i++)
14 {
15     if (i == j * progress_inc)
16     {

```

```

17     print_dbg("\rFilling SDRAM with test pattern:");
18     print_dbg_ulong(j++);
19     print_dbg("%");
20 }
21 sdram[i] = i;

23 }
24 print_dbg("\rSDRAM filled with test pattern      \r\n");
// Recover the test pattern from the SDRAM and verify it.
25 for (i = 0, j = 0; i < sdram_size; i++)
26 {
27     if (i == j * progress_inc)
28     {
29         print_dbg("\rRecovering test pattern from SDRAM: ");
30         print_dbg_ulong(j++);
31         print_dbg("%");
32     }
33     tmp = sdram[i];
34     if (tmp != i)//failed
35     {
36         noErrors++;
37     }
38 }
39 print_dbg("\rSDRAM tested: ");
40 print_dbg_ulong(noErrors);
41 print_dbg(" corrupted word(s)      \r\n");

```

3.4.3.5 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 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 MUX, 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 MUX. The camera should only acknowledge when the I^2C MUX 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

```

1 ///////////////////////////////////////////////////////////////////
2 // TWI Test ///////////////////////////////////////////////////////////////////
3 ///////////////////////////////////////////////////////////////////
4 print_dbg("\n\n\rTWI Test:\n\r");
5 Log_Write("\n\n\rTWI Test:\n\r", 14);
6 for(k = 0; tkmp < 3; k++)
7 {
8     if(k == 0){

```

```

9      print_dbg("Scanning all Channels\n\r");
10     }
11
12     else if (k == 1){
13         //Channel 0
14         PCA9542A_Chан_Sel(I2C_CHANNEL_0);
15         print_dbg("\n\rScanning Channel 0\n\r");
16     }
17
18     else {
19         //Channel 1
20         PCA9542A_Chан_Sel(I2C_CHANNEL_1);
21         print_dbg("\n\rScanning Channel 1\n\r");
22     }
23
24
25     print_dbg("h 0 1 2 3 4 5 6 7 8 9 A B C D E F\n\r");
26     tmp = 0;
27     for(i = 0; i < 8; i++)
28     {
29         print_dbg_ulong(i);
30         print_dbg(" ");
31         for(j = 0; j < 16; j++){
32             int status = twim_probe(TWIM, tmp++);
33             if(status == STATUS_OK){
34                 print_dbg("A");
35             }
36             else{
37                 print_dbg("-");
38             }
39             print_dbg(" ");
40         }
41         print_dbg("\n\r");
42     }
43
44     noErrors = 0;
45
46     //Check cameras exist
47     PCA9542A_Chан_Sel(I2C_CHANNEL_0);
48     if(twim_probe(TWIM, 0x21) != STATUS_OK)
49         print_dbg("\n\rCamera 0 Not Found;");
50     PCA9542A_Chан_Sel(I2C_CHANNEL_1);
51     if(twim_probe(TWIM, 0x21) != STATUS_OK)
52         print_dbg("\n\rCamera 1 Not Found;");
```

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

3.4.3.6 Camera Test

This test consists of initialising both cameras and checking it passes. 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 exists on the SD card from the two cameras. Listing 3.6 shows the code to conduct this test.

Listing 3.6: Camera Test Code

```

1  ///////////////////////////////////////////////////////////////////
2  // Camera Test ///////////////////////////////////////////////////////////////////
3  ///////////////////////////////////////////////////////////////////
4  print_dbg("\n\rInitialising Cameras");
5  OV7670_Init();
6  FIFO_Reset(CAMERA_LEFT | CAMERA_RIGHT);
7  if(STATUS_OK == OV7670_Status.Error)
8  {
9    print_dbg("\n\rCamera Initialise Okay!");
10 }
11 else
12   print_dbg("\n\rCamara Initialise Fail.");
13
14 print_dbg("\n\rTaking Photos");
15
16 TakePhoto(CAMERA_LEFT | CAMERA_RIGHT);
17 while(Photos_Ready() == false)
18   ;
19
20 if(Store_Both_Images() == true)
21   print_dbg("\n\rImages Stored Successfully!");
22 else
23   print_dbg("\n\rImages Store Fail.");

```

3.4.3.7 Motor Driver Test

An extensive test of the motor driver is discussed in section 3.3.3. 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

```

1  ///////////////////////////////////////////////////////////////////
2  // Motor Test ///////////////////////////////////////////////////////////////////
3  ///////////////////////////////////////////////////////////////////
4  print_dbg("\n\rMotor Testing:\n\rMotor Initialised");

```

```
5     Motor_Init();
6     Motors_Reset(); //reset the motors to test them
7     while(Motors_Moving() == true)
8         ;//wait for the motors to finish moving
```

3.4.4 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.

3.4.4.1 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.

3.4.4.2 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 than 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.

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.

3.4.4.3 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

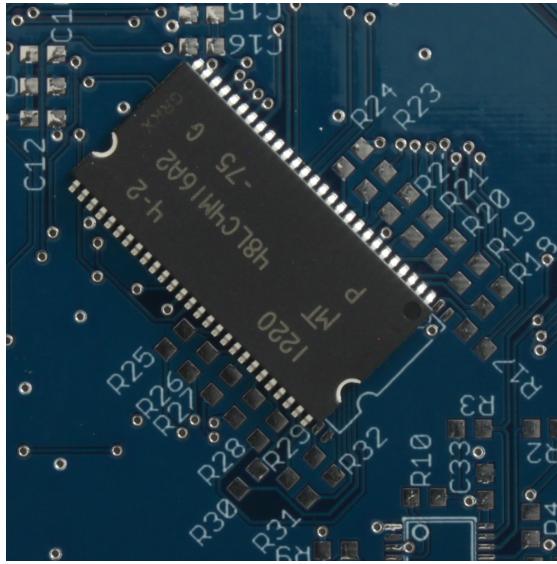


Figure 3.13: SDRAM Chip shown against its footprint.

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.

3.4.4.4 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. This means read cycles are done per word read. 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 3.6 shows some examples of the problematic data bits. The data written should match the data read back.

Table 3.6: 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.

3.4.4.5 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 UC3C0512C 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 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).

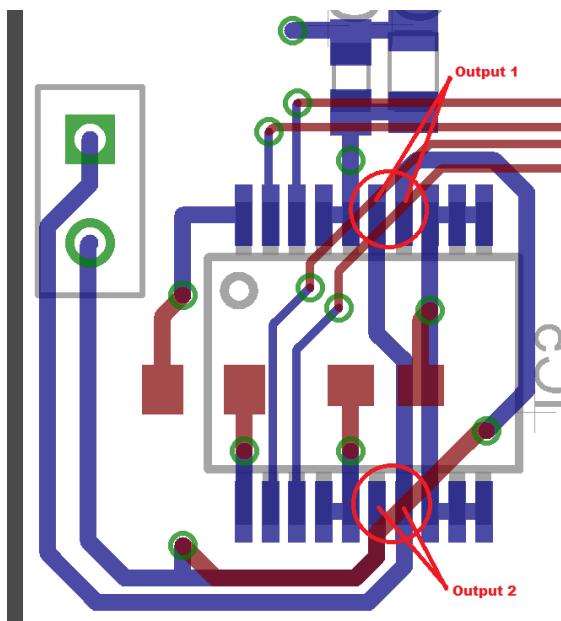


Figure 3.14: Motor Driver error. Outputs incorrectly connected

3.4.4.6 Motor Driver Pinout

An error was made in creating the device for the TB6593FNG Motor Driver in EAGLE. On the device, 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 this. 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.

3.4.5 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 pinout

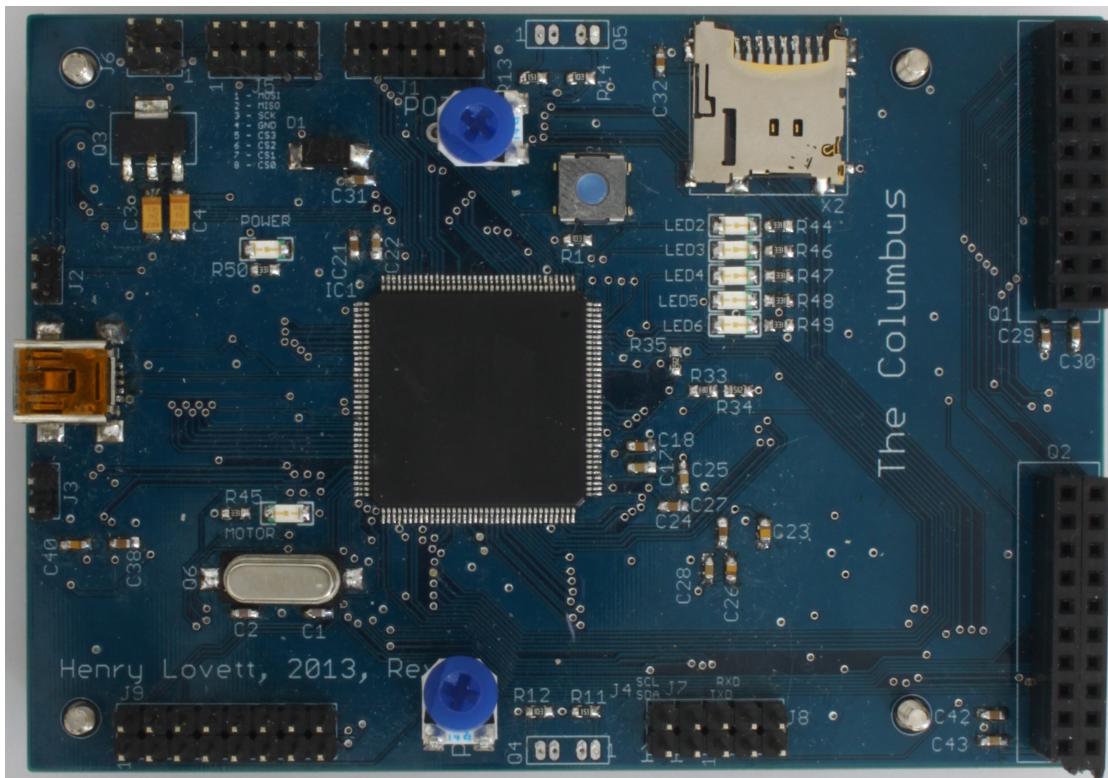
Three of the faults could have been avoided by consulting the datasheet more carefully during the circuit design stage. The footprint error was due to not being experienced in designing footprints and the data line resistors was a mistake due to lack of attention being paid.

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.

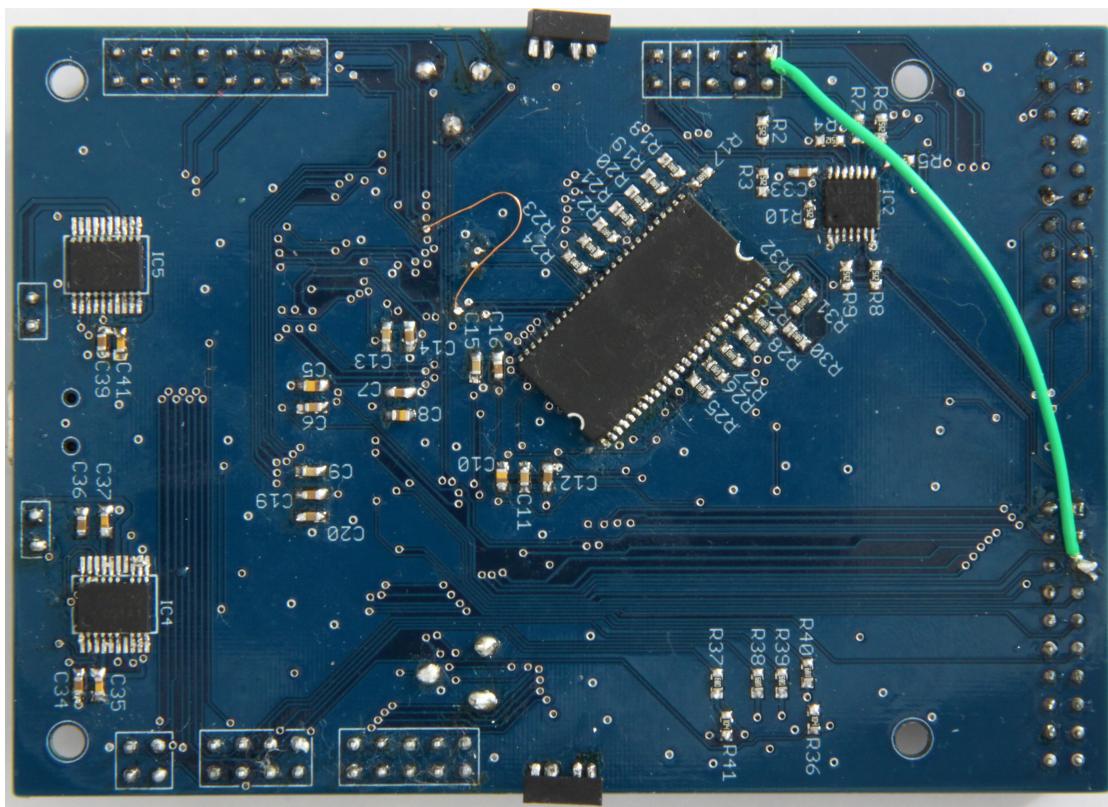
The PCB itself, was a success. It was a complex PCB with many potential things that could have gone wrong. All devices are functional (with a few small modifications) on the PCB so firmware development could continue with all hardware able to be used.

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



(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.

Chapter 4

Investigation into 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 window (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

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

This method subtracts the observed window 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

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

This is very similar to S.A.D. 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 the other with SAD). Again, a low result is a match in this case.

test effect of box size?

4.1.3 NCC

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

Where n is the number of pixels in A and B ,
 σ is the standard deviation of the image, and
 \bar{A} is the 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 S.S.D. and S.A.D., 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 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 operate slowly on a microcontroller without any.

4.1.4 Comparison

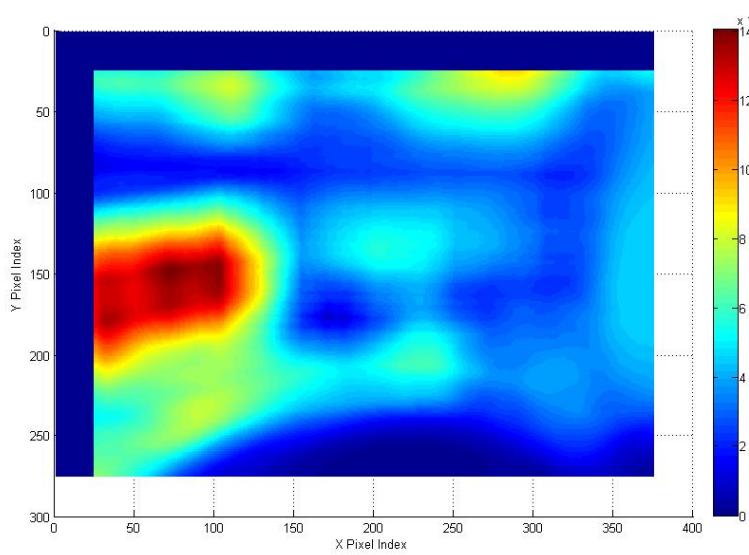
To compare these equations, a 50×50 template taken from the right picture was compared with the left image over the entire valid range. The coordinates on the graph give the centre pixel of the calculation.

Each graph shows the correct area being identified as a match, but this also highlights the downfalls of the SAD and SSD. The figures 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](#)).

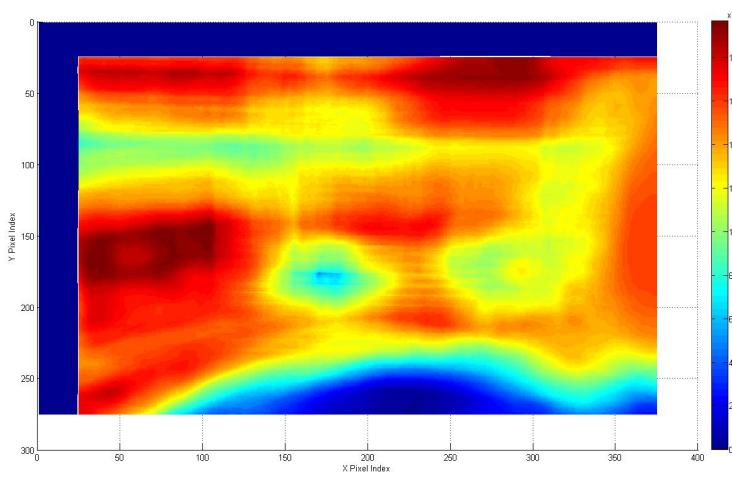
SAD results in figure 4.2(a) show large areas of matching. A minimum occurs around the location expected(170, 175) of a value of 5.66×10^4 . However, along the bottom of the image, where a dark area occurs below the desk in the lower part of figures 4.1, the SAD algorithm detects a greater comparison, 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.

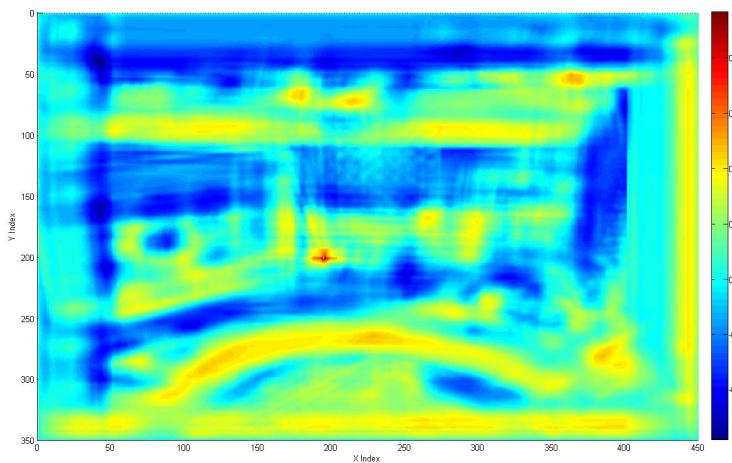
The NCC results are visible in figure 4.2(c). A match can be seen at coordinate (195, 201) with a peak value of 0.9654. The coordinate is different to the previous results because the cross correlation works over the boundary of the image creating more results. The dimensions of the image are 300×400 , but the NCC returns



(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

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 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 a large area of research, 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:

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

Object is to the same side in both images:

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

Object is directly in front of a camera:

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

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

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

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

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 then found 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 $\varphi_0 = 0.6249^\circ$.

Figure to show this?

4.2.4 Testing

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

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

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

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

select the template when prompted.

A stereo pair of images 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 $B = 42\text{mm}$, $\varphi_0 = 0.6249$ 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.

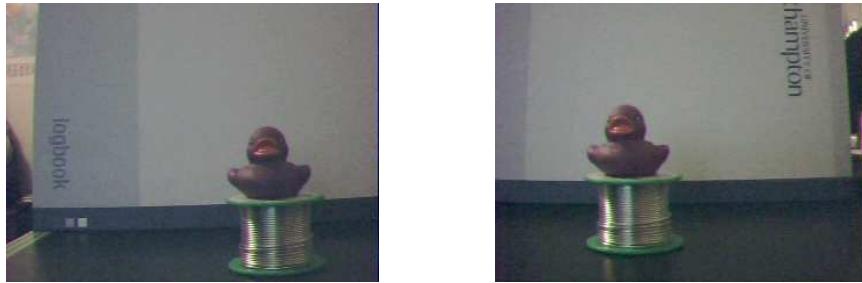


Figure 4.3: Stereo pair of images of a rubber duck on a reel of solder

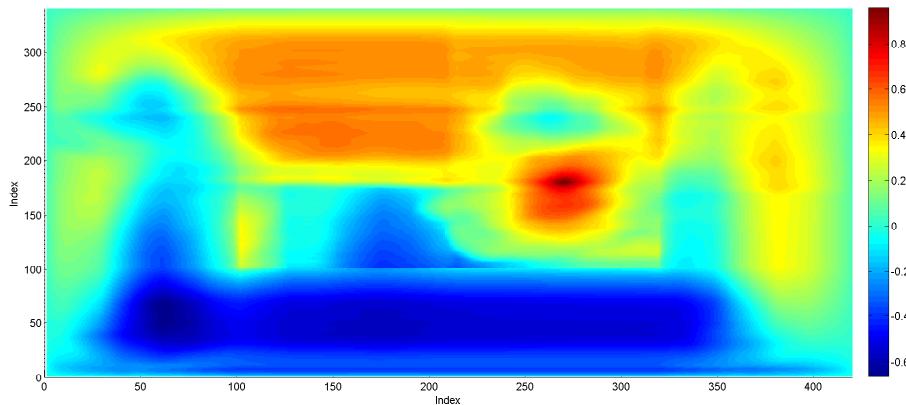


Figure 4.4: NCC results from matching using the duck's head from the right image as the template to the left image

4.2.5 Conclusions

Extensive testing of the effect of separation of the cameras has been done before ([Mrovlje and Vrancić, 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 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 $D = \frac{k}{x}$. This gives a range of distances that the can be detected. For $B = 42\text{mm}$, 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

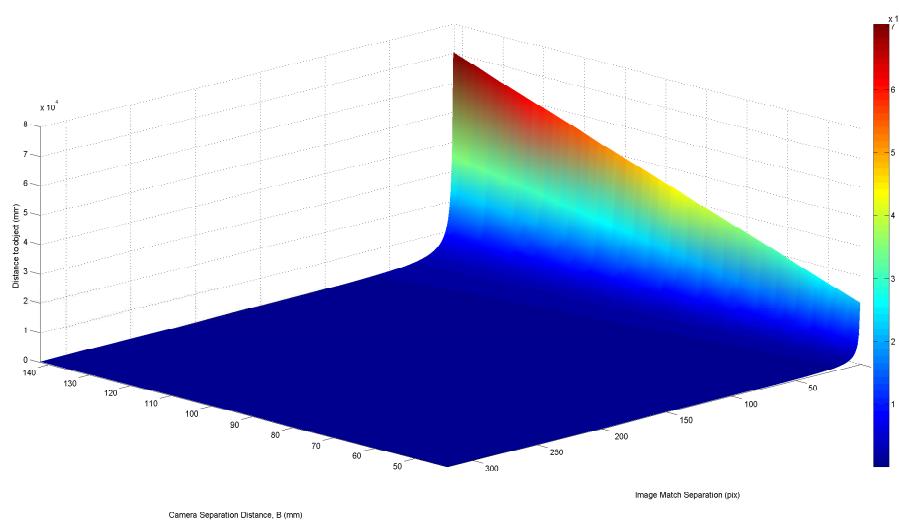


Figure 4.5: 3D Surface plot graph showing range of distances that can be calculated over a range of camera separation, B , values.

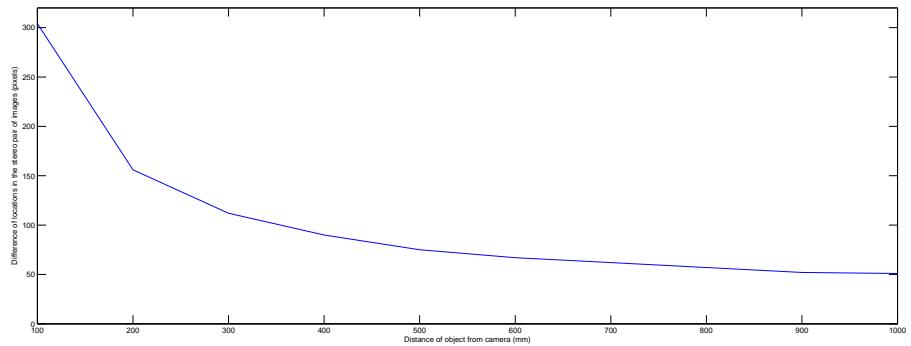


Figure 4.6: Graph showing Distance of the object against the difference in the match locations.

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.

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.

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft} dt \quad (4.9)$$

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

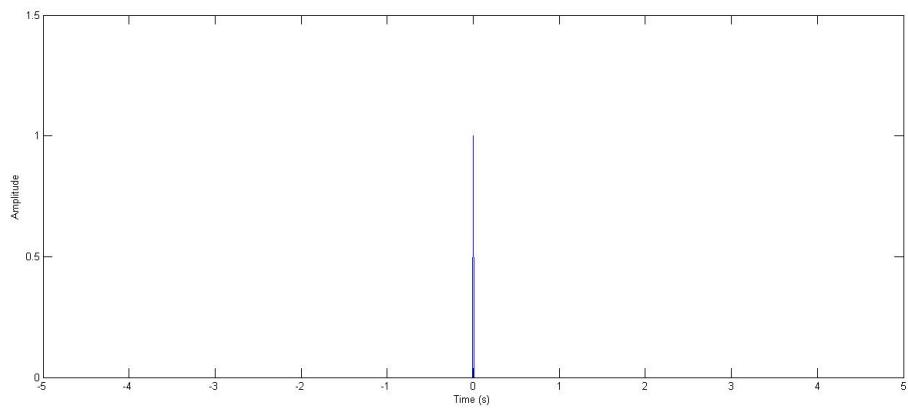
$$X[k] = \sum_0^{N-1} x[n]e^{-j\Omega_0 kn} \quad (4.10)$$

Where Ω_0 is the sample frequency

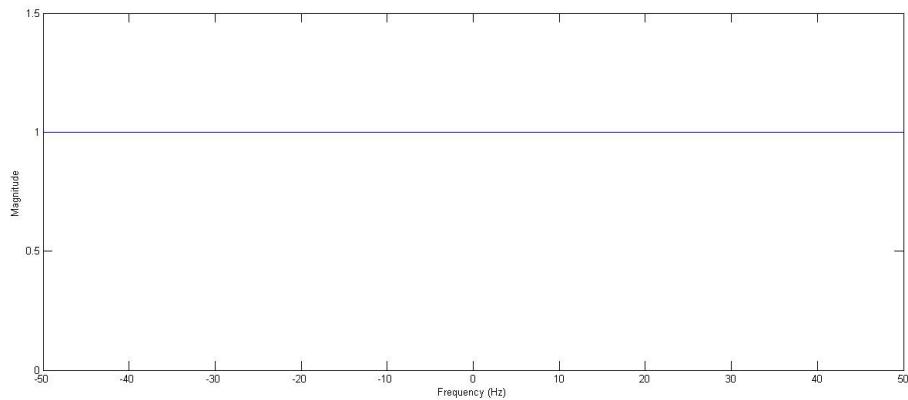
A property of the Fourier Transform of interest is the convolution theorem which states that convolution in time is multiplication in frequency and is defined mathematically in equation (4.11). As discussed in section 4.1.3, cross correlation is very similar to convolution. Cross correlation is defined in equation (4.12). 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 template can be used and multiplied together. FFTs are quick and widely available so this should make the implementation more simple and fast.

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

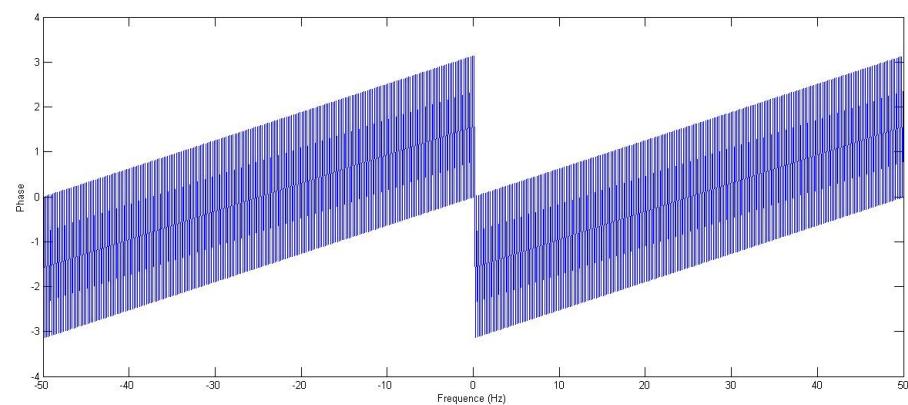
$$\int_{-\infty}^{\infty} f^*(\tau)g(t + \tau)d\tau = f(t) \star g(t) = X(f) \cdot Y(f) \quad (4.12)$$



(a) A graph showing a Dirac Function

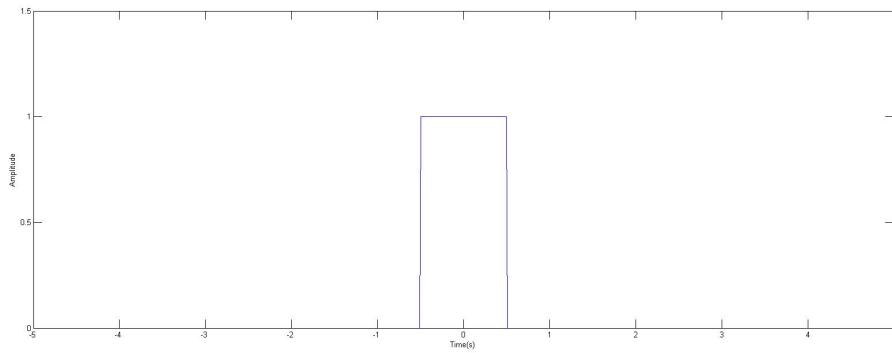


(b) A graph showing the magnitude of the Fourier transform of the Dirac Function

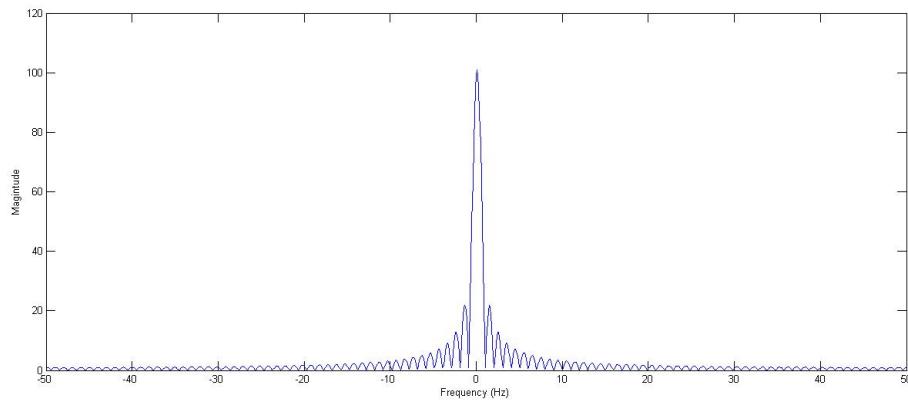


(c) A graph showing the phase of the Fourier transform of the Dirac Function

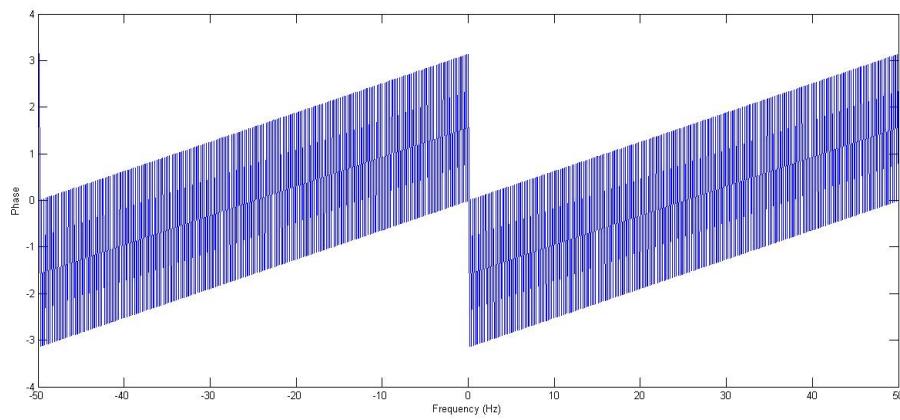
Figure 4.7: A Dirac signal and the phase and magnitude of its Fourier Transform



(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

$$f(t) \star g(t) = f(t) * g(-t) = F(f) \cdot G(-f) \quad (4.13)$$

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 (4.14) and the discrete version is shown in (4.15)

$$F(u, v) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-2\pi j(xu+yv)} dx dy \quad (4.14)$$

$$F(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) e^{-\frac{2\pi j(xu+yv)}{N}} \quad x, y, u, v \in \{0 \dots N-1\} \quad (4.15)$$

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)).

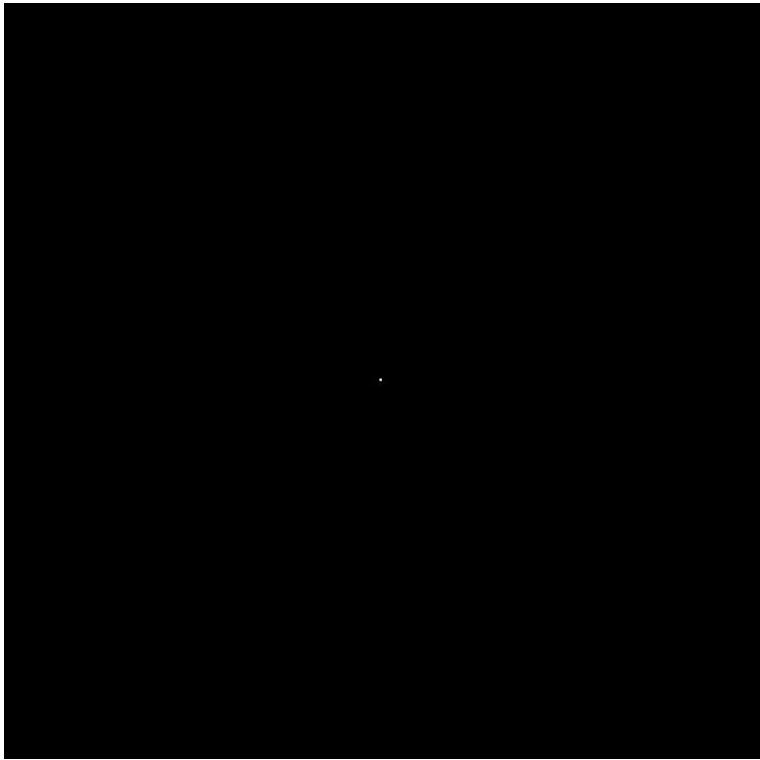
The 2D Fourier transform can also be optimised to a fast Fourier transform algorithm in a similar way as the 1D case. The algorithm, sometimes referred to as the Butterfly transform, is briefly discussed in Nixon and Aguado (2012) where it is explained that the algorithm can be easily applied to images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

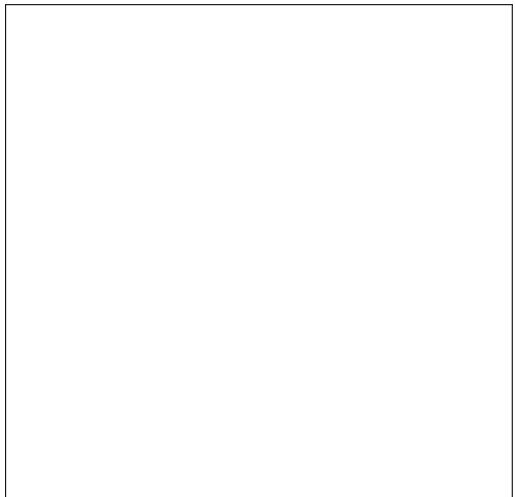
1. Calculate the 1D FFT of each of the rows of the 2D data. (An FFT of data of length n returns an array, also of length n)
2. Calculate the 1D FFT of each of the columns of the 2D data returned from the previous step.

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

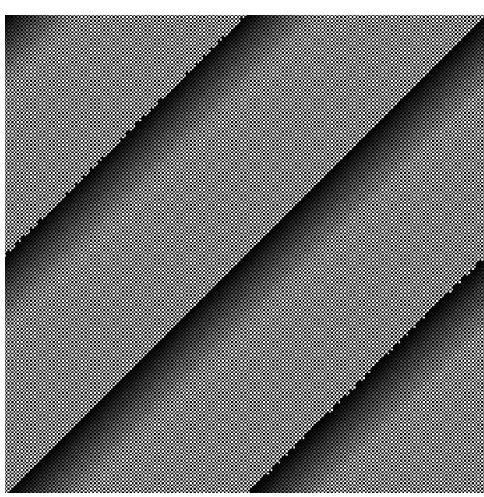
Maybe make a figure to help explain?



(a) An image of a 2D Dirac Function

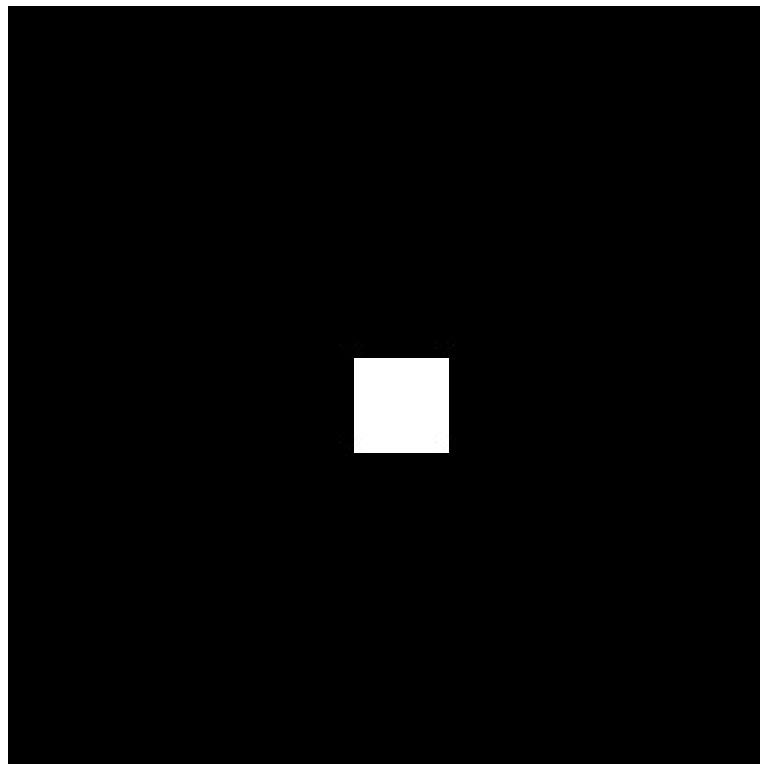


(b) An image of the magnitude of the Fourier transform of the 2D Dirac Function

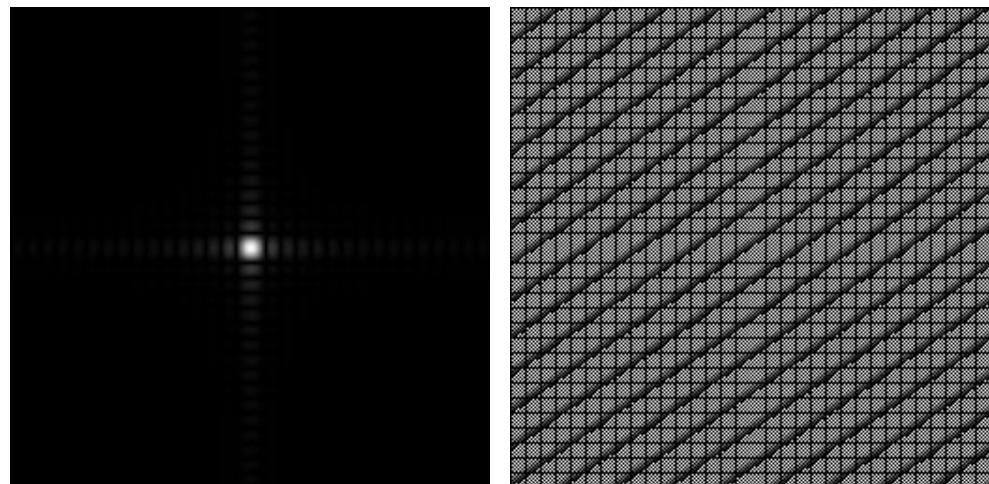


(c) An image of the phase of the Fourier transform of the 2D Dirac Function

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

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, magnitude and phase 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 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 seem not to match. 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 [4.13](#) is obtained. This, though not exactly 1 as expected, is completely flat. It is computed from the same transformed data. It suggests that

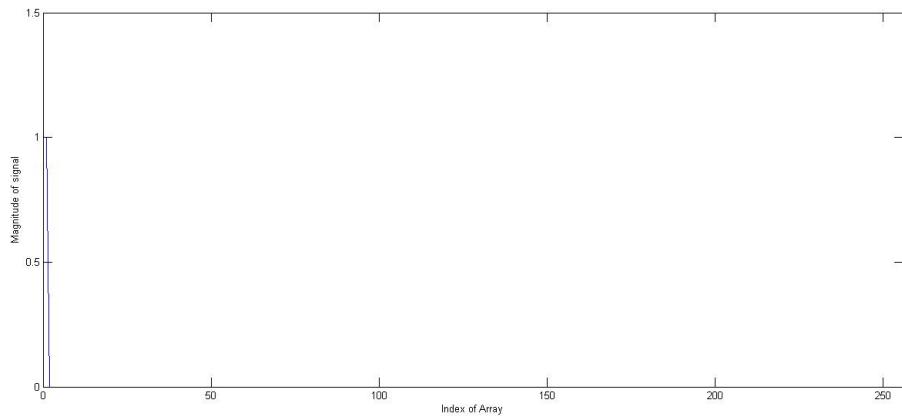


Figure 4.11: Input Dirac Signal for AVR fast Fourier transform

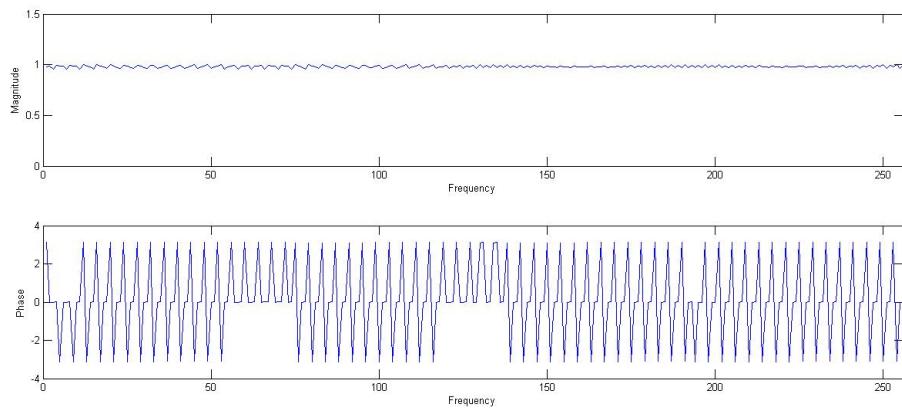


Figure 4.12: Output phase and magnitude of the complex output from AVR fast Fourier transform of a Dirac function

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 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.

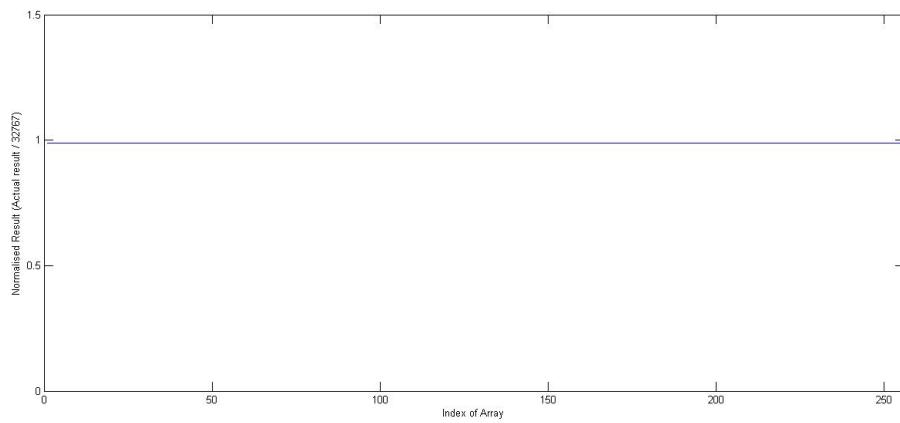


Figure 4.13: Magnitude calculated by the AVR of the Fourier transform of a Dirac function

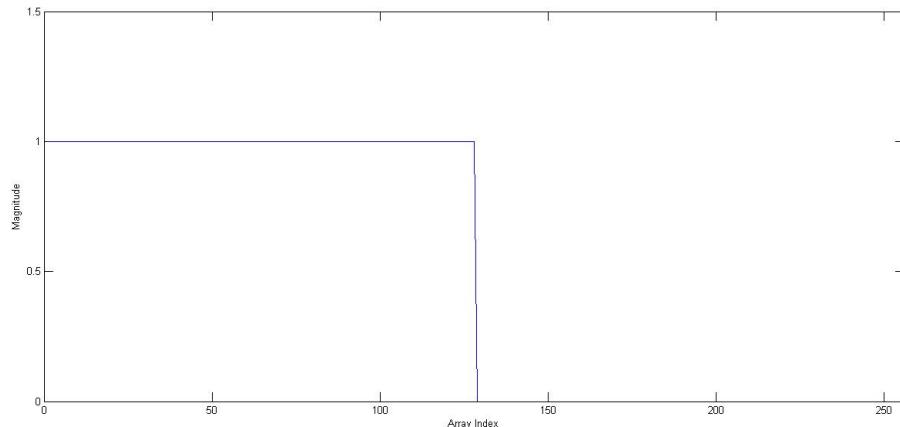


Figure 4.14: Input Rectangular Pulse for AVR fast Fourier transform

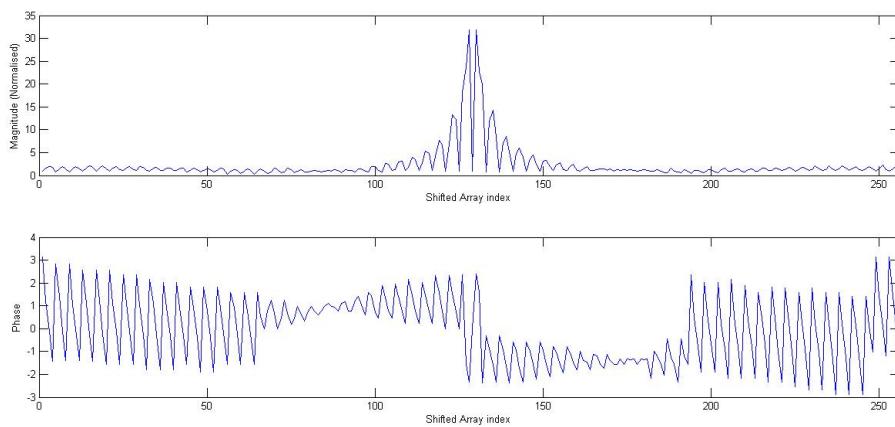


Figure 4.15: Output phase and magnitude of the complex output from AVR fast Fourier transform of a Rectangular Pulse

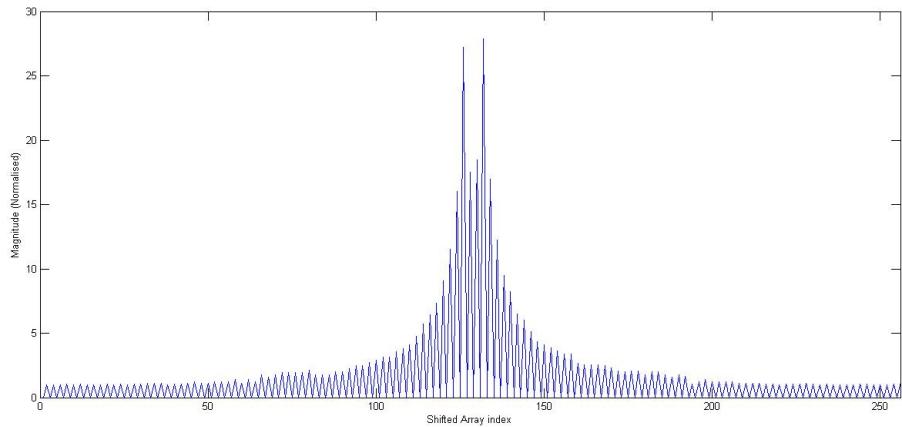


Figure 4.16: Magnitude calculated by the AVR of the Fourier transform of a Rectangular Pulse

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 scaling and the data shifted so that frequency 0 is in the centre. These test 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.

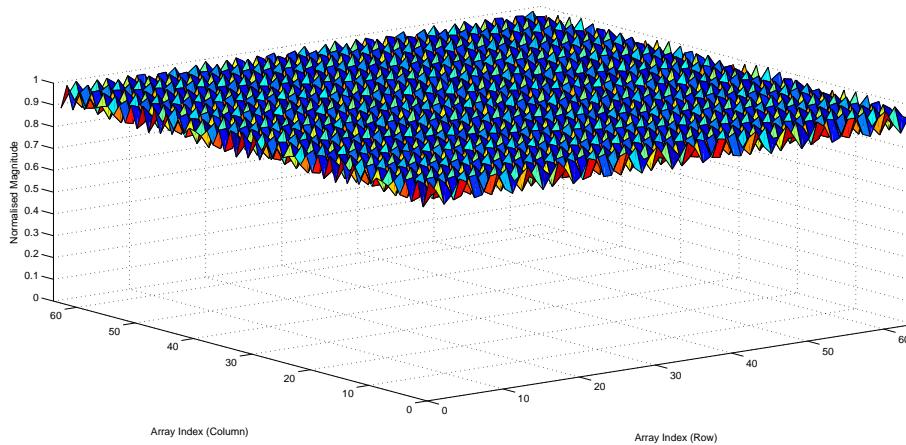
I would like to get this to work by the end but not vital.

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).

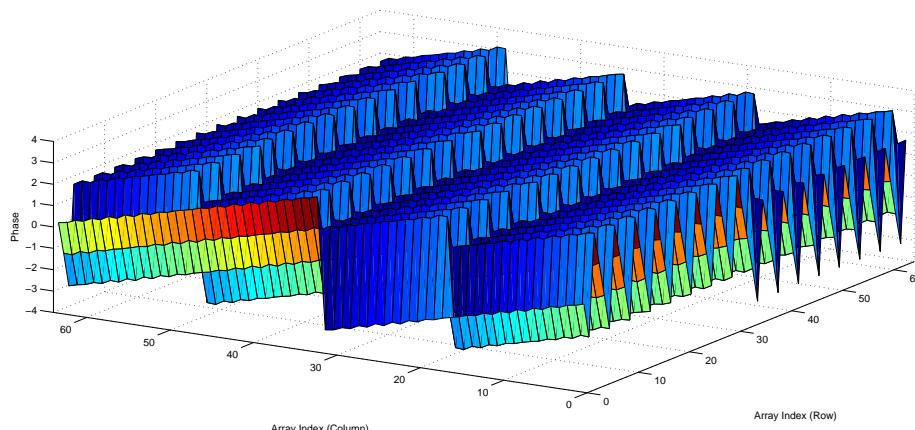
The transforms are calculated in real time with a 16MHz clock source. Table 4.3 shows the number of clock cycles taken to calculate the relevant sized transform. A 64×64 transform takes 39ms to compute with a 16MHz clock. This could

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

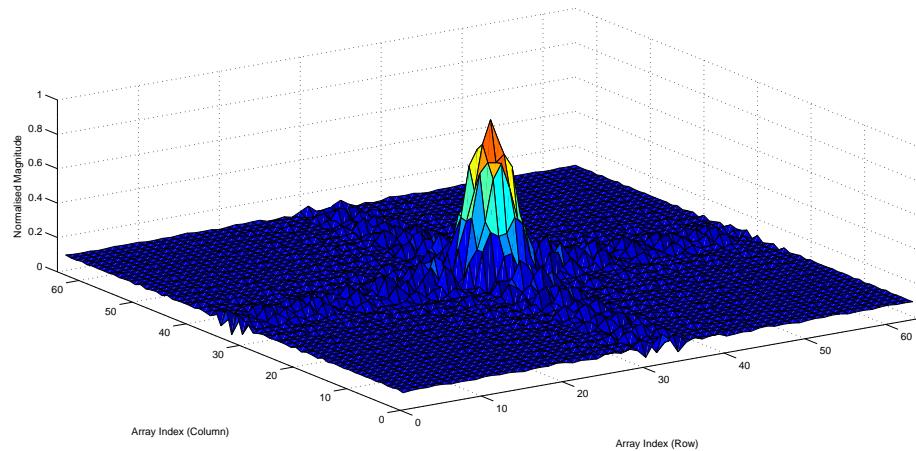


(b) A 3D Plot of the phase of the complex data returned by the 2D fast Fourier Transform on the AVR

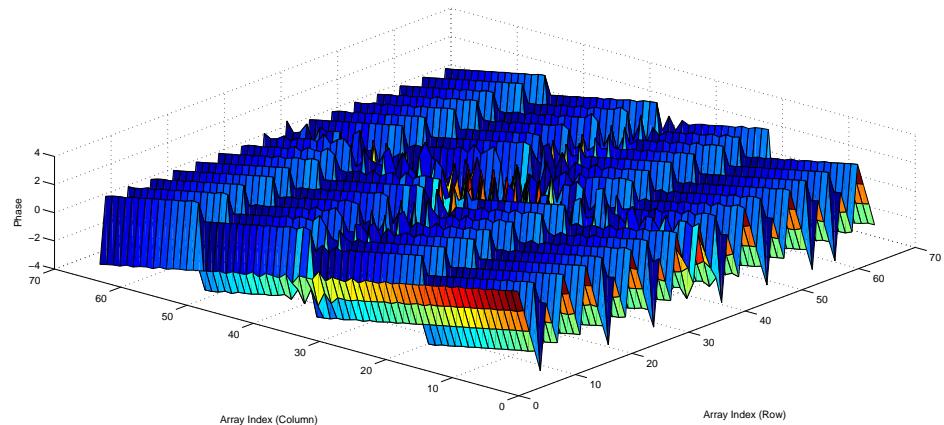
Figure 4.17: 3D Plots of the phase and magnitude of the Complex Data returned from the 2D FFT on the AVR of a 2D Dirac Function

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 and

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



(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

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. SRAM
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 full image can be read from a camera 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 SRAM 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.

The Fourier transform was also investigated and implemented. The system allows for a 2×2 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.

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.

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 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 in the Auto Run procedure (see table 5.1) or by connecting Pin D23, available at

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)

Pin 1 of J9, to ground on system boot. 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.

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

MOTOR	LED					Meaning
	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

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

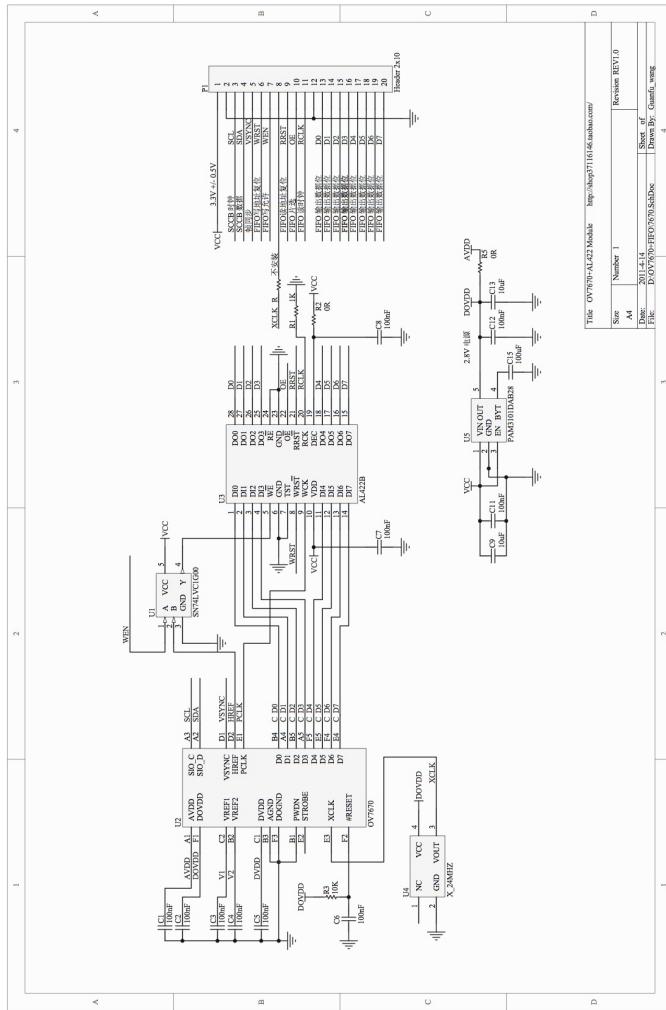


Figure B.1: The circuit diagram for the OV7670 breakout board

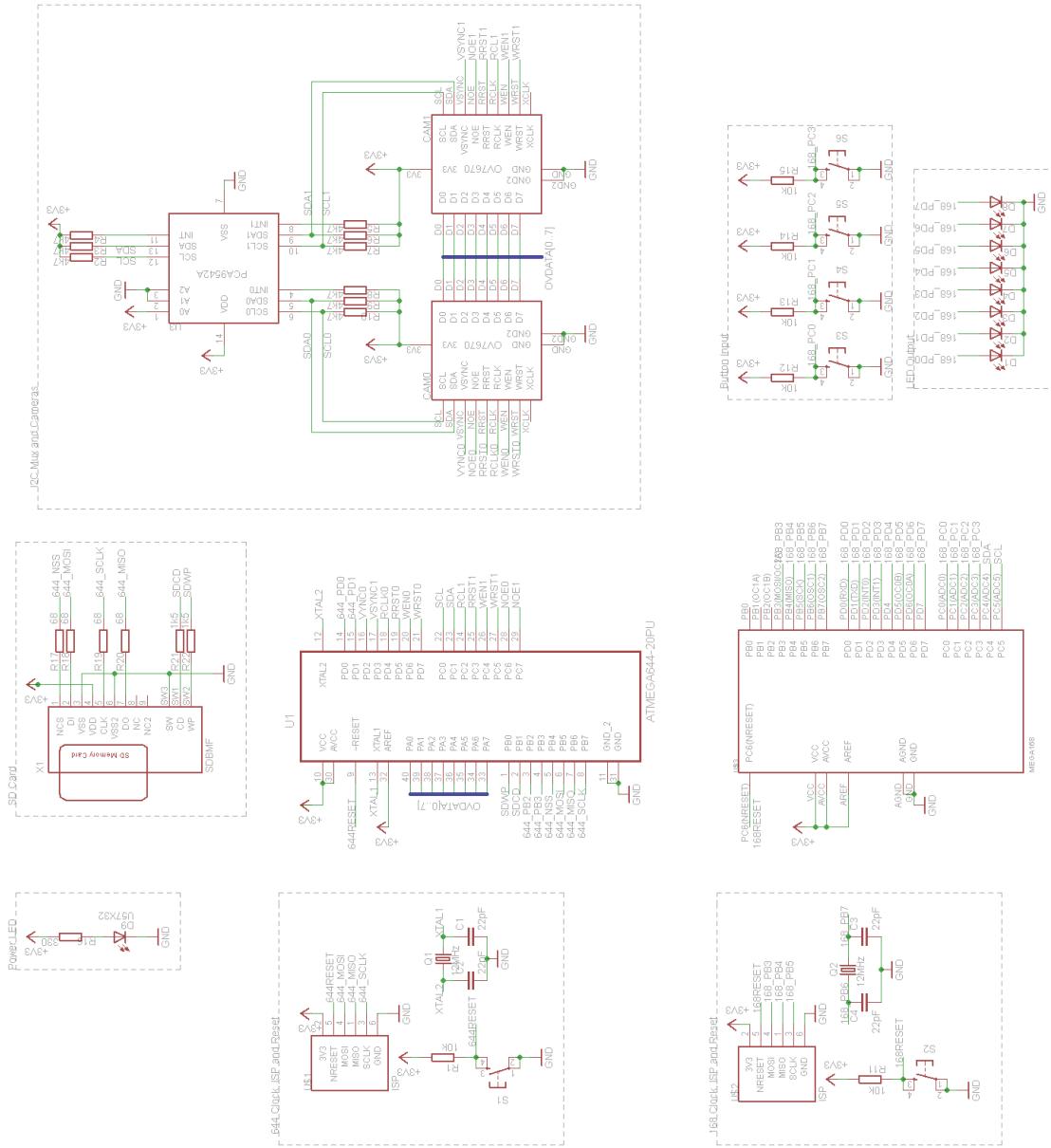


Figure B.2: The circuit diagram for Dual Cameras using the Il Matto Board

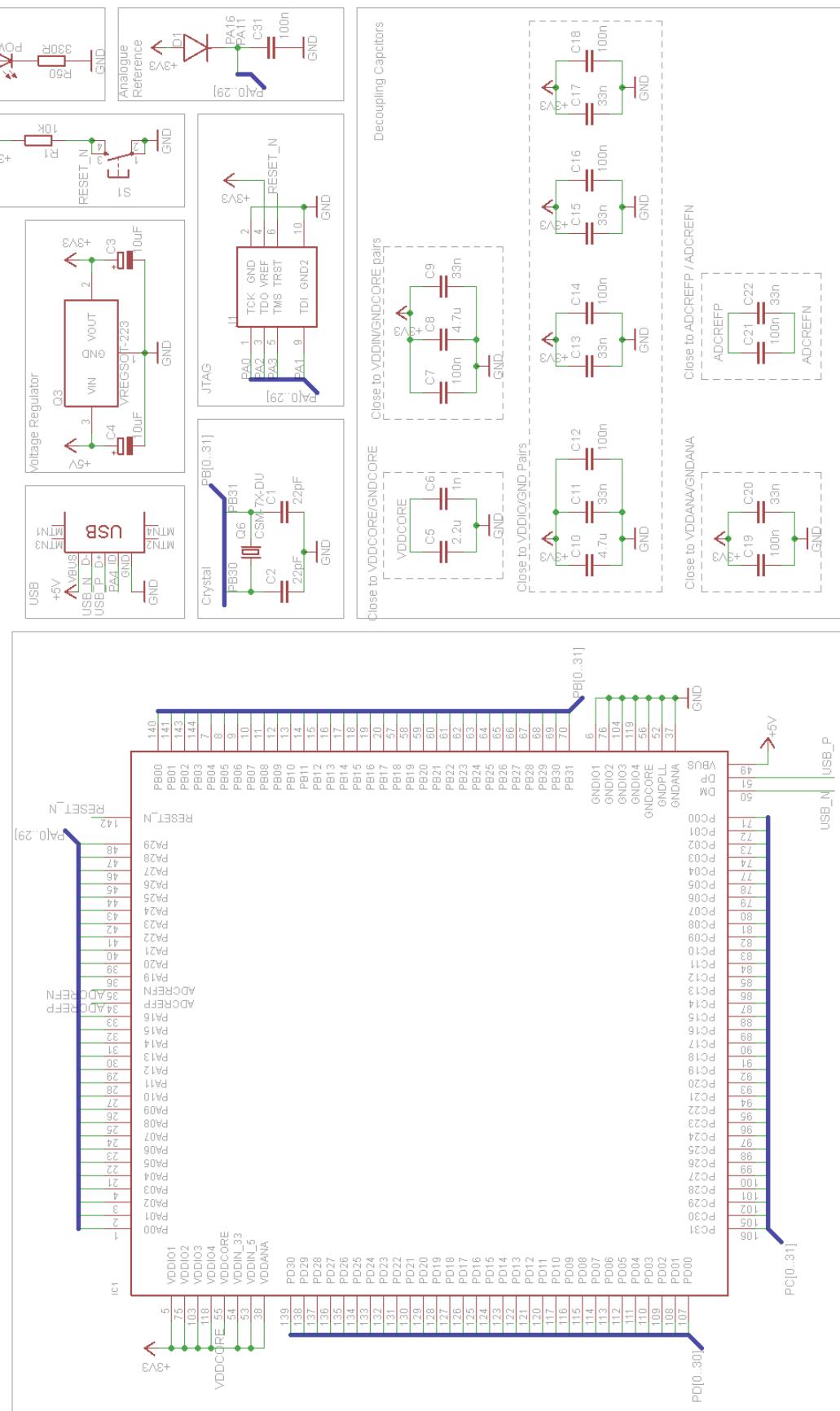


Figure B.3: The Columbus Circuit Diagram Page 1

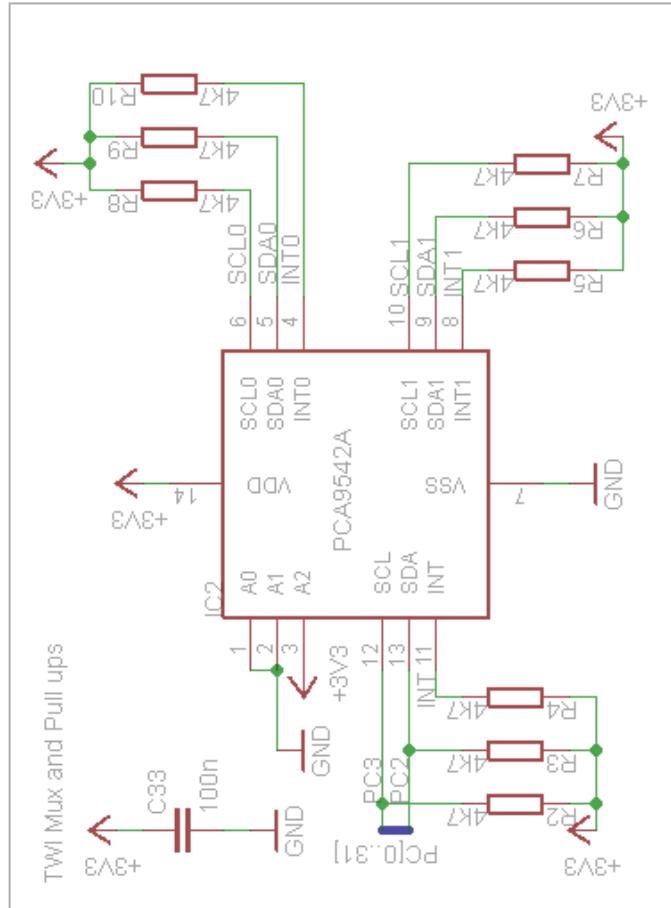
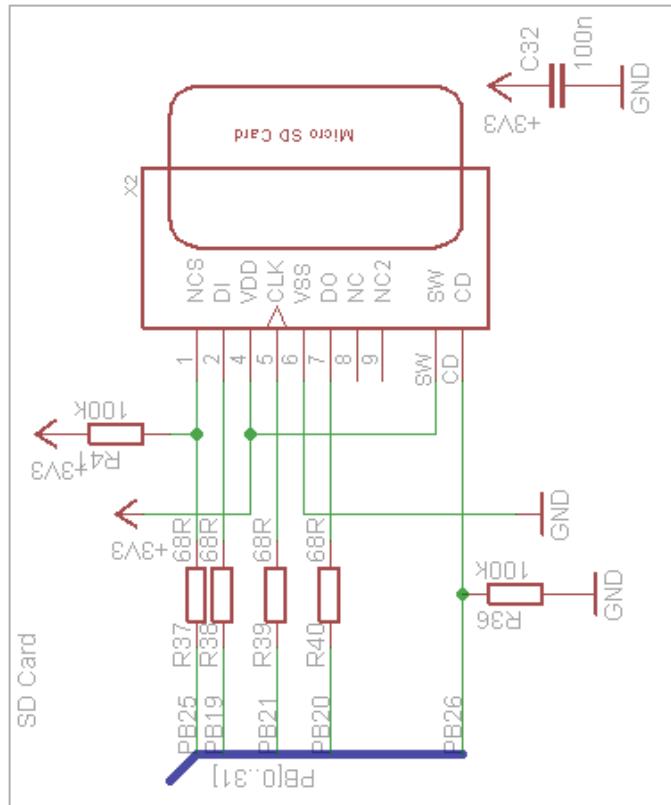


Figure B.4: The Columbus Circuit Diagram Page 2

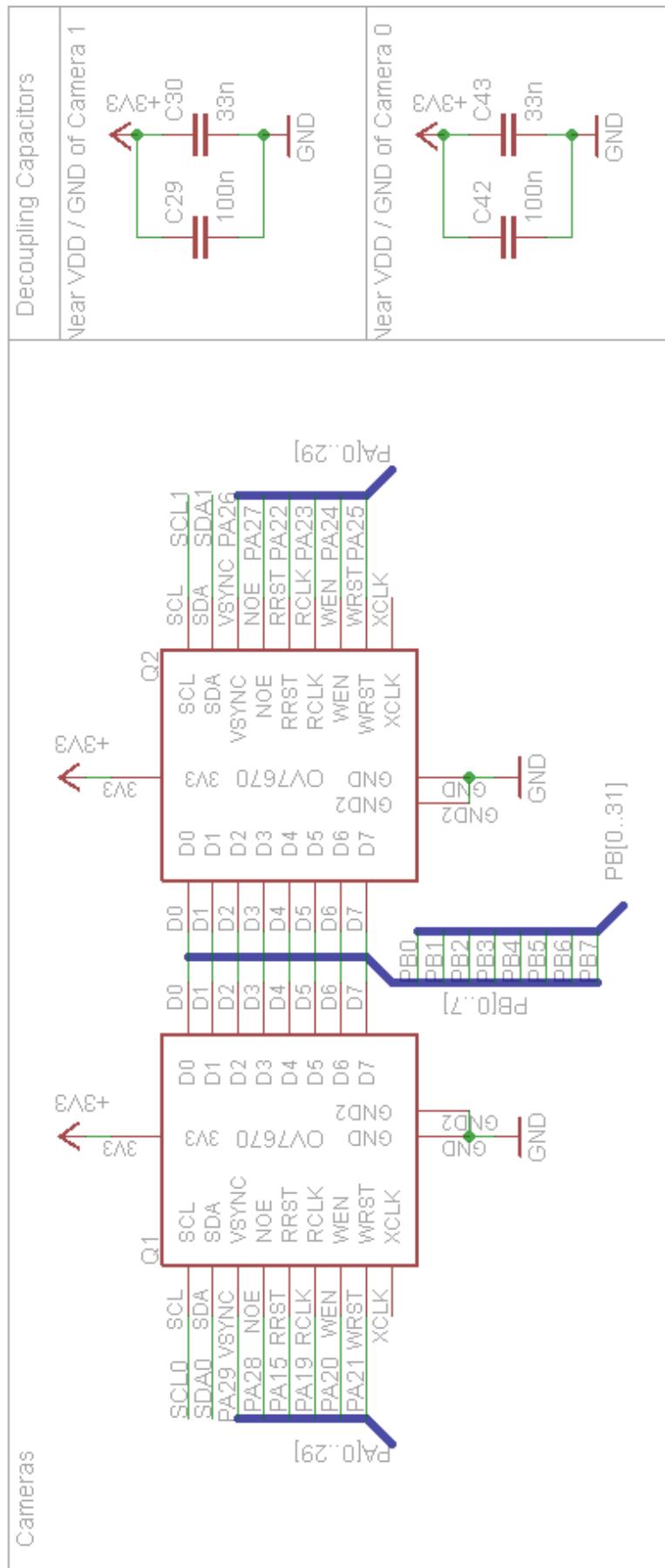


Figure B.5: The Columbus Circuit Diagram Page 3

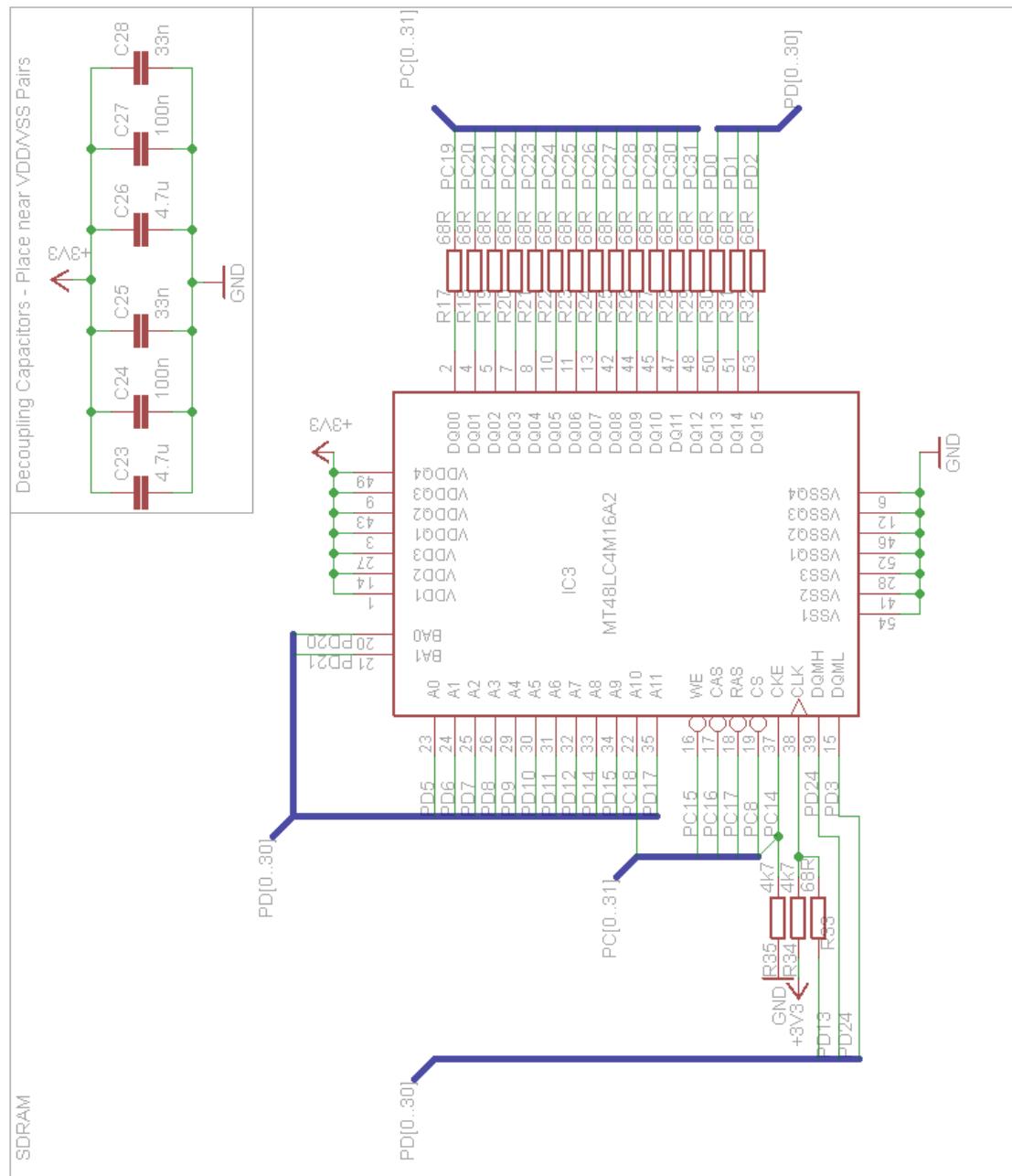


Figure B.6: The Columbus Circuit Diagram Page 4

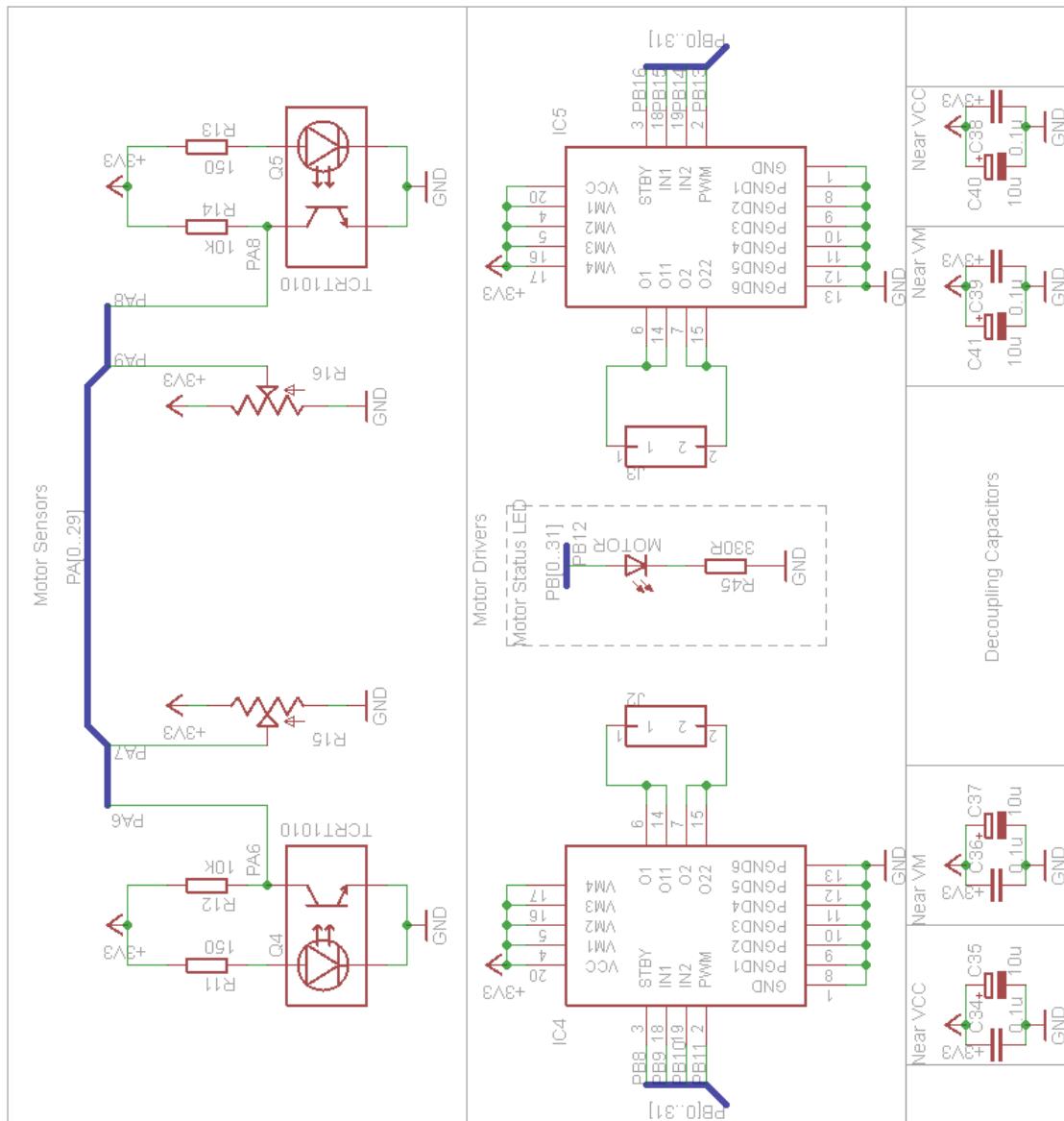


Figure B.7: The Columbus Circuit Diagram Page 5

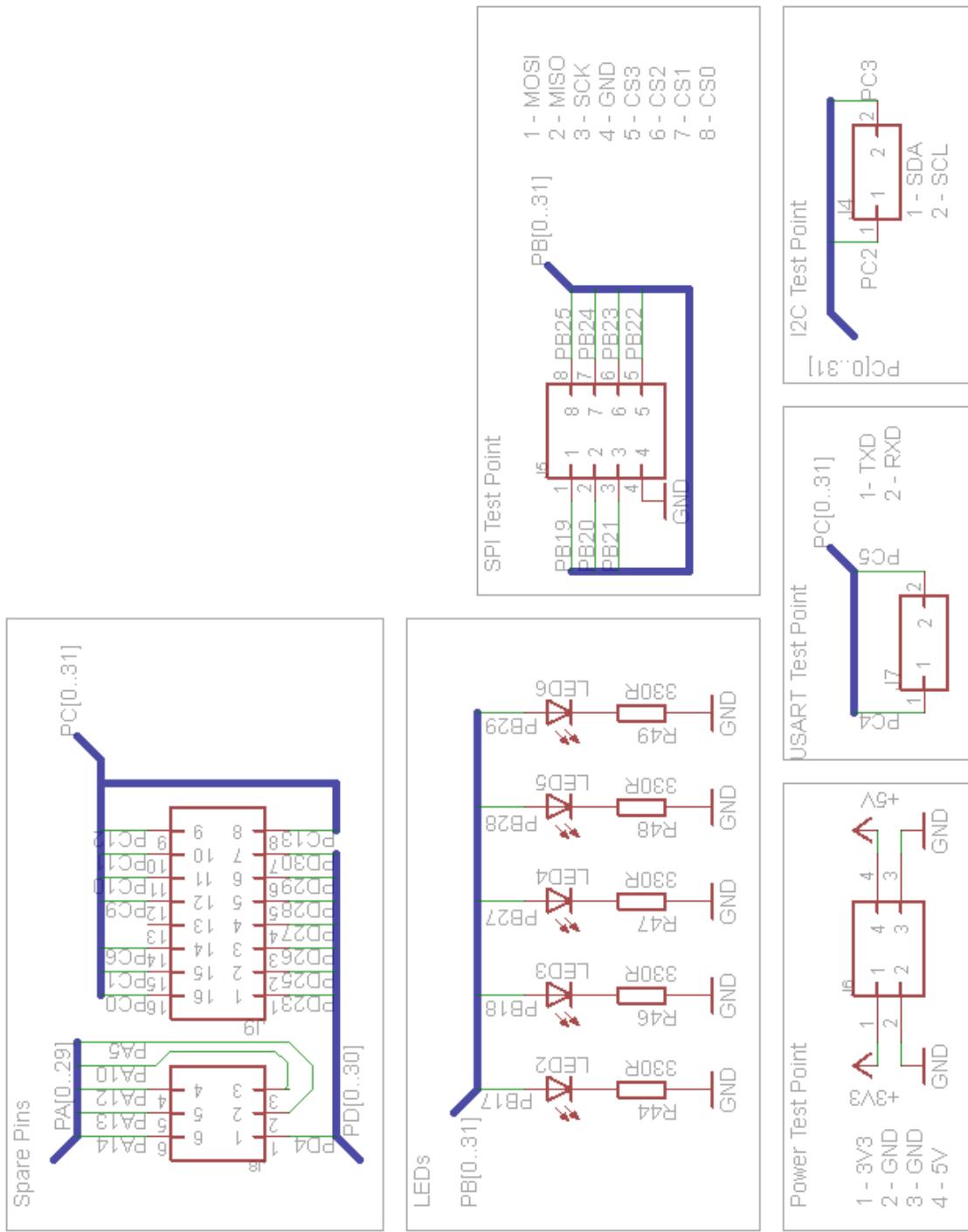


Figure B.8: The Columbus Circuit Diagram Page 6

Appendix C

PCB Design

C.1 PCB Top Side

C.2 PCB Bottom Side

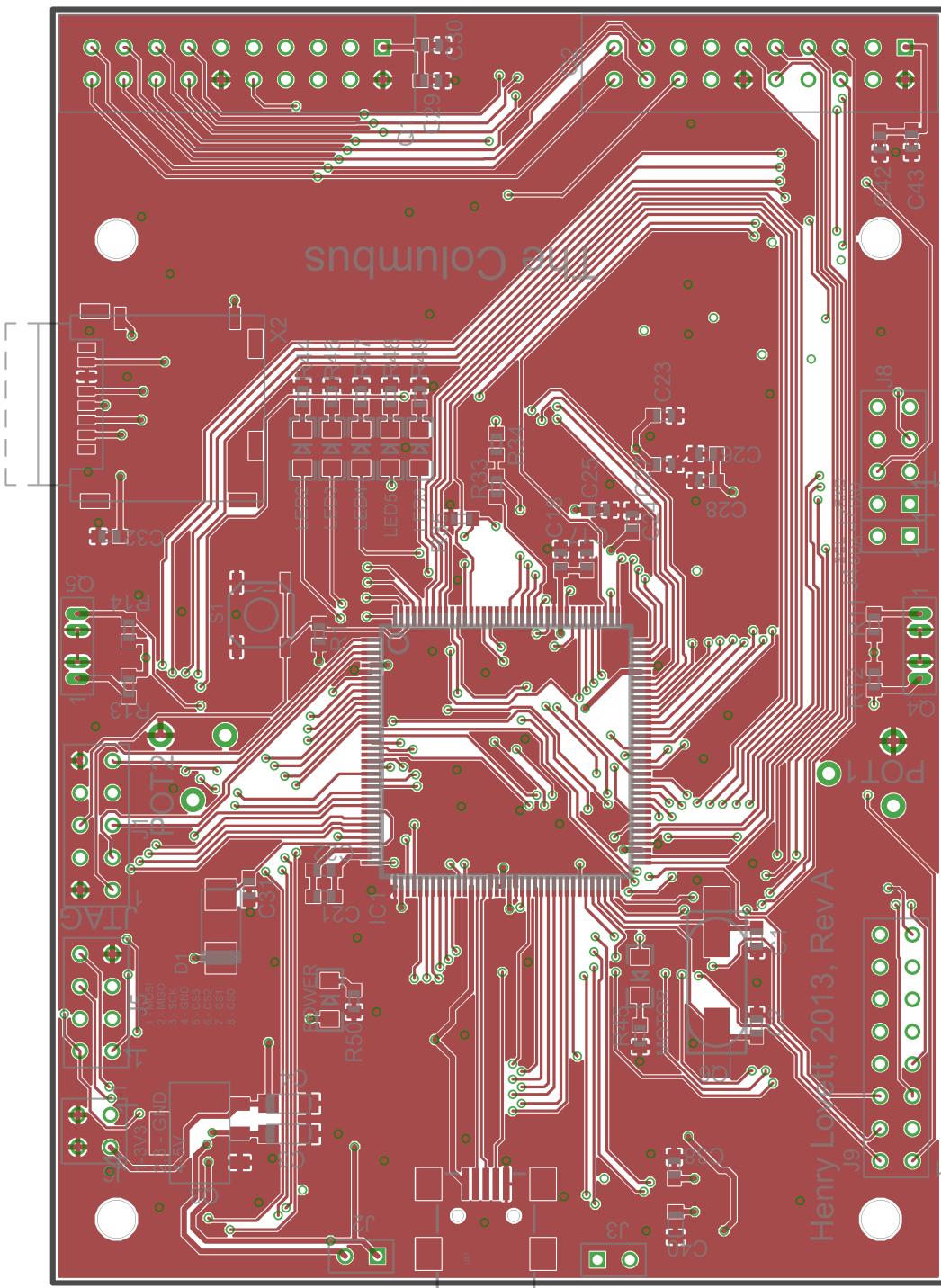


Figure C.1: The Top side of the CAD Design of the PCB

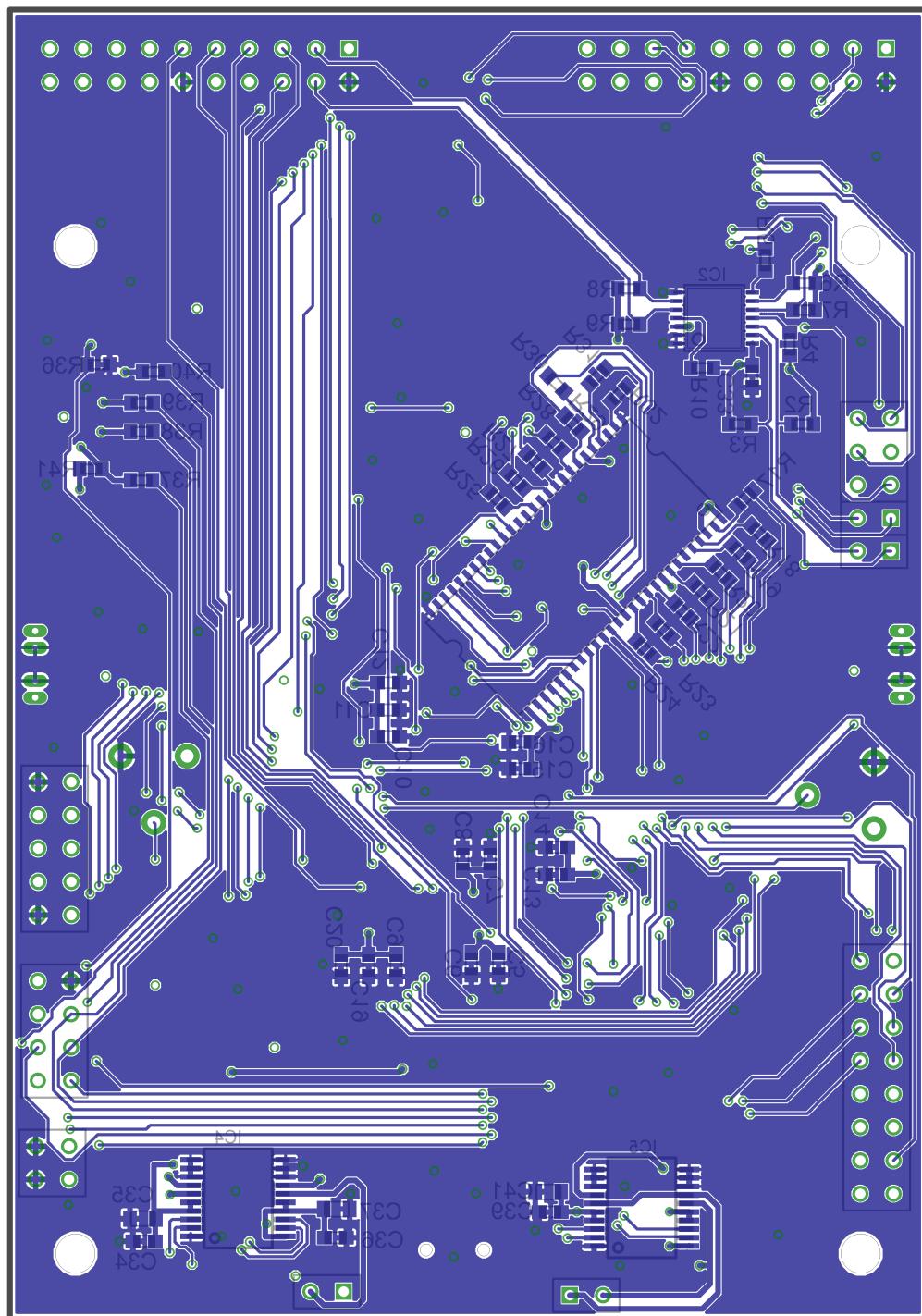


Figure C.2: The Bottom side of the CAD Design of the PCB

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)

Component(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	10kΩ 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	4K7Ω
R1, R12, R14	10kΩ
R36, R41	100kΩ
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	2.2μF
C8, C10, C23, C26	4.7μF
C3, C4, C35, C37, C40, C41	10μF
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

¹This 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 \(2008\)](#).

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

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

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

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

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

$$\frac{\tan(\varphi_1)}{\tan\left(\frac{\varphi_0}{2}\right)} = \frac{2x_1}{x_0} \quad (\text{G.5})$$

$$\tan(\varphi_1) = \frac{2x_1 \tan\left(\frac{\varphi_0}{2}\right)}{x_0} \quad (\text{G.6})$$

This can also be shown for the right camera:

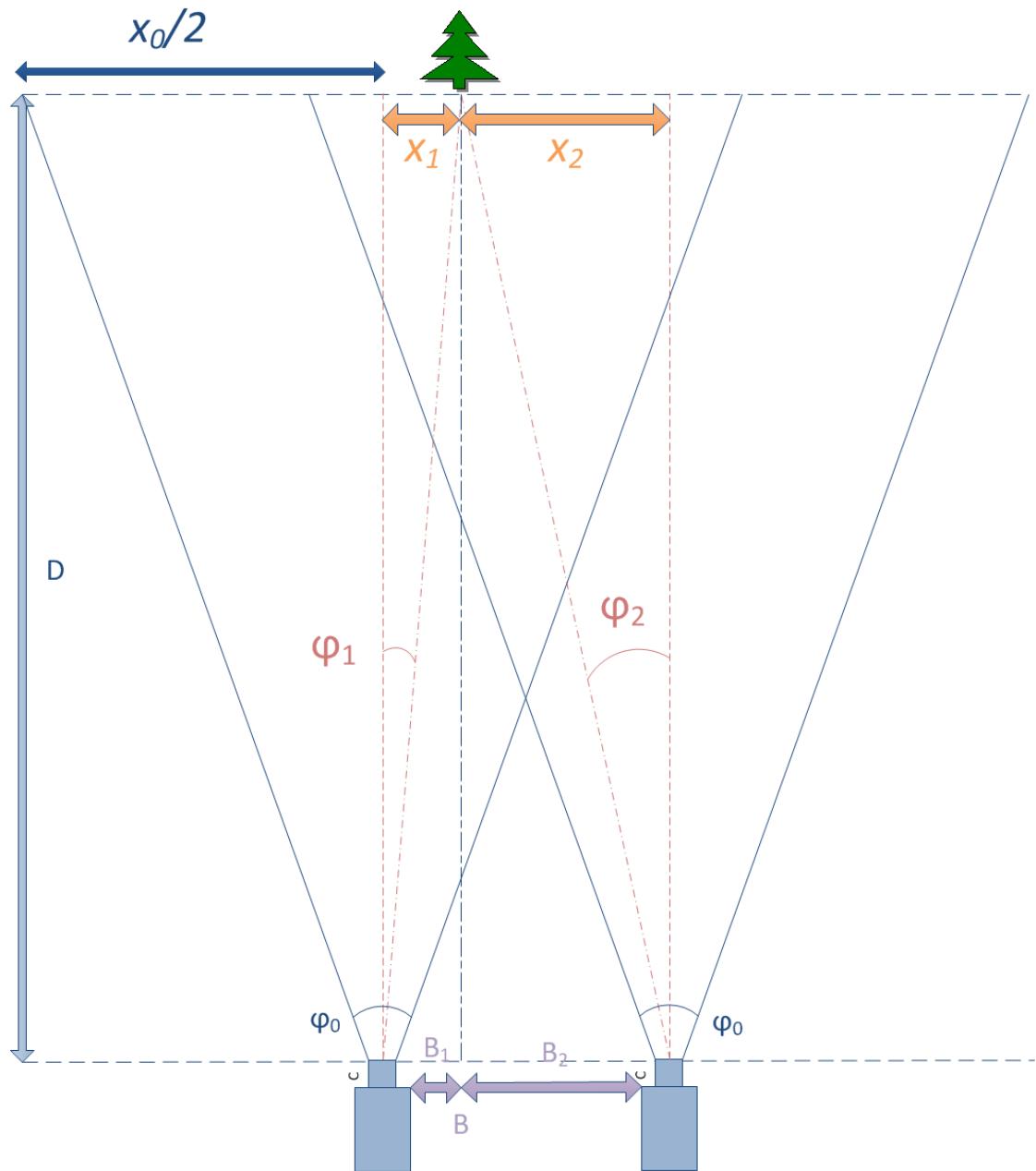


Figure G.1: Problem 1 - Object is between the Cameras

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

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

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

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

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

$$D \cdot \tan(\varphi_1) = x_1 \quad (\text{G.9})$$

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

$$\frac{\tan(\varphi_1)}{\tan\left(\frac{\varphi_0}{2}\right)} = \frac{2x_1}{x_0} \quad (\text{G.11})$$

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

and similarly

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

$$\theta = \varphi_2 - \varphi_1 \quad (\text{G.14})$$

Using the sine equality rule:

$$\frac{R}{\sin\left(\frac{\pi}{2} - \varphi_2\right)} = \frac{B}{\sin(\theta)} \quad (\text{G.15})$$

$$R = B \cdot \frac{\sin\left(\frac{\pi}{2} - \varphi_2\right)}{\sin(\theta)} = B \frac{\cos(\varphi_2)}{\sin(\theta)} \quad (\text{G.16})$$

$$D = \cos(\varphi_1) \cdot R \quad (\text{G.17})$$

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

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

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

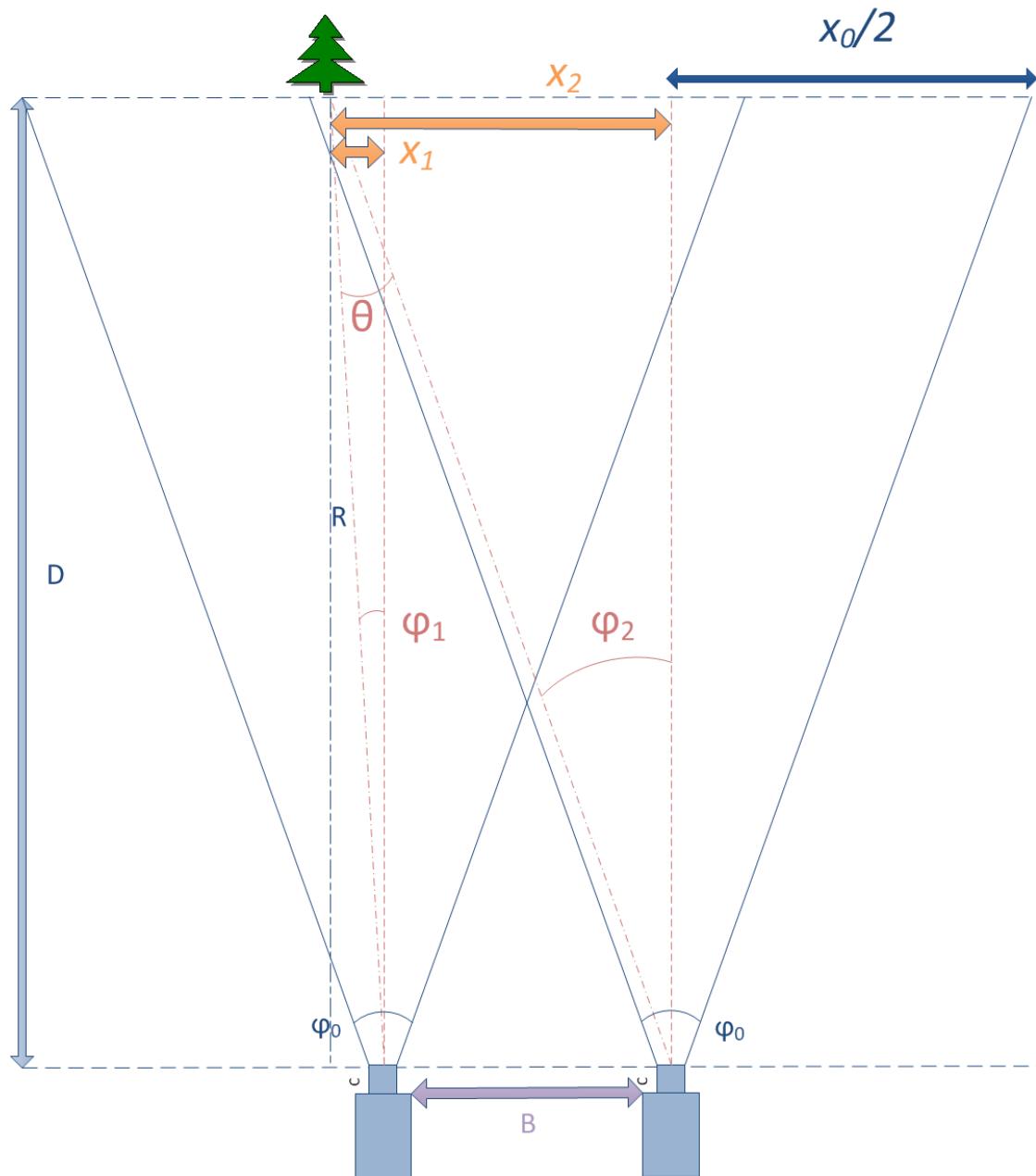


Figure G.2: Problem 2 - Object is to the same side in both cameras

G.3 Object is in front of a camera

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

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

Where φ_2 can be found from equation 4.8.

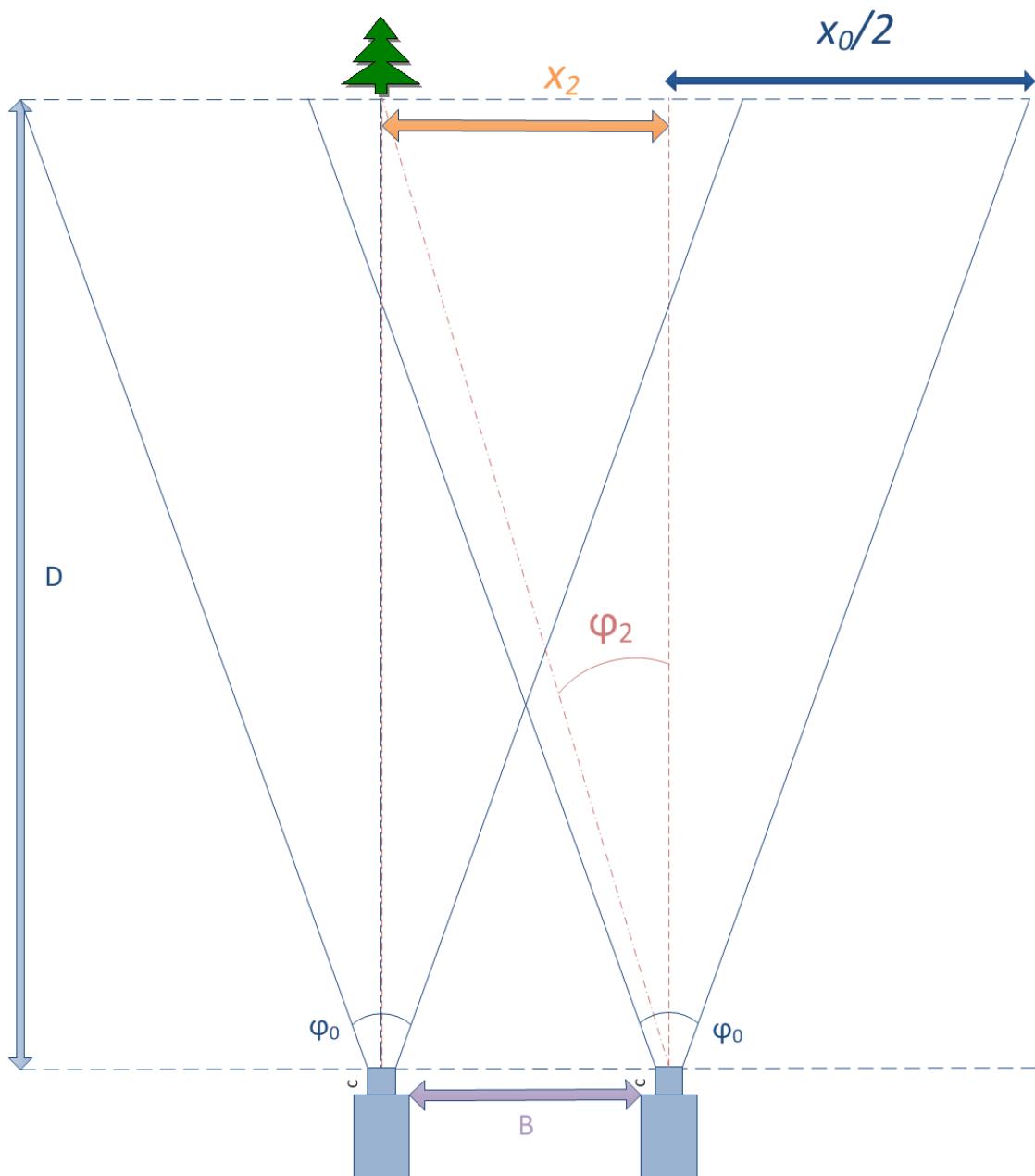


Figure G.3: Problem 3 - Object is directly in front of a camera

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

```
1  /**
2  * \file
3  *
4  * \brief Empty user application template
5  *
6  */
7
8  /*
9  * Include header files for all drivers that have been imported from
10 * Atmel Software Framework (ASF).
11 */
12 #define DSP16_FORMAT 10
13 #define COMMAND_BUFFER_SIZE    128
14 #include <asf.h>
15 #include <conf_board.h>
16 #include "CustomDevices/CustomDevices.h"
17 #include "conf_sd_mmc_spi.h"
18 #include "fat.h"
19 #include "file.h"
20 #include "navigation.h"
21 #include "fastmath.h"
22 #include "delay.h"
23 #include "stdio.h"
```

```
27 void Get_Line( char * CommandBuffer );
28 int Auto_Run();
29 void Debug_Mode();
30 void System_Error();
31
32
33 int main (void)
34 {
35     Columbus_Status.SD_Card = &SD_Status;
36     Columbus_Status.Cameras = &OV7670_Status;
37     Columbus_Status.I2CMux = &PCA9542A;
38     Columbus_Status.SD_Card = &SD_Status;
39     Columbus_Status.Motors = &Motor_Control;
40     board_init();
41     print_dbg("\n\r");
42     print_dbg(THE);
43     print_dbg(COLUMBUS);
44     print_dbg(ASCII_SHIP);
45     System_Test();
46
47     if(Columbus_Status.Status != STATUS_OK)
48     {
49         System_Error();
50     }
51     print_dbg("\n\rColumbus Ready!");
52
53 //Enter into Auto Run if not in debug mode
54 if(gpio_get_pin_value(DEBUG_PIN))
55 {
56     print_dbg("\n\rEntering Auto Run");
57     if(1 == Auto_Run(0)) //if
58         System_Error(); //use the system error loop to stop operation
59 }
60 print_dbg("\n\rEntering Debug Mode... ");
61 Debug_Mode();
62
63 }
64 void Get_Line( char * CommandBuffer )
65 {
66     int c = 0;
67
68     while(c != 13)
69     {
70         c = usart_getchar(DBG_USART);
71         if(c == '\b')
72         {
73             CommandBuffer--;
74             *(CommandBuffer) = 0;
75         }
76         else
77         {
78             *(CommandBuffer) = c;
79             CommandBuffer++;
80         }
81         print_dbg_char(c);
82     }
83     usart_putchar(DBG_USART, 6);
```

```
84    }
85    void PrintStatus()
86    {
87        print_dbg("\n\n\rColumbus_Status:\t0x");
88        print_dbg_hex(Columbus_Status.Status);
89        print_dbg("\n\rSD Card:\n\rStatus:\t\t\t0x");
90        print_dbg_hex(Columbus_Status.SD_Card->Status);
91        print_dbg("\n\rMemory Size :\t\t0x");
92        print_dbg_hex(Columbus_Status.SD_Card->Memory_size);
93        print_dbg("\n\rMotors:");
94        print_dbg("\n\rLeft State :\t\t0x");
95        print_dbg_hex(Columbus_Status.Motors->Left_State);
96        print_dbg("\n\rLeft Count :\t\t0x");
97        print_dbg_hex(Columbus_Status.Motors->Left_Count);
98        print_dbg("\n\rRight State :\t\t0x");
99        print_dbg_hex(Columbus_Status.Motors->Right_State);
100       print_dbg("\n\rRight Count :\t\t0x");
101       print_dbg_hex(Columbus_Status.Motors->Right_Count);
102       print_dbg("\n\rCameras:");
103       print_dbg("\n\rStatus :\t\t0x");
104       print_dbg_hex(Columbus_Status.Cameras->Status);
105       print_dbg("\n\rVSYNC0 State :\t\t0x");
106       print_dbg_hex(Columbus_Status.Cameras->VSYNC0_State);
107       print_dbg("\n\rVSYNC1 State :\t\t0x");
108       print_dbg_hex(Columbus_Status.Cameras->VSYNC1_State);
109       print_dbg("\n\rI2C Mux:");
110       print_dbg("\n\rStatus :\t\t0x");
111       print_dbg_hex(Columbus_Status.I2CMux->Status);
112       print_dbg("\n\rChannel Selected :\t0x");
113       print_dbg_hex(Columbus_Status.I2CMux->ChannelSelected);
114   }
115   void System_Error()
116   {
117       while(1)
118       {
119           LED2_SET;
120           if(Columbus_Status.Status & SD_ERR)
121               LED3_SET;
122           if(Columbus_Status.Status & CAM_ERR)
123               LED4_SET;
124           delay_ms(500);
125
126           LED2_CLR;
127           if(Columbus_Status.Status & SD_ERR)
128               LED3_CLR;
129           if(Columbus_Status.Status & CAM_ERR)
130               LED4_CLR;
131           delay_ms(500);
132           PrintStatus();
133       }
134   }
136 #define AutoRun_Commands_FileName "AutoRun.txt"
137 const AutoCommand_t DefaultCommands[13] = {
138     {'F', 10},
139     {'P', 0},
140     {'R', 90},
141     {'F', 10},
```

```

142     { 'P', 0},
143     { 'R', 90},
144     { 'F', 10},
145     { 'P', 0},
146     { 'R', -90},
147     { 'F', 10},
148     { 'P', 0},
149     { 'R', -90},
150     { 'q', 0}
151 };
152 AutoCommand_t *AutoCommands = DefaultCommands; //Use the default by default.
153 #define ReadBufferSize 256
154 void LoadCommands()
155 {
156     int i = 1;
157     int j, k;
158     char c = 0;
159     char buff[ReadBufferSize];
160     nav_filelist_reset();
161     if(nav_filelist_findname((FS_STRING)AutoRun_Commands_FileName, false))
162     { //If the file exists, load it
163         print_dbg("\n\rAuto Run File Found");
164         nav_setcwd((FS_STRING)AutoRun_Commands_FileName, false, false);
165         file_open(FOPEN_MODE_R);
166         while(!file_eof())//count how many commands there are
167     {
168             c = file_getc();
169             if(c == 0xD)//if a carriage return if found
170                 i++;
171
172         }
173         sprintf(buff, "\n\r%d commands found\n\r", i);
174         print_dbg(buff);
175
176         AutoCommands = malloc(i*sizeof(AutoCommand_t)); //Initialise the array
177
178         file_seek(0, FS_SEEK_SET);
179         i = 0;//command counter
180         c = 0;
181         while(!file_eof())
182     {
183             k = file_read_buf(buff, ReadBufferSize); //read a chunk of the files
184             if(k > ReadBufferSize)
185                 k = ReadBufferSize;
186             //parse it
187             for(j = 0; j < k; )
188             {
189                 while(buff[j] == ' ') //find next non-whitespace
190                     j++;
191                 AutoCommands[i].Command = buff[j++];
192                 while(buff[j] == ' ') //find next non-whitespace
193                     j++;
194                 if((AutoCommands[i].Command == 'F') || (AutoCommands[i].Command == 'B')
195 ) || AutoCommands[i].Command == 'R' || AutoCommands[i].Command == 'J')
196                     AutoCommands[i].Arg = atoi(buff+j);
197                 else
198                     AutoCommands[i].Arg = 0;
199             i++;
200         }
201     }
202 }
```

```
199         while((buff[j++] != 0xA) && (j < k));//find the next line char
200
201     }
202 }
203 file_close();
204 }
205 else
206 { //Load default commands
207     print_dbg("\n\rAuto Run File Not Found, Using Default Commands:");
208 }
209 }

210 int Auto_Run(int Debug)//Debug = 1, don't run commands
211 {
212     int PC = 0;
213
214     LoadCommands();
215     while(1)
216     {
217         switch(AutoCommands[PC].Command)
218         {
219             case 'F'://Move Forward
220                 print_dbg("\n\rMove Forward ");
221                 LED4_SET;
222                 print_dbg_ulong(AutoCommands[PC].Arg);
223                 if(!Debug){
224                     Motors_Move(AutoCommands[PC].Arg);
225                 }
226                 LED4_CLR;
227                 break;
228
229             case 'B'://Move Backward
230                 print_dbg("\n\rMove Backward ");
231                 LED4_SET;
232                 print_dbg_ulong(AutoCommands[PC].Arg);
233                 if(!Debug){
234                     Motors_Move(-AutoCommands[PC].Arg);
235                 }
236                 LED4_CLR;
237                 break;
238
239             case 'R'://Rotate
240                 print_dbg("\n\rRotate ");
241                 LED3_SET;
242                 print_dbg_ulong(AutoCommands[PC].Arg);
243                 if(!Debug){
244                     Motors_Rotate(AutoCommands[PC].Arg);
245                 }
246                 LED3_CLR;
247                 break;
248
249             case 'P'://Take Photo
250                 print_dbg("\n\rTaking Photos");
251                 LED2_SET;
252                 if(!Debug){
253                     FIFO_Reset(CAMERA_LEFT | CAMERA_RIGHT);
254                     if(TakePhoto(CAMERA_LEFT | CAMERA_RIGHT) == CAMERAS_BUSY){
255                         print_dbg("\n\rCameras Busy");
256                     }
257                 }
258             }
259         }
260     }
261 }
```

```
257         break;
258     }
259     while(Photos_Ready() == false)
260     ;
261
262     if(Store_Both_Images() == true)
263     print_dbg("\n\rImages Stored Successfully!");
264     }
265     LED2_CLR;
266     break;
267
268 case 'J': //Jump
269     print_dbg("\n\rProgram Jump to ");
270     print_dbg_ulong(AutoCommands[PC].Arg);
271     if(!Debug){
272     PC = AutoCommands[PC].Arg;
273     }
274     break;
275
276 case 'q': //End and enter debug
277     print_dbg("\n\rQuit to Debug");
278     return 0;
279
280 default: //System Error
281     if(!Debug){//Inevitable in Debug mode
282     Columbus_Status.Status |= AutoRunCMD_ERR;
283     print_dbg("\n\rSystem Error");
284     }
285     return 1;
286   }
287   PC++;
288 }
289 }

293 void Debug_Mode()
294 {
295   Image_t image;
296   unsigned long i, j, tmp = 0;
297   char *Ptr;
298   // volatile unsigned long *sdram = SDRAM;
299   char CommandBuffer[COMMAND_BUFFER_SIZE];
300   int *Working_Buffer = NULL;
301   int SizeOfWorking_Buffer = 0;
302   A_ALIGNED dsp16_complex_t *ComplexBuffer;
303   int SizeOfComplex_Buffer = 0;
304   LED2_SET;//Set these LEDs to indicate Columbus is in debug mode;
305   LED3_SET;
306   LED4_SET;
307   while(1)
308   {
309     print_dbg(PROMPT);
310     Get_Line(CommandBuffer);
311     Ptr = CommandBuffer;
312     switch(*Ptr++)
313     {
314       case '?':
```

```
315     print_dbg(HELP);
316     break;
317 case 'A':
318     print_dbg("\rRunning AutoRun in Debug Mode;");
319     Auto_Run(1); //Runs in debug mode
320     break;
321 case '1':
322     print_dbg("\r1D FFT;");
323     SizeOfComplex_Buffer = FFT_SIZE;
324     ComplexBuffer = mspace_malloc(sdrdram_msp, SizeOfComplex_Buffer *
325     sizeof(ComplexBuffer));
326     i = Get_sys_count();
327     FFT1D(Working_Buffer, ComplexBuffer);
328     i = Get_sys_count() - i;
329     print_dbg("\n\rCycles Taken for 1D FFT = ");
330     print_dbg_ulong(i);
331     break;
332 case '2':
333     print_dbg("\r2D FFT;");
334     FFT2Dabs(Working_Buffer);
335     break;
336 case '3':
337     print_dbg("\rComplex FFT2D:");
338     SizeOfComplex_Buffer = FFT_SIZE * FFT_SIZE;
339     ComplexBuffer = mspace_malloc(sdrdram_msp, SizeOfComplex_Buffer *
340     sizeof(ComplexBuffer));
341     i = Get_sys_count();
342     FFT2DCOMPLEX(Working_Buffer, ComplexBuffer, SizeOfWorking_Buffer);
343     i = Get_sys_count() - i;
344     print_dbg("\n\rCycles Taken for 2D FFT = ");
345     print_dbg_ulong(i);
346     break;
347 case 'm':
348     print_dbg("\r1D FFT Magnitude");
349     FFT1D_Abs(Working_Buffer);
350     break;
351 case 'B':
352     print_dbg("\rReading Bitmap");
353     ReadBitmap("Image_R_0.bmp", &image);
354     print_dbg("\n\rBitmap Data Returned:\n\rImage Height = ");
355     print_dbg_ulong(image.Height);
356     print_dbg("\n\rImage Width = ");
357     print_dbg_ulong(image.Width);
358     break;

359 case 'c':
360     print_dbg("\rConverting Working Buffer to Fixed Point");
361     for(i = 0; i < SizeOfWorking_Buffer ; i++)
362     {
363         Working_Buffer[i] = DSP16_Q (Working_Buffer[i]);
364     }
365     break;

366 case 'C':
367     print_dbg("\rConverting Working Buffer back from Fixed Point");
368     j = DSP16_Q(1);
369     for(i = 0; i < SizeOfWorking_Buffer ; i++)
370     {
```

```

371     Working_Buffer[i] = Working_Buffer[i] / j;
372 }
373 break;

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

380 case 'i':
381     print_dbg("\rImage info:");
382     print_dbg("\n\rImage Pointer = ");
383     print_dbg_ulong(image.ImagePtr);
384     print_dbg("\n\rImage Height = ");
385     print_dbg_ulong(image.Height);
386     print_dbg("\n\rImage Width = ");
387     print_dbg_ulong(image.Width);
388     break;

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

395 case 'k':
396     print_dbg("\rComplex Buffer:\n[r");
397     for (i = 0; i < SizeOfComplex_Buffer; i++)
398     {
399         //            print_dbg_ulong(ComplexBuffer[i].real);
400         //            print_dbg(" + ");
401         //            print_dbg_ulong(ComplexBuffer[i].imag);
402         //            print_dbg(", ");
403         if(ComplexBuffer[i].imag >= 0)
404             sprintf(CommandBuffer, "%d + %dj,", ComplexBuffer[i].real,
ComplexBuffer[i].imag);
405         else
406             sprintf(CommandBuffer, "%d %dj,", ComplexBuffer[i].real,
ComplexBuffer[i].imag);
407         print_dbg(CommandBuffer);
408     }
409     print_dbg("]\n[r");
410     break;
411 case 'M': //Motor Related
412     while(*Ptr == ' ')
413     Ptr++; //Find next non - space char

415 switch(*(Ptr++))
416 {
417     case 'q': // Reset Motors
418         print_dbg("\rResetting Motors");
419         Motors_Reset();
420         break;

422     case 'F': //Move Forward
423         while(*Ptr == ' ')
424             Ptr++; //Find next non - space char
425         i = atoi(Ptr);
426         Motors_Move(i);

```

```
427         break;
428     case 'T':
429         while(*Ptr == ' ')
430             Ptr++; //Find next non - space char
431         i = atoi(Ptr);
432         Motors_Rotate(i);
433         break;
434     case 'L':
435         Columbus_Status.Motors->Left_Count = GAMMA + 1;
436         Columbus_Status.Motors->Left_State = FORWARD;
437         Motor_Start(MOTOR_L);
438         Motors_Execute();
439         break;
440     case 'R':
441         Columbus_Status.Motors->Right_Count = GAMMA + 1;
442         Columbus_Status.Motors->Right_State = FORWARD;
443         Motor_Start(MOTOR_R);
444         Motors_Execute();
445         break;
446     default:
447         print_dbg("\rCommand Not Recognised");
448         break;
449     }

450     break;

453 //    case 'p':
454 //        print_dbg("\rPreparing Image;");
455 //        PrepareImage(&image);
456 //        print_dbg("\rImage Prepared!");
457 //        break;

459     case 'P'://take a photo
460         FIFO_Reset(CAMERA_LEFT | CAMERA_RIGHT);
461         print_dbg("\rTaking Photos");
462         if(TakePhoto(CAMERA_LEFT | CAMERA_RIGHT) == CAMERAS_BUSY){
463             print_dbg("Cameras Busy");
464             break;
465         }
466         while(Photos_Ready() == false)
467         ;

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

473     case 'r':
474         if (Working_Buffer == 0)
475         {
476             print_dbg("\rWorking Buffer Not Initialised");
477             break;
478         }
479         print_dbg("\rWorking Buffer:\n\r[");
480         for(i = 0; i < SizeOfWorking_Buffer; i++)
481         {
482             sprintf(CommandBuffer, "%d,", (dsp16_t)Working_Buffer[i]);
483             print_dbg(CommandBuffer);
484         }
```

```

485     print_dbg("\b\b]\n\r");
486     break;
487 case 'R':
488     Working_Buffer = mspace_malloc(sdram_msp, FFT_SIZE);
489     SizeOfWorking_Buffer = FFT_SIZE;
490     print_dbg("\rReading in signal.bin");
491     ReadSignal(Working_Buffer);
492     break;

494 case 's'://save the working buffer
495     print_dbg("\rSaving Working Buffer;");
496     SaveBuff(Working_Buffer, SizeOfWorking_Buffer);
497     break;
498 case 'd':
499     print_dbg("\rSaving Working Buffer as CSV;");
500     SaveBuff_CSV("Buffer_result.csv", Working_Buffer, SizeOfWorking_Buffer
501 );
502     break;
503 case 'g':
504     print_dbg("\rSaving Complex Buffer as CSV");
505     SaveCBuff_CSV("Buffer_Complex.csv", ComplexBuffer,
506     SizeOfComplex_Buffer);
507     break;
508 case 'f':
509     SizeOfWorking_Buffer = FFT_SIZE*FFT_SIZE;
510     Working_Buffer = mspace_malloc(sdram_msp, SizeOfWorking_Buffer);
511     print_dbg("\rReading in Buffer.csv");
512     Read_CSV("Buffer.csv", Working_Buffer, SizeOfWorking_Buffer);
513     print_dbg("\n\rComplete!");
514     break;

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

521 case 'T':
522     print_dbg("\rReading in 2D Signal");
523     Working_Buffer = mspace_malloc(sdram_msp, FFT_SIZE * FFT_SIZE);
524     SizeOfWorking_Buffer = FFT_SIZE * FFT_SIZE;
525     Read2DSignal(Working_Buffer);
526     break;
527 case 'v':
528     PrintStatus();
529     break;
530 case 'o':
531     print_dbg("\r1 in Fixed point = ");
532     i = DSP16_Q(1);
533     print_dbg_ulong(i);
534     break;

536 default:
537     print_dbg("\rCommand Not Recognised;");
538     break;
539 }

```

```
540     }
541 }
```

H.1.1.2 Bitmap.c

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

```

1  /*
2   * Bitmap.c
3   *
4   * Created: 16/02/2013 23:14:34
5   * Author: hslovett
6   */
7 #include "CustomDevices/CustomDevices.h"

9 const uint8_t DIBHead[DIBHEADERSIZE] = { 0x7C, 0x00, 0x00, 0x00, //Number of
10   bytes
11   0x40, 0x01, 0x00, 0x00, //Width - 320
12   0xF0, 0x00, 0x00, 0x00, //Height - 240
13   0x01, 0x00,           //Planes
14   0x10, 0x00,           //Bits per Pixel
15   0x03, 0x00, 0x00, 0x00, //Compression
16   0x00, 0x58, 0x02, 0x00, //Size of Raw Data
17   0x13, 0x0B, 0x00, 0x00, //Horizontal Resolution
18   0x13, 0x0B, 0x00, 0x00, //Vertical Resolution
19   0x00, 0x00, 0x00, 0x00, //Colours in Palette
20   0x00, 0x00, 0x00, 0x00, //Important Colours
21   0x00, 0xF8, 0x00, 0x00, //Red Mask
22   0xE0, 0x07, 0x00, 0x00, //Green Mask
23   0x1F, 0x00, 0x00, 0x00, //Blue Mask
24   0x00, 0x00, 0x00, 0x00, //Alpha Mask
25   0x01, 0x00, 0x00, 0x00, //Colour Space Type
26   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
27   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
28   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
29   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
30   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
31   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
32   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
33   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
34   0x00, 0x00, 0x00, 0x00, //Gamma Red
35   0x00, 0x00, 0x00, 0x00, //Gamma Green
36   0x00, 0x00, 0x00, 0x00, //Gamma Blue
37   0x03, 0x00, 0x00, 0x00, //Intent - Photo
38   0x00, 0x00, 0x00, 0x00, //ICC Profile Data
39   0x00, 0x00, 0x00, 0x00, //ICC Profile Size
40   0x00, 0x00, 0x00, 0x00}; //Reserved

42 const uint8_t BMPHeader[BMPHEADERSIZE] = { 0x42, 0x4D,
43   0x8A, 0x58, 0x02, 0x00, //Size
44   0x00, 0x00, 0x00, 0x00, //Reserved
45   0x8A, 0x00, 0x00, 0x00 //Offset to Pixel Array
46 };
```

H.1.1.3 CustomDevices.h

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


```
89 #endif /* CUSTOMDEVICES_H_ */
```

H.1.1.4 ImageProcessor.h

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

```

1  /*
2   * ImageProcessor.h
3   *
4   * Created: 28/02/2013 17:46:37
5   * Author: hslovett
6   */
7
8
9 #ifndef IMAGEPROCESSOR_H_
10#define IMAGEPROCESSOR_H_
11
12#define BMP_FORMAT_RGB565      1
13#define BMP_FORMAT_RGB555      2
14#define BMP_FORMAT_GREYSCALE   3
15#define BMP_FORMAT_1xUINT      4
16#define BMP_FORMAT_2xUINT8T    5
17typedef struct {
18    uint16_t *ImagePtr;
19    int Height;
20    int Width;
21    uint8_t Format;
22} Image_t;
23
24
25
26void FFT1D( int *Signal, dsp16_complex_t *ComplexBuffer);
27int FFT2Dabs(int *Signal);
28int log_2(int i);
29
30void FFT2DCOMPLEX( int *Signal, dsp16_complex_t *ComplexBuffer, int size );
31void PrepareImage(Image_t *Image);
32//int* IFFT2D (dsp16_complex_t *Result, dsp16_complex_t *Input);
33void IFFT2D (dsp16_complex_t *Signal); /*Need to test this! */;
34int FFT1D_Abs( int *Signal);
35void Complex_Abs( int *Signal, dsp16_complex_t *ComplexBuffer, int size);
36#endif /* IMAGEPROCESSOR_H_ */
```

H.1.1.5 ImageProcessor.c

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

```

1 /*
2  * ImageProcessor.c
3  *
4  * Created: 28/02/2013 17:46:50
5  */
6
```

```

5   * Author: hslovett
6   */
7 #include <asf.h>
8 #include "CustomDevices/CustomDevices.h"

10 /*#define FFT_SIZE 64*/
11 //Returns log base 2 of i - checks if it is an integer power of 2
12 int log_2(int i)
13 {
14     int ret = 0;
15     if((i & (i - 1)) != 0)
16     {
17         return -1;
18     }
19     while((i & 1) == 0) //while the bit isn't in the lowest bit (already
20         established this is a integer power of 2)
21     {
22         i >>= 1;
23         ret++;
24     }
25
26     return ret;
27 }
28 //*****Method:      FFT2DCOMPLEX
29 // FullName:      FFT2DCOMPLEX
30 // Access:       public
31 // Returns:      int*
32 // Qualifier:
33 // Parameter:    int * Signal
34 // Parameter:    A_ALIGNED dsp16_complex_t * ComplexBuffer
35 // Parameter:    int size
36 //*****
37 void FFT2DCOMPLEX( int *Signal , dsp16_complex_t *ComplexBuffer , int size )
38 {
39     int i, j = 0;
40     int Ptr;
41     Ptr = 0;
42     A_ALIGNED dsp16_complex_t Input_C_1D[FFT_SIZE];
43     A_ALIGNED dsp16_complex_t Result_C_1D[FFT_SIZE];
44     A_ALIGNED dsp16_complex_t Result_C_2D[FFT_SIZE*FFT_SIZE];
45     A_ALIGNED dsp16_t Input_R_1D[FFT_SIZE];
46     //Stage 1 - FFT Real values from Signal. Store VERTICALLY in Result_2D
47     //print_dbg("\n\rStage 1 Reached. Iteration: \n\r");
48     for(i = 0; i < FFT_SIZE; i ++){ //for each row
49         for(j = 0; j < FFT_SIZE; j++){
50             Input_R_1D[j] = Signal[Ptr++]; //copy the data across
51         }
52         //Do the FFT
53         //    print_dbg("\r");
54         //    print_dbg_ulong(i);

55         dsp16_trans_realcomplexfft(Result_C_1D , Input_R_1D , log_2(FFT_SIZE));
56         //Copy data into 2D result TRANSPOSED
57         for(j = 0; j < FFT_SIZE; j++){
58             Result_C_2D[i + (j * FFT_SIZE)].imag = Result_C_1D[j].imag * FFT_SIZE;//
59             scale back up
60             Result_C_2D[i + (j * FFT_SIZE)].real = Result_C_1D[j].real * FFT_SIZE;

```

```

61     }
62 }
63 //print_dbg("\n\rStage 2 Reached. Iteration: \n\r");
64 //Stage 2 - FFT Complex Values from Result_2D, put back into Rows
65 for(i = 0; i < FFT_SIZE; i++){//for each row
66     for(j = 0; j < FFT_SIZE; j++){//copy the data across
67         Input_C_1D[j].imag = Result_C_2D[j + i * FFT_SIZE].imag;
68         Input_C_1D[j].real = Result_C_2D[j + i * FFT_SIZE].real;
69     }
70 //    print_dbg("\r");
71 //    print_dbg_ulong(i);
72 //Do Fourier
73 dsp16_trans_complexfft(Result_C_1D, Input_C_1D, log_2(FFT_SIZE));
74 //Copy back
75 for(j = 0; j < FFT_SIZE; j++){
76     ComplexBuffer[i + j * FFT_SIZE].imag = Result_C_1D[j].imag;
77     ComplexBuffer[i + j * FFT_SIZE].real = Result_C_1D[j].real;
78 }
79 }
80 return;
81 }

83 //One Dimensional Fast Fourier Transform
84 int FFT1D_Abs( int *Signal)
85 {
86     int log2Size, i =0;
87     A_ALIGNED dsp16_complex_t vect1[FFT_SIZE];
88     A_ALIGNED dsp16_t vect2[FFT_SIZE];
89     for(i = 0; i < FFT_SIZE; i++)
90     {
91         vect2[i] = (dsp16_t)Signal[i];
92     }
93     dsp16_trans_realcomplexfft(vect1, vect2, log_2(FFT_SIZE));
94     for(i = 0; i < FFT_SIZE; i++)
95     {
96         vect1[i].imag = vect1[i].imag * FFT_SIZE;
97         vect1[i].real = vect1[i].real * FFT_SIZE;
98     }
99     dsp16_vect_complex_abs(vect2, vect1, FFT_SIZE);
100    for(i = 0; i < FFT_SIZE; i++)
101    {
102        Signal[i] = vect2[i];// * FFT_SIZE;
103    }

105    return Signal;
106 }

108 void Complex_Abs( int *Signal, dsp16_complex_t *ComplexBuffer, int size)
109 {
110     int log2Size, i =0;
111     A_ALIGNED dsp16_complex_t vect1[FFT_SIZE];
112     A_ALIGNED dsp16_t vect2[FFT_SIZE];
113     for(i = 0; i < FFT_SIZE; i++)
114     {
115         vect1[i].imag = ComplexBuffer[i].imag;
116         vect1[i].real = ComplexBuffer[i].real;
117     }
118     dsp16_vect_complex_abs(vect2, vect1, FFT_SIZE);

```

```

119     for(i = 0; i < FFT_SIZE; i++)
120     {
121         Signal[i] = vect2[i];// * FFT_SIZE;
122     }
123 }
124 //One Dimensional Fast Fourier Transform returning complex values
125 void FFT1D( int *Signal, dsp16_complex_t *ComplexBuffer)
126 {
127     int log2Size, i =0;
128     A_ALIGNED dsp16_complex_t vect1[FFT_SIZE];
129     A_ALIGNED dsp16_t vect2[FFT_SIZE];
130     for(i = 0; i < FFT_SIZE; i++)
131     {
132         vect2[i] = Signal[i];
133     }
134     dsp16_trans_realcomplexfft(vect1, vect2, log_2(FFT_SIZE));
135     for(i = 0; i < FFT_SIZE; i++)
136     {
137         ComplexBuffer[i].imag = vect1[i].imag * FFT_SIZE;
138         ComplexBuffer[i].real = vect1[i].real * FFT_SIZE;
139     }
140 }
141 int FFT2Dabs( int *Signal )
142 {
143     int i, j = 0;
144     int Ptr;
145     Ptr = 0;
146     A_ALIGNED dsp16_complex_t Input_C_1D[FFT_SIZE];
147     A_ALIGNED dsp16_complex_t Result_C_1D[FFT_SIZE];
148     A_ALIGNED dsp16_complex_t Result_C_2D[FFT_SIZE*FFT_SIZE];
149     A_ALIGNED dsp16_t Input_R_1D[FFT_SIZE];

151     //Stage 1 - FFT Real values from Signal. Store VERTICALLY in Result_2D
152     for(i = 0; i < FFT_SIZE; i++) //for each row
153     {
154 //      print_dbg("\n\rInput to FFT: \n\r");
155 //      for(j = 0; j < FFT_SIZE; j++)
156 //      {
157 //          Input_R_1D[j] = Signal[Ptr++]; //copy the data across
158 //          print_dbg_ulong(Input_R_1D[j]);
159 //          print_dbg(" , ");
160 //      }
161 //      print_dbg("\b\b");
162 //Do the FFT
163     dsp16_trans_realcomplexfft(Result_C_1D, Input_R_1D, log_2(FFT_SIZE));
164 //Copy data into 2D result TRANSPOSED
165 //      print_dbg("\n\rOutput of FFT:\n\r");
166     for(j = 0; j < FFT_SIZE; j++)
167     {
168         Result_C_2D[i + (j * FFT_SIZE)].imag = Result_C_1D[j].imag * FFT_SIZE;// scale back up
169         Result_C_2D[i + (j * FFT_SIZE)].real = Result_C_1D[j].real * FFT_SIZE;
170 //         print_dbg_ulong(Result_C_2D[i + (j * FFT_SIZE)].real);
171 //         print_dbg(" + j");
172 //         print_dbg_ulong(Result_C_2D[i + (j * FFT_SIZE)].imag);
173 //         print_dbg(" , ");
174     }
175 //      print_dbg("\b\b");

```

```

176     }
177     //Stage 2 - FFT Complex Values from Result_2D, put back into Rows
178
179     for(i = 0; i < FFT_SIZE; i++)//for each row
180     {
181         //    print_dbg("\n\rInput to FFT: \n\r[");
182         //    for(j = 0; j < FFT_SIZE; j++)//copy the data across
183         {
184             Input_C_1D[j].imag = Result_C_2D[j + i * FFT_SIZE].imag;
185             Input_C_1D[j].real = Result_C_2D[j + i * FFT_SIZE].real;
186             //    print_dbg_ulong(Input_C_1D[j].real);
187             //    print_dbg(" + j");
188             //    print_dbg_ulong(Input_C_1D[j].imag);
189             //    print_dbg(" , ");
190         }
191         //    print_dbg("\b\b]");
192         //Do Fourier
193         dsp16_trans_complexfft(Result_C_1D, Input_C_1D, log_2(FFT_SIZE));
194         //Copy back
195
196         //    print_dbg("\n\rOutput to FFT: \n\r[");
197         //    for(j = 0; j < FFT_SIZE; j++)//copy the data across
198         //{
199             //    print_dbg_ulong(Result_C_1D[j].real);
200             //    print_dbg(" + j");
201             //    print_dbg_ulong(Result_C_1D[j].imag);
202             //    print_dbg(" , ");
203         }
204         //    print_dbg("\b\b]");
205         //Calculate Abs and put back into Signal TRANSPOSED
206         dsp16_vect_complex_abs(Input_R_1D, Result_C_1D, FFT_SIZE);
207
208         for(j = 0; j < FFT_SIZE; j++)
209         {
210             Signal[i + (j*FFT_SIZE)] = Input_R_1D[j] * FFT_SIZE;
211         }
212     }
213     return Signal;
214 }

215
216
217
218
219 void PrepareImage(Image_t *Image)
220 {
221     int row, col;
222     uint16_t *PreparedImage;
223     //Allocate some memory in the RAM
224     PreparedImage = mspace_malloc(sdram_msp, 256*256);

225     //print_dbg("\n\rPrepared Image Pointer = ");
226     //print_dbg_ulong(PreparedImage);
227     for(row = 0; row < 256; row++)
228     {
229         for(col = 0; col < 256; col++)
230         {
231             if(row < 240)
232                 PreparedImage[row*256 + col] = Image->ImagePtr[row * 256 + col];
233

```

```

234     else
235         PreparedImage[row *256 + col] = 0; //Image->ImagePtr[(row - 240) * 256
236         + col + 32];
236     }
237 }

239 mspace_free(sdram_msp, Image->ImagePtr); //free up the old image
240 Image->ImagePtr = PreparedImage; //move the pointer to the prepared image
241 Image->Height = 256;
242 Image->Width = 256;
243 //SaveBitmap(PreparedImage, 256, 256, "PreparedImage.bmp");
244 //mspace_free(sdram_msp, PreparedImage);
245 //return PreparedImage;
246 }

249 //*****
250 // Method: IFFT2D
251 // FullName: IFFT2D
252 // Access: public
253 // Returns: void
254 // Qualifier:
255 // Parameter: dsp16_complex_t * Signal
256 //*****
257 void IFFT2D (dsp16_complex_t *Signal) //Need to test this!
258 {
259     int i, j = 0;
260     int Ptr;
261     Ptr = 0;
262     A_ALIGNED dsp16_complex_t Input_C_1D[FFT_SIZE];
263     A_ALIGNED dsp16_complex_t Result_C_1D[FFT_SIZE];
264     A_ALIGNED dsp16_complex_t Result_C_2D[FFT_SIZE*FFT_SIZE];
265     A_ALIGNED dsp16_t Input_R_1D[FFT_SIZE];

268     //Stage 1 - FFT Real values from Signal. Store VERTICALLY in Result_2D
269     for(i = 0; i < FFT_SIZE; i++) //for each row
270     {
271         for(j = 0; j < FFT_SIZE; j++)
272         {
273             Input_C_1D[j].real = Signal[Ptr].real; //copy the data across
274             Input_C_1D[j].imag = Signal[Ptr].imag;
275         }

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

281         for(j = 0; j < FFT_SIZE; j++)
282         {
283             Result_C_2D[i + (j * FFT_SIZE)].imag = Result_C_1D[j].imag * FFT_SIZE;//
284             scale back up
285             Result_C_2D[i + (j * FFT_SIZE)].real = Result_C_1D[j].real * FFT_SIZE;
286         }

287     }
288     //Stage 2 - FFT Complex Values from Result_2D, put back into Rows

```

```

291     for(i = 0; i < FFT_SIZE; i++) //for each row
292     {
293
294         for(j = 0; j < FFT_SIZE; j++) //copy the data across
295         {
296             Input_C_1D[j].imag = Result_C_2D[j + i * FFT_SIZE].imag;
297             Input_C_1D[j].real = Result_C_2D[j + i * FFT_SIZE].real;
298
299         }
300
301         //Do Fourier
302         dsp16_trans_complexifft(Result_C_1D, Input_C_1D, log_2(FFT_SIZE));
303         //Copy back
304
305
306         //Put back into Signal TRANSPOSED
307         //dsp16_vect_complex_abs(Input_R_1D, Result_C_1D, FFT_SIZE);
308
309         for(j = 0; j < FFT_SIZE; j++)
310         {
311             Signal[i + j * FFT_SIZE].imag = Result_C_1D[j].imag;
312             Signal[i + j * FFT_SIZE].real = Result_C_1D[j].real;
313             //Signal[i + (j*FFT_SIZE)] = Input_R_1D[j] * FFT_SIZE;
314         }
315     }
316
317     //return Signal;
318 }
319
320 void ComplexMultiply(dsp16_complex_t *Result_Input1, dsp16_complex_t *Input2,
321 int size)
322 {
323     int i = 0;
324     dsp16_complex_t c;
325     for(i = 0; i < size; i++)
326     {
327         // (a+jb).(c+jd) = (ac - bd) + j(ad + bc)
328         c.real = (Result_Input1[i].real * Input2[i].real) - (Result_Input1[i].imag
329             * Input2[i].imag);
330         c.imag = (Result_Input1[i].real * Input2[i].imag) + (Result_Input1[i].imag
331             * Input2[i].real);
332         Result_Input1[i].imag = c.imag;
333         Result_Input1[i].real = c.real;
334     }
335 }
```

H.1.1.6 MotorDriver.h

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

```

1  /*
2   * MotorDriver.h
3   *
```

```
4  * Created: 10/02/2013 18:11:55
5  * Author: hslovett
6  */

9 #ifndef MOTORDRIVER_H_
10 #define MOTORDRIVER_H_
11 #include <asf.h>
12 //Definitions
13 #define MOTOR_L      ML_PWM_CHANNEL_ID
14 #define MOTOR_R      MR_PWM_CHANNEL_ID

16 #define FORWARD      2
17 #define BACKWARD     3
18 #define LEFT_SPOT    4
19 #define RIGHT_SPOT   5
20 #define SPOT_PIVOT   6
21 #define STOP         7

24 #define ENABLE ACA_INTERRUPT    // {AVR32_ACIFA1.iер = 1;}
25 #define DISABLE ACA_INTERRUPT  // {AVR32_ACIFA1.idр = 1;}
26 #define ENABLE ACB_INTERRUPT    // {AVR32_ACIFA1.iер = 2;}
27 #define DISABLE ACB_INTERRUPT  // {AVR32_ACIFA1.idр = 2;}
28 #define GAMMA 10 //Interrupts caused per full rotation of a wheel
29 #define CIRCUMFERENCE_WHEEL_MM 116 //in millimeters
30 // #define CIRCUMFERENCE_WHEEL_CM 12 //in centimeters
31 #define MIN_DISTANCE_RESOLUTION CIRCUMFERENCE_WHEEL_MM / GAMMA
32 #define MIN_ROTATION_RESOLUTION (CIRCUMFERENCE_WHEEL_MM*360)/(GAMMA * C_b)
33 #define C_b             282
34 #define ROTATION_CONST    (GAMMA * C_b) / (CIRCUMFERENCE_WHEEL_MM * 360)
35 #define ROTATION_CONST_INV (CIRCUMFERENCE_WHEEL_MM * 360) / (GAMMA *
36           C_b)
36 //Type Defs
37 typedef struct {
38     int Left_State;
39     int Right_State;
40     int Left_Count;
41     int Right_Count;
42 } Motor_Control_t;

44 //Globals
45 pwm_opt_t pwm_opt;           // PWM option config.
46 avr32_pwm_channel_t pwm_channel;
47 Motor_Control_t Motor_Control;

49 void Motor_Init();
50 void Motor_Go();
51 //void Analogue_Comparator_Init();
52 void Motor_Start(int Motors);
53 void Motors_Reset(void);
54 void Motor_Stop(int Motors);
55 bool Motors_Moving();
56 int Motors_Move(int centimetres_fwd)/*Move this amount forward in centimeters
57           */;
57 void Motors_Execute();
58 int Motors_Rotate(int angle_degs);
59 /*static void ACISignHandler(void);*/
```

```
60 #endif /* MOTORDRIVER_H_ */
```

H.1.1.7 MotorDriver.c

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

```

1  /*
2   * MotorDriver.c
3   *
4   * Created: 10/02/2013 18:12:07
5   * Author: hslogett
6   */
7 #include <asf.h>
8 #include "CustomDevices/CustomDevices.h"
9 #include <delay.h>

13 static void local_start_highfreq_clock(void)
14 {
15     const scif_pll_opt_t opt = {
16         .osc = SCIF_OSC0,           // Sel Osc0/PLL0 or Osc1/PLL1
17         .lockcount = 16,           // lockcount in main clock for the PLL wait
18         lock
19         .div = 1,                  // DIV=1 in the formula
20         .mul = 6,                  // MUL=7 in the formula
21         .pll_div2 = 1,             // pll_div2 Divide the PLL output frequency
22         by 2 (this settings does not change the FVCO value)
23         .pll_wbwdisable = 0,        // pll_wbwdisable 1 Disable the Wide-Bandith
24         Mode (Wide-Bandith mode allow a faster startup time and out-of-lock time
25         ). 0 to enable the Wide-Bandith Mode.
26         .pll_freq = 1,              // Set to 1 for VCO frequency range 80-180
27         MHz, set to 0 for VCO frequency range 160-240Mhz.
28     };
29     // Switch main clock to Osc0.
30     // pcl_switch_to_osc(PCL_OSC0, FOSCO, OSCO_STARTUP);

31     /* Setup PLL0 on Osc0, mul=7 ,no divisor, lockcount=16, ie. (16Mhz*7)/(div2)
32      = 56MHz output */
33     scif_pll_setup(SCIF_PLL0, &opt); // lockcount in main clock for the PLL wait
34     lock

35     /* Enable PLL0 */
36     scif_pll_enable(SCIF_PLL0);

37     /* Wait for PLL0 locked */
38     scif_wait_for_pll_locked(SCIF_PLL0) ;
39 }
40 static void pwm_start_gc(void)
41 {
42     scif_gc_setup(AVR32_SCIF_GCLK_PWM,
43                 SCIF_GCCTRL_PLL0,
44                 AVR32_SCIF_GC_NO_DIV_CLOCK,
45                 0);

```

```
42     scif_gc_enable(AVR32_SCIF_GCLK_PWM);
43 }
44 void Analogue_Comparator_Init()
45 {
46     static const gpio_map_t ACIFA_GPIO_MAP =
47     {
48         {POTO_AC1AP1_PIN, POTO_AC1AP1_FUNCTION},
49         {POT1_AC1BP1_PIN, POT1_AC1BP1_FUNCTION},
50         {SENSEO_AC1AN1_PIN, SENSEO_AC1AN1_FUNCTION},
51         {SENSE1_AC1BN1_PIN, SENSE1_AC1BN1_FUNCTION},
52     };
53
54     gpio_enable_module(ACIFA_GPIO_MAP, sizeof(ACIFA_GPIO_MAP) / sizeof(
55         ACIFA_GPIO_MAP[0]));
56     //Make it an interrupt
57     Disable_global_interrupt();
58
59     acifa_configure_hysteresis(&AVR32_ACIFA1, ACIFA_COMP_SELA, 2);
60     acifa_configure(&AVR32_ACIFA1,
61         ACIFA_COMP_SELA,
62         POTO_AC1AP1_INPUT,
63         SENSEO_AC1AN1_INPUT,
64         FOSCO);
65
66     acifa_configure_hysteresis(&AVR32_ACIFA1, ACIFA_COMP_SELB, 2);
67     acifa_configure(&AVR32_ACIFA1,
68         ACIFA_COMP_SELB,
69         POT1_AC1BP1_INPUT,
70         SENSE1_AC1BN1_INPUT,
71         FOSCO);
72
73     // acifa_enable_interrupt(&AVR32_ACIFA1, (1 << AVR32_ACIFA_ACBINT )| (1 <<
74     // AVR32_ACIFA_ACAINT));//Enable ACBINT and ACAINT
75     // AVR32_ACIFA1.ier = 3; //enable interrupts
76     // acifa_enable_interrupt_toggle(&AVR32_ACIFA1, ACIFA_COMP_SELA);
77     // acifa_enable_interrupt_toggle(&AVR32_ACIFA1, ACIFA_COMP_SELB);
78     // acifa_enable_interrupt_inp_lower(&AVR32_ACIFA1, ACIFA_COMP_SELA);
79     // acifa_enable_interrupt_inp_lower(&AVR32_ACIFA1, ACIFA_COMP_SELB);
80     acifa_start(&AVR32_ACIFA1, (ACIFA_COMP_SELA|ACIFA_COMP_SELB));
81
82     // INTC_register_interrupt(&ACIInterruptHandler, AVR32_ACIFA1_IRQ ,
83     // AVR32_INTC INTO);
84
85     Enable_global_interrupt();
86 }
87 void Motor_Init()
88 {
89     //Turn boths motors off
90     ML_STANDBY;
91     MR_STANDBY;
92
93     ML_IN1_CLR;
94     ML_IN2_CLR;
95
96     MR_IN1_CLR;
97     MR_IN2_CLR;
98
99     Motor_Control.Left_Count = 0;
```

```

97     Motor_Control.Right_Count = 0;
98     Motor_Control.Left_State = STOP;
99     Motor_Control.Right_State = STOP;

101    avr32_pwm_channel_t pwm_channel = { {0}, // cmr
102                                         {0}, // cdt
103                                         {0}, // cdtupd
104                                         {0}, // cpd
105                                         {0}, // cpdrupd
106                                         {0}, // ccnt
107                                         {0}, // dt
108                                         {0}}; // dtupd ; One channel config.
109 /* unsigned int channel_id; */

111    // Start PLL for PWM
112    local_start_highfreq_clock();
113    // Start Enable Generic Clock with PLL as source clock
114    pwm_start_gc();
115    gpio_enable_module_pin(ML_PWM_H_PIN, ML_PWM_H_FUNCTION);
116    gpio_enable_module_pin(MR_PWM_H_PIN, MR_PWM_H_FUNCTION);
117    // PWM controller configuration.
118    pwm_opt.diva = AVR32_PWM_DIVA_CLK_OFF;
119    pwm_opt.divb = AVR32_PWM_DIVB_CLK_OFF;
120    pwm_opt.prea = AVR32_PWM_PREA_CCK;
121    pwm_opt.preb = AVR32_PWM_PREB_CCK;
122    pwm_opt.fault_detection_activated = false;
123    pwm_opt.sync_channel_activated = true;
124    pwm_opt.sync_update_channel_mode =
        PWM_SYNC_UPDATE_MANUAL_WRITE_MANUAL_UPDATE;
125    pwm_opt.sync_channel_select[0] = false;
126    pwm_opt.sync_channel_select[1] = false;
127    pwm_opt.sync_channel_select[2] = false;
128    pwm_opt.sync_channel_select[3] = false;
129    pwm_opt.cksel = PWM_CKSEL_GCLK;
130    pwm_init(&pwm_opt);
131    // Update the period
132    pwm_update_period_value(10);
133    // Channel configuration
134    pwm_channel.CMR.dte = 0; // Enable Deadtime for complementary
      Mode
135    pwm_channel.CMR.dthi = 0; // Deadtime Inverted on PWMH
136    pwm_channel.CMR.dtli = 0; // Deadtime Not Inverted on PWML
137    pwm_channel.CMR.ces = 0; // 0/1 Channel Event at the End of
      PWM Period
138    pwm_channel.CMR.calg = PWM_MODE_LEFT_ALIGNED; // Channel mode.
139    pwm_channel.CMR.cpol = PWM_POLARITY_HIGH; // Channel polarity.
140    pwm_channel.CMR.cpre = AVR32_PWM_CPRE_CCK; // Channel prescaler.
141    pwm_channel.cdt = 50; // Channel duty cycle, should be <
      CPRD.
142    pwm_channel.cprd = 200; // Channel period.

144    pwm_channel_init(ML_PWM_CHANNEL_ID, &pwm_channel); // Set channel
      configuration to channel 0
145    pwm_channel_init(MR_PWM_CHANNEL_ID, &pwm_channel); // Set channel
      configuration to channel 0
146    Analogue_Comparator_Init();
147 }

```

```
149 // __attribute__((__interrupt__)) static void ACInterruptHandler(void)
150 // {
151 // }
152 // }

154 void Motor_Start(int Motors)
155 {
156     if(Motors & MOTOR_L)
157     {
158         if(Motor_Control.Left_State == FORWARD)
159         {
160             ML_IN1_SET;
161             ML_IN2_CLR;
162         }
163         else if (Motor_Control.Left_State == BACKWARD)
164         {
165             ML_IN1_CLR;
166             ML_IN2_SET;
167         }
168         else //Somethings gone wrong
169         {
170             ML_IN1_CLR;
171             ML_IN2_CLR;
172             return;//don't start any pwm channel
173         }
174         ML_GO;
175         pwm_start_channels((1 << MOTOR_L)); //Start PWM Channel on M0 line
176     }

178     if(Motors & MOTOR_R)
179     {
180         if(Motor_Control.Right_State == FORWARD)
181         {
182             MR_IN1_SET;
183             MR_IN2_CLR;
184         }
185         else if (Motor_Control.Right_State == BACKWARD)
186         {
187             MR_IN1_CLR;
188             MR_IN2_SET;
189         }
190         else //Somethings gone wrong
191         {
192             MR_IN1_CLR;
193             MR_IN2_CLR;
194             return;//don't start any pwm channel
195         }
196         MR_GO;
197         pwm_start_channels((1 << MOTOR_R));
198     }
199 }

201 #define HYST_MAX 10
202 #define SAMPLE_TIME 10
203 void Motors_Execute()
204 {
205     LEDMOTOR_SET;
206     int Left_State = acifa_is_acb_inp_higher(&AVR32_ACIFA1);
```

```

207 int Right_State = acifa_is_aca_inp_higher(&AVR32_ACIFA1);
208 int Left_Hyst = 0;
209 int Right_Hyst = 0;
210 bool Left_OnTab = false;
211 bool Right_OnTab = false;

213 while(Motors_Moving())
214 {
215 ////////////////LEFT MOTOR/////////////////////////////
216 // LEFT MOTOR /////////////////////////////////
217 /////////////////////////////////
218 Left_OnTab = acifa_is_acb_inp_higher(&AVR32_ACIFA1); //Read the status of
the comparator

220 if(Left_State == 0) //State Not on Tab
221 {
222     LED5_CLR;
223     if(Left_OnTab == true)
224     {
225         Left_Hyst++;
226         if(Left_Hyst == HYST_MAX)//reached threshold, change state
227         {
228             Left_State = 1;
229             Motor_Control.Left_Count--; //Decrement the global counter
230             Left_Hyst = 0;
231             print_dbg("\n\rLeft Count Decrement");
232         }
233     }
234     else //Reset counter
235     {
236         Left_Hyst = 0;
237     }
238 }
239 else //State On Tab
240 {
241     LED5_SET;
242     if(Left_OnTab == true)//still on a tab
243     {
244         Left_Hyst = 0;//Reset counter
245     }
246     else
247     {
248         Left_Hyst++;
249         if(Left_Hyst == HYST_MAX)//Reached Threshold. Change state
250         {
251             Left_State = 0;
252             Left_Hyst = 0;
253         }
254     }
255 }
256 ////////////////RIGHT MOTOR/////////////////////////////
257 // RIGHT MOTOR /////////////////////////////////
258 /////////////////////////////////
259 Right_OnTab = acifa_is_aca_inp_higher(&AVR32_ACIFA1); //Read the status of
the comparator

261 if(Right_State == 0) //State Not on Tab
262 {

```

```
263     LED6_CLR;
264     if(Right_OnTab == true)
265     {
266         Right_Hyst++;
267         if(Right_Hyst == HYST_MAX)//reached threshold, change state
268         {
269             Right_State = 1;
270             Motor_Control.Right_Count--; //Decrement the global counter
271             Right_Hyst = 0;
272             print_dbg("\n\rRight Count Decrement");
273         }
274     }
275     else //Reset counter
276     {
277         Right_Hyst = 0;
278     }
279 }
280 else //State On Tab
281 {
282     LED6_SET;
283     if(Right_OnTab == true)//still on a tab
284     {
285         Right_Hyst = 0;//Reset counter
286     }
287     else
288     {
289         Right_Hyst++;
290         if(Right_Hyst == HYST_MAX)//Reached Threshold. Change state
291         {
292             Right_State = 0;
293             Right_Hyst = 0;
294         }
295     }
296 }
297 int temp = 0;
298 if(Motor_Control.Left_Count <= 0) //if we have reached the end of the
movement on left wheel
299     temp |= MOTOR_L;

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

304 //Delay to keep a low sample time
305 delay_ms(SAMPLE_TIME);
306 }
307 LEDMOTOR_CLR;
308 }
309 void Motor_Stop(int Motors)
310 {
311     if(Motors & MOTOR_L)
312     {
313         ML_STANDBY;
314         Motor_Control.Left_State = STOP;
315         pwm_stop_channels((1 << MOTOR_L)); //Start PWM Channel on M0 line
316     }
317 }
```

```
320     if(Motors & MOTOR_R)
321     {
322         MR_STANDBY;
323         Motor_Control.Right_State = STOP;
324         pwm_stop_channels((1 << MOTOR_R));
325     }
326 }
327 int Motors_Move(int millimetres_fwd)//Move this amount forward in centimeters
328 {
329     //Calculate number of interrupts of each wheel
330     int number_interrupts;
331     int distance_moved;
332     if(millimetres_fwd > 0)
333     {
334         Motor_Control.Left_State = FORWARD;
335         Motor_Control.Right_State = FORWARD;
336     }
337     else
338     {
339         millimetres_fwd = Abs(millimetres_fwd);
340         Motor_Control.Left_State = BACKWARD;
341         Motor_Control.Right_State = BACKWARD;
342     }
343     number_interrupts = (millimetres_fwd * (int)GAMMA) / (int)
344         CIRCUMFERENCE_WHEEL_MM;
345     distance_moved = number_interrupts * MIN_DISTANCE_RESOLUTION;
346     print_dbg("\n\rNumber of interrupts to move = ");
347     print_dbg_ulong(number_interrupts);

348     Motor_Control.Left_Count = number_interrupts;
349     Motor_Control.Right_Count = number_interrupts;
350     Motor_Start(MOTOR_L | MOTOR_R);
351     Motors_Execute();
352     return distance_moved;
353 }

355 void Motors_Reset(void)
356 {
357     Motor_Control.Left_State = FORWARD;
358     Motor_Control.Left_Count = 1;
359     Motor_Control.Right_State = FORWARD;
360     Motor_Control.Right_Count = 1;
361     Motor_Start(MOTOR_L | MOTOR_R);
362     Motors_Execute();
363 }

365 bool Motors_Moving()
366 {
367     //  if(Motor_Control.Left_State != STOP)
368     //  {
369     //      if(Motor_Control.Right_State != STOP)
370     //      {
371     //          return true;
372     //      }
373     //      else
374     //          return false;
375     //  }
376     //  else
```

```

377 //  {
378 //    return false;
379 //}
380 if(Motor_Control.Left_State != STOP) //Left is moving
381 {
382   return true;
383 }
384 else if (Motor_Control.Right_State != STOP) //Right is moving
385 {
386   return true;
387 }
388 else
389 {
390   return false;
391 }
392 }

395 int Motors_Rotate(int angle_degs)
{
396   int interrupts_to_move = 0;
397   int angle_rotated;
398   //calculate interrupts to move
399   interrupts_to_move = angle_degs * ROTATION_CONST;
400   angle_rotated = interrupts_to_move * ROTATION_CONST_INV;

403   //Both Wheels Move
404   if(interrupts_to_move > 0)
405   {
406     Motor_Control.Left_State = FORWARD;
407     Motor_Control.Right_State = BACKWARD;
408   }
409   else
410   {
411     Motor_Control.Right_State = FORWARD;
412     Motor_Control.Left_State = BACKWARD;
413   }
414   Motor_Control.Left_Count = Abs(interrupts_to_move);
415   Motor_Control.Right_Count = Abs(interrupts_to_move);
416   Motor_Start(MOTOR_L | MOTOR_R);
417   Motors_Execute();
418   return angle_rotated;
419 }

420 }

```

H.1.1.8 OV7670.h

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

```

1 /*
2 * OV7670.h
3 *
4 * Created: 15/02/2013 13:12:00
5 * Author: hslovett

```

```
6  */

9 #ifndef OV7670_H_
10 #define OV7670_H_
11 #include <asf.h>
12 ///////////////////////////////////////////////////////////////////
13 // Constants
14 ///////////////////////////////////////////////////////////////////
15 #define HEIGHT          240
16 #define WIDTH           320
17 #define PIXELSIZE       2
18 #define SETTINGS_LENGTH 167
19 #define OV7670_ADDR     0x21

21 #define CAMERA_LEFT      1
22 #define CAMERA_RIGHT     2

24 #define CAMERA_LEFT_ERR   0x10
25 #define CAMERA_RIGHT_ERR  0x20

27 #define BMPHEADERSIZE 14
28 #define DIBHEADERSIZE 124 //v5
29 #define FILESIZE 153738
30 ///////////////////////////////////////////////////////////////////
31 // Globals
32 ///////////////////////////////////////////////////////////////////
33 const char default_settings[SETTINGS_LENGTH][2];
34 const uint8_t DIBHead[DIBHEADERSIZE];
35 const uint8_t BMPHeader[BMPHEADERSIZE];
36 typedef struct {
37     uint8_t Status;
38     bool Camera_0_Found;
39     bool Camera_1_Found;
40     bool Camera_0_Error;
41     bool Camera_1_Error;
42     uint8_t VSYNC0_State;
43     uint8_t VSYNC1_State;
44 } OV7670_t;

46 OV7670_t OV7670_Status;

48 #define IDLE          0
49 #define TAKE_PHOTO    1
50 #define TAKING_PHOTO  2
51 #define TAKEN_PHOTO   3
52 #define CAMERAS_BUSY  4

54 #define Image0Name  "Image_L_%d.bmp"
55 #define Image1Name  "Image_R_%d.bmp"
56 ///////////////////////////////////////////////////////////////////
57 // Methods
58 ///////////////////////////////////////////////////////////////////
59 void OV7670_Init(void);           //Initialises Camera
60 void FIFO_Init();
61 int TakePhoto(uint8_t Cameras);
62 bool Photos_Ready(void);
63 void Store_Image_0();
```

```

64 void Store_Image_1();
65 void FIFO_Reset(uint8_t CameraID);
66 bool Store_Both_Images();
67 //void FIFO_Reset(uint8_t CameraID);
68 ///////////////////////////////////////////////////////////////////
69 // Pins & Macros
70 ///////////////////////////////////////////////////////////////////
71 #define FIFO_0_RCLK AVR32_PIN_PA19
72 #define FIFO_0_nRRST AVR32_PIN_PA15
73 #define FIFO_0_WEN AVR32_PIN_PA20
74 #define FIFO_0_WRST AVR32_PIN_PA21
75 #define FIFO_0_nOE AVR32_PIN_PA28
76 #define FIFO_0_VSYNC AVR32_PIN_PA29

78 #define FIFO_1_RCLK AVR32_PIN_PA23
79 #define FIFO_1_nRRST AVR32_PIN_PA22
80 #define FIFO_1_WEN AVR32_PIN_PA24
81 #define FIFO_1_WRST AVR32_PIN_PA25
82 #define FIFO_1_nOE AVR32_PIN_PA27

84 #define VSYNC_1_PIN AVR32_EIC_EXTINT_1_2_PIN
85 #define VSYNC_1_FUNCTION AVR32_EIC_EXTINT_1_2_FUNCTION
86 #define VSYNC_1_LINE 1
87 #define VSYNC_1_ENABLE_INTERRUPT {eic_enable_interrupt_line(&AVR32_EIC,
88 VSYNC_1_LINE);}
88 #define VSYNC_1_DISABLE_INTERRUPT {eic_disable_interrupt_line(&AVR32_EIC,
89 VSYNC_1_LINE);}

90 #define VSYNC_0_PIN AVR32_EIC_EXTINT_4_0_PIN
91 #define VSYNC_0_FUNCTION AVR32_EIC_EXTINT_4_0_FUNCTION
92 #define VSYNC_0_LINE 4
93 #define VSYNC_0_ENABLE_INTERRUPT {eic_enable_interrupt_line(&AVR32_EIC,
94 VSYNC_0_LINE);}
94 #define VSYNC_0_DISABLE_INTERRUPT {eic_disable_interrupt_line(&AVR32_EIC,
95 VSYNC_0_LINE);}

97 #define FIFO_0_RCLK_SET {gpio_set_gpio_pin(FIFO_0_RCLK);}
98 #define FIFO_0_nRRST_SET {gpio_set_gpio_pin(FIFO_0_nRRST);}
99 #define FIFO_0_WEN_SET {gpio_set_gpio_pin(FIFO_0_WEN);}
100 #define FIFO_0_WRST_SET {gpio_set_gpio_pin(FIFO_0_WRST);}
101 #define FIFO_0_nOE_SET {gpio_set_gpio_pin(FIFO_0_nOE);}

103 #define FIFO_0_RCLK_CLR {gpio_clr_gpio_pin(FIFO_0_RCLK);}
104 #define FIFO_0_nRRST_CLR {gpio_clr_gpio_pin(FIFO_0_nRRST);}
105 #define FIFO_0_WEN_CLR {gpio_clr_gpio_pin(FIFO_0_WEN);}
106 #define FIFO_0_WRST_CLR {gpio_clr_gpio_pin(FIFO_0_WRST);}
107 #define FIFO_0_nOE_CLR {gpio_clr_gpio_pin(FIFO_0_nOE);}

111 #define FIFO_1_RCLK_SET {gpio_set_gpio_pin(FIFO_1_RCLK);}
112 #define FIFO_1_nRRST_SET {gpio_set_gpio_pin(FIFO_1_nRRST);}
113 #define FIFO_1_WEN_SET {gpio_set_gpio_pin(FIFO_1_WEN);}
114 #define FIFO_1_WRST_SET {gpio_set_gpio_pin(FIFO_1_WRST);}
115 #define FIFO_1_nOE_SET {gpio_set_gpio_pin(FIFO_1_nOE);}

117 #define FIFO_1_RCLK_CLR {gpio_clr_gpio_pin(FIFO_1_RCLK);}

```

```

118 #define FIFO_1_nRRST_CLR {gpio_clr_gpio_pin(FIFO_1_nRRST);}
119 #define FIFO_1_WEN_CLR {gpio_clr_gpio_pin(FIFO_1_WEN);}
120 #define FIFO_1_WRST_CLR {gpio_clr_gpio_pin(FIFO_1_WRST);}
121 #define FIFO_1_nOE_CLR {gpio_clr_gpio_pin(FIFO_1_nOE);}

124 #define CAMERA_INPUT {(uint8_t)((AVR32_GPIO.port[1].pvr) & 0xFF);}
125 //////////////////////////////////////////////////////////////////
126 //Camera Register Address definitions
127 //////////////////////////////////////////////////////////////////
128 #define OV_GAIN 0x00 //Gain Control Setting - ACG[7:0]
129 #define OV_BLUE 0x01 //Blue Channel Gain
130 #define OV_RED 0x02 //Red Channel Gain
131 #define OV_VREF 0x03 //Vertical Frame Control & ACG[9:8]
132 #define OV_COM1 0x04 //CCIR656 enable, AEC low bits (AECHH, AECH)
133 #define OV_BAVE 0x05 //U/B Average level - AUTO UPDATED
134 #define OV_GbAVE 0x06 //Y/Gb Average Level - AUTO UPDATED
135 #define OV_AECHH 0x07 //Exposure value [15:10] (AECH, COM1)
136 #define OV_RAVE 0x08 //V/R Average level - AUTO UPDATED
137 #define OV_COM2 0x09 //Soft Sleep, Output drive capability
138 #define OV_PID 0x0A //Product ID MSB Read only
139 #define OV_VER 0x0B //Product ID LSB Read Only
140 #define OV_COM3 0x0C //Output data MSB/LSB swap + other stuff
141 #define OV_COM4 0x0D //Average values - MUST BE SAME AS COM1
142 #define OV_COM5 0x0E //RESERVED
143 #define OV_COM6 0x0F //COM6
144 #define OV_AECH 0x10 //Exposure value [9:2] (see AECHH, COM1)
145 #define OV_CLKRC 0x11 //Internal Clock options
146 #define OV_COM7 0x12 //RESET, Output format
147 #define OV_COM8 0x13 //Common control 8
148 #define OV_COM9 0x14 //Automatic Gain Ceiling
149 #define OV_COM10 0x15 //PCLK, HREF and VSYNC options
150 #define OV_RSVD 0x16 //RESERVED
151 #define OV_HSTART 0x17 //Output format Horizontal Frame start
152 #define OV_HSTOP 0x18 //Output format Horizontal Frame end
153 #define OV_VSTRT 0x19 //Output format Vertical Frame start
154 #define OV_VSTOP 0x1A //Output format Vertical Frame Stop
155 #define OV_PSHFT 0x1B //Pixel Delay Select
156 #define OV_MIDH 0x1C //Manufacturer ID MSB - READ ONLY
157 #define OV_MIDL 0x1D //Manufacturer ID LSB - READ ONLY
158 #define OV_MVFP 0x1E //Mirror / Vflip Enable
159 #define OV_LAEC 0x1F //RESERVED
160 #define OV_ADCCTR0 0x20 //ADC Control
161 #define OV_ADCCTR1 0x21 //RESERVED
162 #define OV_ADCCTR2 0x22 //RESERVED
163 #define OV_ADCCTR3 0x23 //RESERVED
164 #define OV_AEW 0x24 //ACG/AEC Stable Operating Region Upper Limit
165 #define OV_AEB 0x25 //ACG/AEC Stable Operation Region Lower Limit
166 #define OV_VPT 0x26 //ACG/AEC Fast Mode Operation Region
167 #define OV_BBIAIS 0x27 //B Channel Signal Output Bias
168 #define OV_GbBIAS 0x28 //Gb Channel Output Bias
169 #define OV_RSVD1 0x29 //RESERVED
170 #define OV_EXHCH 0x2A //Dummy Pixel Insert MSB
171 #define OV_EXHCL 0x2B //Dummy Pixel Insert LSB
172 #define OV_RBIAS 0x2C //R Channel Signal Output Bias
173 #define OV_ADVFL 0x2D //LSB of insert dummy line in vertical direction
174 #define OV_ADVFH 0x2E //MSB of insert dummy line in vertical direction
175 #define OV_YAVE 0x2F //Y/G Channel Average Value

```

```
176 #define OV_HSYST      0x30 //HSYNC Rising Edge Delay (low 8 bits)
177 #define OV_HSYEN      0x31 //HSYNC Falling Edge Delay (low 8 bits)
178 #define OV_HREF       0x32 //HREF Control
179 #define OV_CHLF       0x33 //Array Current Control - RESERVED
180 #define OV_ARBLM      0x34 //Array Reference Control - RESERVED
181 #define OV_RSVD2      0x35 //RESERVED
182 #define OV_RSVD3      0x36 //RESERVED
183 #define OV_ADCCTRL     0x37 //ADC Control - RESERVED
184 #define OV_ACOM        0x38 //ADC and Analog Common Mode Control - RESERVED
185 #define OV_OFON        0x39 //ADC Offset Control
186 #define OV_TSLB        0x3A //Line Buffer Test Option
187 #define OV_COM11       0x3B //COM11
188 #define OV_COM12       0x3C //COM12
189 #define OV_COM13       0x3D //COM13
190 #define OV_COM14       0x3E //COM14
191 #define OV_EDGE         0x3F //Edge Detection Adjustment
192 #define OV_COM15       0x40 //COM15
193 #define OV_COM16       0x41 //COM16
194 #define OV_COM17       0x42 //COM17
195 #define OV_AWBC1        0x43
196 #define OV_AWBC2        0x44
197 #define OV_AWBC3        0x45
198 #define OV_AWBC4        0x46
199 #define OV_AWBC5        0x47
200 #define OV_AWBC6        0x48
201 #define OV_RSVD4        0x49
202 #define OV_RSVD5        0x40
203 #define OV_RSVD6        0x4A
204 #define OV_REG4B        0x4B
205 #define OV_DNSTH        0x4C
206 #define OV_RSVD7        0x4D
207 #define OV_RSVD8        0x4E
208 #define OV_MTX1         0x4F
209 #define OV_MTX2         0x50
210 #define OV_MTX3         0x51
211 #define OV_MTX4         0x52
212 #define OV_MTX5         0x53
213 #define OV_MTX6         0x54
214 #define OV_BRIGHT        0x55
215 #define OV CONTRAS      0x56
216 #define OV CONTRASCNTR   0x57
217 #define OV_MTXS         0x58
218 #define OV_RSVD9        0x59
219 #define OV_RSVD9_1       0x5A
220 #define OV_RSVD9_2       0x5B
221 #define OV_RSVD9_3       0x5C
222 #define OV_RSVD9_4       0x5D
223 #define OV_RSVD9_5       0x5E
224 #define OV_RSVD9_6       0x5F
225 #define OV_RSVD10       0x60
226 #define OV_RSVD11       0x61
227 #define OV_LCC1          0x62
228 #define OV_LCC2          0x63
229 #define OV_LCC3          0x64
230 #define OV_LCC4          0x65
231 #define OV_LCC5          0x66
232 #define OV_MANU          0x67
233 #define OV_MANV          0x68
```

```
234 #define OV_GFIX      0x69
235 #define OV_GGAIN     0x6A
236 #define OV_DBLV      0x6B
237 #define OV_AWBCTR3   0x6C
238 #define OV_AWBCTR2   0x6D
239 #define OV_AWBCTR1   0x6E
240 #define OV_AWBCTR0   0x6F
241 #define OV_SCALING_XSC 0x70
242 #define OV_SCALING_YSC 0x71
243 #define OV_SCALING_DCWCTR 0x72
244 #define OV_SCALING_PCLK_DIV 0x73
245 #define OV_REG74      0x74
246 #define OV_REG75      0x75
247 #define OV_REG76      0x76
248 #define OV_REG77      0x77
249 #define OV_RSVD12    0x78
250 #define OV_RSVD13    0x79
251 #define OV_GAM1       0x7A
252 #define OV_GAM2       0x7B
253 #define OV_GAM3       0x7C
254 #define OV_GAM4       0x7D
255 #define OV_GAM5       0x7E
256 #define OV_GAM6       0x7F
257 #define OV_GAM7       0x80
258 #define OV_GAM8       0x81
259 #define OV_GAM9       0x82
260 #define OV_GAM10      0x83
261 #define OV_GAM11      0x84
262 #define OV_GAM12      0x85
263 #define OV_GAM13      0x86
264 #define OV_GAM14      0x87
265 #define OV_GAM15      0x88
266 #define OV_GAM16      0x89
267 #define OV_RSVD14    0x8A
268 #define OV_RSVD15    0x8B
269 #define OV_RSVD16    0x8C
270 #define OV_RSVD17    0x8D
271 #define OV_RSVD18    0x8E
272 #define OV_RSVD19    0x8F
273 #define OV_RSVD20    0x90
274 #define OV_RSVD21    0x91
275 #define OV_DM_LNL     0x92
276 #define OV_DM_LNH     0x93
277 #define OV_LCC6       0x94
278 #define OV_LCC7       0x95
279 #define OV_RSVD22    0x96
280 #define OV_RSVD23    0x97
281 #define OV_RSVD24    0x98
282 #define OV_RSVD25    0x99
283 #define OV_RSVD26    0x9A
284 #define OV_RSVD27    0x9B
285 #define OV_RSVD28    0x9C
286 #define OV_BD50ST    0x9D
287 #define OV_BD60ST    0x9E
288 #define OV_HISTO     0x9F
289 #define OV_HIST1     0xA0
290 #define OV_HIST2     0xA1
291 #define OV_HIST3     0xA2
```

```

292 #define OV_HIST4      0xA3
293 #define OV_HIST5      0xA4
294 #define OV_HIST6      0xA5
295 #define OV_HIST7      0xA6
296 #define OV_HIST8      0xA7
297 #define OV_HIST9      0xA8
298 #define OV_HIST10     0xA9
299 #define OV_HIST11     0xAA
300 #define OV_HIST12     0xAB
301 #define OV_STR_OPT    0xAC
302 #define OV_STR_R       0xAD
303 #define OV_STR_G       0xAE
304 #define OV_STR_B       0xAF
305 #define OV_RSVD28_1    0xB0
306 #define OV_RSVD29    0xB1
307 #define OV_RSVD30    0xB2
308 #define OV_THL_ST     0xB3
309 #define OV_RSVD31    0xB4
310 #define OV_THL_DLT    0xB5
311 #define OV_RSVD32    0xB6
312 #define OV_RSVD33    0xB7
313 #define OV_RSVD34    0xB8
314 #define OV_RSVD35    0xB9
315 #define OV_RSVD36    0xBA
316 #define OV_RSVD37    0xBB
317 #define OV_RSVD38    0xBC
318 #define OV_RSVD39    0xBD
319 #define OV_AD_CHB     0xBE
320 #define OV_AD_CHR     0xBF
321 #define OV_AD_CHGb    0xC0
322 #define OV_AD_CHGr    0xC1
323 #define OV_RSVD40    0xC2
324 #define OV_RSVD41    0xC3
325 #define OV_RSVD42    0xC4
326 #define OV_RSVD43    0xC5
327 #define OV_RSVD44    0xC6
328 #define OV_RSVD45    0xC7
329 #define OV_RSVD46    0xC8
330 #define OV_SATCTR   0xC9

334 #endif /* OV7670_H_ */

```

H.1.1.9 OV7670.c

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

```

1 /*
2  * OV7670.c
3  *
4  * Created: 15/02/2013 13:12:12
5  * Author: hslovett
6  */

```

```
9 #include <asf.h>
10 #include "CustomDevices/CustomDevices.h"
11 #include "stdio.h"
12 #include "delay.h"
13 // Camera
14 // #include "CustomDevices/0V7670.h"
15 // I2C Mux
16 // #include "CustomDevices/PCA9542A.h"
17 // MotorDriver
18 // /*#include "CustomDevices/MotorDriver.h"*/
19 // SDCard
20 // #include "CustomDevices/SD_Card.h"

22 __attribute__((__interrupt__)) static void VSYNC0_Handler (void)
23 {
24     //print_dbg("\n\rVSYNC0 Detected!");
25     eic_clear_interrupt_line(&AVR32_EIC, VSYNC_0_LINE);
26     //VSYNC_0_DISABLE_INTERRUPT;
27     switch(OV7670_Status.VSYNC0_State)
28     {
29         case(TAKE_PHOTO):
30             FIFO_0_WEN_SET;
31             OV7670_Status.VSYNC0_State = TAKING_PHOTO;
32             break;
33
34         case(TAKING_PHOTO):
35             FIFO_0_WEN_CLR;
36             OV7670_Status.VSYNC0_State = TAKEN_PHOTO;
37             break;
38
39         case (TAKEN_PHOTO):
40             FIFO_0_WEN_CLR;
41             break;
42
43         case(IDLE):
44     default:
45             VSYNC_0_DISABLE_INTERRUPT;
46             FIFO_0_WEN_CLR;
47             OV7670_Status.VSYNC0_State = IDLE;
48             break;
49     }
50 }

52 __attribute__((__interrupt__)) static void VSYNC1_Handler (void)
53 {
54     //print_dbg("\n\rVSYNC1 Detected!");
55     eic_clear_interrupt_line(&AVR32_EIC, VSYNC_1_LINE);
56     //VSYNC_1_DISABLE_INTERRUPT;
57     switch(OV7670_Status.VSYNC1_State)
58     {
59         case(TAKE_PHOTO):
60             FIFO_1_WEN_SET;
61             OV7670_Status.VSYNC1_State = TAKING_PHOTO;
62             //print_dbg("\n\rCase: Take Photo;");
63             break;
```

```
65     case(TAKING_PHOTO):
66     FIFO_1_WEN_CLR;
67     OV7670_Status.VSYNC1_State = TAKEN_PHOTO;
68     //print_dbg("\n\rCase: Taking Photo;");
69     break;

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

76     case(IDLE):
77     default:
78     VSYNC_1_DISABLE_INTERRUPT;
79     FIFO_1_WEN_CLR;
80     OV7670_Status.VSYNC1_State = IDLE;
81     //print_dbg("\n\rCase: Idle;");
82     break;
83   }
84 }
85 unsigned char Write_Reg(unsigned char Register, unsigned char Data)
86 {
87   /* I2C Traffic Generated:
88   * S | OV_7670 + W | A | RegID | A | Data | A | P |
89   */
90   uint8_t Buff[2] = {Register, Data};
91   int status = twim_write(&AVR32_TWIMO, &Buff, 2, OV7670_ADDR, false);
92   return status;
93 }
94 unsigned char Read_Reg(unsigned char Register, unsigned char *Data)
95 {
96   /* I2C Traffic Generated:
97   * S | OV_ADDR + W | A | RegID | A | P |
98   * S | OV_ADDR + R | A | Data | ~A | P |
99   */
100  unsigned char Buff[2] = {Register, 0};
101  int status = twim_write(&AVR32_TWIMO, &Buff, 1, OV7670_ADDR, false);
102  if(status != STATUS_OK)
103    return status;

105  status = twim_read(&AVR32_TWIMO, &Buff, 1, OV7670_ADDR, false);
106  *Data = Buff[0];

108  return status;
109 }
110 void OV7670_Init()
111 {
112

114   //Check Cameras Exist
115   PCA9542A_Chан_Sel(I2C_CHANNEL_0);
116   if (twim_probe(&AVR32_TWIMO, OV7670_ADDR) == STATUS_OK)
117     OV7670_Status.Camera_0_Found = true;
118   else
119     OV7670_Status.Camera_0_Found = false;

121   PCA9542A_Chан_Sel(I2C_CHANNEL_1);
122   if (twim_probe(&AVR32_TWIMO, OV7670_ADDR) == STATUS_OK)
```

```
123     OV7670_Status.Camera_1_Found = true;
124
125     OV7670_Status.Camera_1_Found = false;
126
127
128 //Initialise Cameras
129 if(OV7670_Status.Camera_0_Found)
130 {
131     PCA9542A_Chain_Sel(I2C_CHANNEL_0);
132 //Reset Camera
133 if(STATUS_OK != Write_Reg(OV_COM7, 0x80))
134 {
135     print_dbg("\n\rCamera Reset Fail");
136     OV7670_Status.Camera_0_Error = true;
137     OV7670_Status.Status = ERR_DEVICE;
138     //return FAIL;
139 }
140 delay_ms(10); //wait for Camera to reset
141 for (int i = 0; i < SETTINGS_LENGTH; i++)
142 {
143     if(STATUS_OK != Write_Reg(default_settings[i][0], default_settings[i]
144 [1]))
145     {
146         print_dbg("\n\rCamera Initialise Fail");
147         //return FAIL;
148         OV7670_Status.Camera_0_Error = true;
149         OV7670_Status.Status = ERR_DEVICE;
150         break;
151     }
152     delay_ms(1);
153 }
154 else
155 {
156     OV7670_Status.Status |= CAM0_NotFOUND;
157 }
158 if(OV7670_Status.Camera_1_Found)
159 {
160     PCA9542A_Chain_Sel(I2C_CHANNEL_1);
161
162 //Reset Camera
163 if(STATUS_OK != Write_Reg(OV_COM7, 0x80))
164 {
165     print_dbg("\n\rCamera Reset Fail");
166     OV7670_Status.Camera_1_Error = true;
167     OV7670_Status.Status = ERR_DEVICE;
168     //return FAIL;
169 }
170 delay_ms(10); //wait for Camera to reset
171 for (int i = 0; i < SETTINGS_LENGTH; i++)
172 {
173     if(STATUS_OK != Write_Reg(default_settings[i][0], default_settings[i]
174 [1]))
175     {
176         print_dbg("\n\rCamera Initialise Fail");
177         //return FAIL;
178         OV7670_Status.Camera_1_Error = true;
179         OV7670_Status.Status = ERR_DEVICE;
```

```
179         break;
180     }
181     delay_ms(1);
182 }
183 }
184 else
185 {
186     OV7670_Status.Status |= CAM1_NotFOUND;
187 }
188 PCA9542A_Chain_Sel(NO_SELECT);

189 //Initialise VSYNC Interrupts
190 eic_options_t eic_options;
191 eic_options.eic_mode = EIC_MODE_EDGE_TRIGGERED;
192 eic_options.eic_edge = EIC_EDGE_FALLING_EDGE;
193 eic_options.eic_async = EIC_SYNCH_MODE;
194 eic_options.eic_line = VSYNC_1_LINE;
195 //eic_options.eic_line = VSYNC_0_LINE;

196 Disable_global_interrupt();
197 gpio_enable_module_pin(VSYNC_1_PIN, VSYNC_1_FUNCTION);
198 gpio_enable_module_pin(VSYNC_0_PIN, VSYNC_0_FUNCTION);

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

206 INTC_register_interrupt(&VSYNC1_Handler, AVR32_EIC_IRQ_1, AVR32_INTC_INT0);
207 INTC_register_interrupt(&VSYNC0_Handler, AVR32_EIC_IRQ_4, AVR32_INTC_INT0);
208 //Enable interrupt on VSYNC1
209 eic_enable_line(&AVR32_EIC, VSYNC_1_LINE);
210 eic_enable_line(&AVR32_EIC, (VSYNC_0_LINE));
211 VSYNC_1_ENABLE_INTERRUPT;
212 VSYNC_0_ENABLE_INTERRUPT;

213 FIFO_Init();
214 Enable_global_interrupt();

215 }
216 void FIFO_Init()
217 {
218     //Disable both outputs
219     FIFO_0_nOE_SET
220     FIFO_1_nOE_SET

221     //Reset Buffer 0
222     FIFO_0_WRST_CLR;
223     FIFO_0_RCLK_CLR;
224     FIFO_0_nRRST_SET;
225     FIFO_0_WEN_CLR;
226     delay_us(10);
227     FIFO_0_RCLK_SET;
228     delay_us(10);
229     FIFO_0_RCLK_CLR;
```

```
236     FIFO_0_nRRST_CLR;
237     delay_us(10);
238     FIFO_0_RCLK_SET;
239     delay_us(10);
240     FIFO_0_RCLK_CLR;
241     FIFO_0_nRRST_SET;
242     delay_us(10);
243     FIFO_0_WRST_SET;

244     //Reset Buffer 1
245     FIFO_1_WRST_CLR;
246     FIFO_1_RCLK_CLR;
247     FIFO_1_nRRST_SET;
248     FIFO_1_WEN_CLR;
249     delay_us(10);
250     FIFO_1_RCLK_SET;
251     delay_us(10);
252     FIFO_0_RCLK_CLR;
253     FIFO_1_nRRST_CLR;
254     delay_us(10);
255     FIFO_1_RCLK_SET;
256     delay_us(10);
257     FIFO_1_RCLK_CLR;
258     FIFO_1_nRRST_SET;
259     delay_us(10);
260     FIFO_1_WRST_SET;
261 }
262

263 void FIFO_Reset(uint8_t CameraID)
264 {
265     FIFO_0_nOE_SET;
266     FIFO_1_nOE_SET;
267     if(CameraID & CAMERA_LEFT)
268     {
269         FIFO_0_WRST_CLR;
270         FIFO_0_nRRST_CLR;
271         FIFO_0_RCLK_SET;
272         delay_us(10);
273         FIFO_0_RCLK_CLR;
274         FIFO_0_nRRST_SET;
275         FIFO_0_WRST_SET;
276     }
277     if(CameraID & CAMERA_RIGHT)
278     {
279         FIFO_1_WRST_CLR;
280         FIFO_1_nRRST_CLR;
281         FIFO_1_RCLK_SET;
282         delay_us(10);
283         FIFO_1_RCLK_CLR;
284         FIFO_1_nRRST_SET;
285         FIFO_1_WRST_SET;
286     }
287 }

288 }

289 int TakePhoto(uint8_t Cameras)
290 {
```

```
294 //Only want to take pictures on cameras found
295 if((OV7670_Status.VSYNC0_State != IDLE) || !OV7670_Status.Camera_0_Found)
296     && ((OV7670_Status.VSYNC1_State != IDLE) || !OV7670_Status.Camera_1_Found)
297 )
298     return CAMERAS_BUSY; //wait for cameras to be idle if they are found
299
300 if(Cameras & CAMERA_LEFT)
301     OV7670_Status.VSYNC0_State = TAKE_PHOTO;
302
303 if(Cameras & CAMERA_RIGHT)
304     OV7670_Status.VSYNC1_State = TAKE_PHOTO;
305 eic_clear_interrupt_line(&AVR32_EIC, VSYNC_1_LINE);
306 eic_clear_interrupt_line(&AVR32_EIC, VSYNC_0_LINE);
307 VSYNC_0_ENABLE_INTERRUPT;
308 VSYNC_1_ENABLE_INTERRUPT;
309
310 return TAKING_PHOTO;
311 }
312
313 bool Photos_Ready(void)
314 {
315     int status = 0;
316     if(OV7670_Status.Camera_0_Found == true) //If camera is there
317     {
318         if(OV7670_Status.Camera_0_Error == false)//and has no errors
319         {
320             if(OV7670_Status.VSYNC0_State == TAKEN_PHOTO)
321             {
322                 status |= 1; //camera0 has taken photo
323             }
324             else
325                 status |= 1;
326         }
327         else
328             status |= 1;
329
330         if(OV7670_Status.Camera_1_Found == true) //If camera is there
331         {
332             if(OV7670_Status.Camera_1_Error == false)//and has no errors
333             {
334                 if(OV7670_Status.VSYNC1_State == TAKEN_PHOTO)
335                 {
336                     status |= 1; //camera0 has taken photo
337                 }
338             }
339             else
340                 status |= 1;
341         }
342         else
343             status |= 1;
344
345         if(status)
346             return true;
347         else
348             return false;
349 }
```

```
353     bool Store_Both_Images()
354     {
355         if(Photos_Ready() == false)
356             return false;
357
358         Store_Image_1();
359         FIFO_Reset(CAMERA_RIGHT);
360
361         Store_Image_0();
362         FIFO_Reset(CAMERA_LEFT);
363
364         OV7670_Status.VSYNC0_State = IDLE;
365         OV7670_Status.VSYNC1_State = IDLE;
366
367         return true;
368     }
369
370 // void Store_Image_0()
371 // {
372 //     int i,j;
373 //     //Image0
374 //     //make file
375 //     //delete file if it exists already
376 //     char Filename_buff[15];
377 //     i = 0;
378 //     while(1)
379 //     {
380 //         nav_filelist_reset();
381 //         sprintf(&Filename_buff, Image0Name, i++);
382 //         if(nav_filelist_findname((FS_STRING)Filename_buff, false))
383 //         {
384 //             //nav_setcwd((FS_STRING)Image0Name, true, false);
385 //             print_dbg("\n\r File Exists");
386 //             print_dbg(&Filename_buff);
387 //             //nav_file_del(false);
388 //         }
389 //         else
390 //         {
391 //             break;
392 //         }
393 //     }
394 //     nav_file_create((FS_STRING)Filename_buff); //create file
395 //     file_open(FOPEN_MODE_W);
396 //     //write bitmap headers
397 //     file_write_buf(BMPHeader, BMPHEADERSIZE);
398 //     file_write_buf(DIBHead, DIBHEADERSIZE);
399 //     //read and write image data
400 //     //Image0
401 //     //reset read pointer
402 //     FIFO_0_nRRST_CLR;
403 //     FIFO_0_RCLK_SET;
404 //     //
405 //     FIFO_0_RCLK_CLR;
406 //     FIFO_0_nRRST_SET;
407 //     delay_us(10);
```

```
408 //    //enable output
409 //    FIFO_0_nOE_CLR;
410 //    uint8_t buffer[WIDTH * 2];
411 //
412 //    for(j = 0; j < HEIGHT; j++)
413 //    {
414 //        for(i = 0; i < WIDTH*2; i+=2)
415 //        {
416 //            FIFO_0_RCLK_SET;
417 //            delay_us(10);
418 //            buffer[i+1] = ((AVR32_GPIO.port[1].pvr) & 0xFF); //CAMERA_INPUT;
419 //            delay_us(10);
420 //            FIFO_0_RCLK_CLR;
421 //            delay_us(10);
422 //            FIFO_0_RCLK_SET;
423 //            delay_us(10);
424 //            buffer[i] = ((AVR32_GPIO.port[1].pvr) & 0xFF); //CAMERA_INPUT;
425 //            delay_us(10);
426 //            FIFO_0_RCLK_CLR;
427 //            delay_us(10);
428 //        }
429 //        file_write_buf(&buffer, WIDTH * 2);
430 //    }
431 //    FIFO_0_nOE_SET;
432 //    file_close();
433 //
434 //
435 // }

437 // void Store_Image_1()
438 // {
439 //     int i, j;
440 //     uint8_t buffer[WIDTH * 2];
441 //     char Filename_buff[15];
442 //     //uint8_t *Buffer_ram;
443 //     //Buffer_ram = mspace_malloc(sdram_msp, WIDTH * 2);
444 //     //if(Buffer_ram == NULL)
445 //     //{
446 //         //print_dbg("\n\rBuffer allocation fail.\n\r");
447 //         //return;
448 //     //}
449 //     i = 0;
450 //     //make file
451 //     //delete file if it exits already
452 //     nav_filelist_reset();
453 //     while(1)
454 //     {
455 //         sprintf(&Filename_buff, Image1Name, i++);
456 //         if(nav_filelist_findname((FS_STRING)Filename_buff, false))
457 //         {
458 //             //nav_setcwd((FS_STRING)Image1Name, true, false);
459 //             //print_dbg("\n\rImage1.bmp File Exists");
460 //             //nav_file_del(false);
461 //         }
462 //         else
463 //         {
464 //             break;
465 //         }
```

```
466 // }
467 // nav_file_create((FS_STRING)Filename_buff); //create file
468 // file_open(FOPEN_MODE_W);
469 // //write bitmap headers
470 // file_write_buf(BMPHeader, BMPHEADERSIZE);
471 // file_write_buf(DIBHead, DIBHEADERSIZE);
472 // //Image1
473 // //reset read pointer
474 // FIFO_1_nRRST_CLR;
475 //
476 // FIFO_1_RCLK_SET;
477 // delay_us(10);
478 // FIFO_1_RCLK_CLR;
479 // FIFO_1_nRRST_SET;
480 //
481 // //enable output
482 // FIFO_1_nOE_CLR;
483 // // uint8_t buffer[WIDTH * 2];
484 //
485 // for(j = 0; j < HEIGHT; j++)
486 //
487 // for(i = 0; i < WIDTH*2; i+=2)
488 //
489 // FIFO_1_RCLK_SET;
490 // delay_us(10);
491 // buffer[i+1] = ((AVR32_GPIO.port[1].pvr) & 0xFF); //CAMERA_INPUT;
492 // delay_us(10);
493 // FIFO_1_RCLK_CLR;
494 // delay_us(10);
495 // FIFO_1_RCLK_SET;
496 // delay_us(10);
497 // buffer[i] = ((AVR32_GPIO.port[1].pvr) & 0xFF); //CAMERA_INPUT;
498 // delay_us(10);
499 // FIFO_1_RCLK_CLR;
500 // delay_us(10);
501 //
502 // file_write_buf(&buffer, WIDTH * 2);
503 //
504 //
505 // FIFO_1_nOE_SET; // disable output
506 // file_close();
507 // //mspace_free(sdram_msp, Buffer_ram);
508 // /* mspace_free(sdram_msp, Buffer_ram); */
509 //

510 void Store_Image_1()
511 {
512     int i, j;
513     //uint8_t buffer[WIDTH * 2];
514     char Filename_buff[15];
515     uint8_t *Buffer_ram;
516     Buffer_ram = mspace_malloc(sdram_msp, HEIGHT * WIDTH * 2);
517     i = 0;
518     //make file
519     //delete file if it exists already
520     nav_filelist_reset();
521     while(1)
522     {
```

```
524     sprintf(&Filename_buff, Image1Name, i++);
525     if(nav_filelist_findname((FS_STRING)Filename_buff, false))
526     {
527         ;
528     }
529     else
530     {
531         break;
532     }
533 }

535 //Image1
536 //reset read pointer
537 FIFO_1_nRRST_CLR;

539 FIFO_1_RCLK_SET;
540 delay_us(10);
541 FIFO_1_RCLK_CLR;
542 FIFO_1_nRRST_SET;

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

548 for(j = 0; j < HEIGHT * WIDTH * 2; j+= 2)
549 {
550     FIFO_1_RCLK_SET;
551     delay_us(10);
552     Buffer_ram[j+1] = ((AVR32_GPIO.port[1].pvr) & 0xFF); //CAMERA_INPUT;
553     delay_us(10);
554     FIFO_1_RCLK_CLR;
555     delay_us(10);
556     FIFO_1_RCLK_SET;
557     delay_us(10);
558     Buffer_ram[j] = ((AVR32_GPIO.port[1].pvr) & 0xFF); //CAMERA_INPUT;
559     delay_us(10);
560     FIFO_1_RCLK_CLR;
561     delay_us(10);
562 }

564 FIFO_1_nOE_SET;//disable output
565 /* file_close();*/
566 SaveBitmap(Buffer_ram, WIDTH, HEIGHT, Filename_buff);
567 mspace_free(sdram_msp, Buffer_ram);
568 }

570 void Store_Image_0()
571 {
572     int i, j;
573     //uint8_t buffer[WIDTH * 2];
574     char Filename_buff[15];
575     uint16_t *Buffer_ram;
576     Buffer_ram = mspace_malloc(sdram_msp, HEIGHT * WIDTH );
577     i = 0;
578     //make file
579     //delete file if it exits already
580     nav_filelist_reset();
581     while(1)
```

```

582 {
583     sprintf(&Filename_buff, ImageOName, i++);
584     if(nav_filelist_findname((FS_STRING)Filename_buff, false))
585     {
586         ;
587     }
588     else
589     {
590         break;
591     }
592 }

594 //Image1
595 //reset read pointer
596 FIFO_O_nRRST_CLR;

598 FIFO_O_RCLK_SET;
599 delay_us(10);
600 FIFO_O_RCLK_CLR;
601 FIFO_O_nRRST_SET;

603 //enable output
604 FIFO_O_nOE_CLR;
605 // uint8_t buffer[WIDTH * 2];

607 for(j = 0; j < HEIGHT * WIDTH; j++)
608 {
609     FIFO_O_RCLK_SET;
610     delay_us(10);
611     Buffer_ram[j] = ((AVR32_GPIO.port[1].pvr) & 0xFF); //CAMERA_INPUT;
612     delay_us(10);
613     FIFO_O_RCLK_CLR;
614     delay_us(10);
615     FIFO_O_RCLK_SET;
616     delay_us(10);
617     Buffer_ram[j] |= (((AVR32_GPIO.port[1].pvr) & 0xFF) << 8); //CAMERA_INPUT;
618     delay_us(10);
619     FIFO_O_RCLK_CLR;
620     delay_us(10);
621 }

623 FIFO_O_nOE_SET;//disable output
624 /* file_close(); */
625 SaveBitmap(Buffer_ram, WIDTH, HEIGHT, Filename_buff);
626 mspace_free(sdram_msp, Buffer_ram);
627 }

```

H.1.1.10 OV7670.c

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

```

1 /*
2  * OV7670_Setup.c
3 */

```

```
4  * Created: 15/02/2013 13:14:09
5  * Author: hslovett
6  */
7
8 #include "CustomDevices/CustomDevices.h"
9
10 const char default_settings[SETTINGS_LENGTH][2] =
11 {
12 {OV_TSLB, 0x04},
13 {OV_COM15, 0xd0}, //RGB565 / RGB555
14 {OV_COM7, 0x14},
15 {OV_HREF, 0x80},
16 {OV_HSTART, 0x16},
17 {OV_HSTOP, 0x04},
18 {OV_VSTART, 0x02},
19 {OV_VSTOP, 0x7b}, //0x7a,
20 {OV_VREF, 0x06}, //0x0a,
21 {OV_COM3, 0x00}, //MSB and LSB swapped
22 {OV_COM14, 0x00}, //
23 {OV_SCALING_XSC, 0x00},
24 {OV_SCALING_YSC, 0x00},
25 {OV_SCALING_DCWCTR, 0x11},
26 {OV_SCALING_PCLK_DIV, 0x00}, //
27 {0xa2, 0x02},
28 {OV_CLKRC, 0x01},
29 {OV_GAM1, 0x20},
30 {OV_GAM2, 0x1c},
31 {OV_GAM3, 0x28},
32 {OV_GAM4, 0x3c},
33 {OV_GAM5, 0x55},
34 {OV_GAM6, 0x68},
35 {OV_GAM7, 0x76},
36 {OV_GAM8, 0x80},
37 {OV_GAM9, 0x88},
38 {OV_GAM10, 0x8f},
39 {OV_GAM11, 0x96},
40 {OV_GAM12, 0xa3},
41 {OV_GAM13, 0xaf},
42 {OV_GAM14, 0xc4},
43 {OV_GAM15, 0xd7},
44 {OV_GAM16, 0xe8},
45 {OV_COM8, 0xe0},
46 {OV_GAIN, 0x00}, //AGC
47 {OV_AECH, 0x00},
48 {OV_COM4, 0x00},
49 {OV_COM9, 0x20}, //0x38, limit the max gain
50 {OV_HIST6, 0x05},
51 {OV_HIST12, 0x07},
52 {OV_AEW, 0x75},
53 {OV_AEB, 0x63},
54 {OV_VPT, 0xA5},
55 {OV_HIST0, 0x78},
56 {OV_HIST1, 0x68},
57 {OV_HIST2, 0x03}, //0x0b,
58 {OV_HIST7, 0xdf}, //0xd8,
59 {OV_HIST8, 0xdf}, //0xd8,
60 {OV_HIST9, 0xf0},
61 {OV_HIST10, 0x90},
```

```
62 {OV_HIST11, 0x94},  
63 {OV_COM8, 0xe5},  
64 {OV_COM5, 0x61},  
65 {OV_COM6, 0x4b},  
66 {0x16, 0x02},  
67 {OV_MVFP, 0x27}, //0x37,  
68 {0x21, 0x02},  
69 {0x22, 0x91},  
70 {0x29, 0x07},  
71 {0x33, 0x0b},  
72 {0x35, 0x0b},  
73 {0x37, 0x1d},  
74 {0x38, 0x71},  
75 {OV_OFON, 0x2a}, //  
76 {OV_COM12, 0x78},  
77 {0x4d, 0x40},  
78 {0x4e, 0x20},  
79 {OV_GFIX, 0x0c}, ///////////////////////////////  
80 {OV_DBLV, 0x60}, //PLL  
81 {OV_REG74, 0x19},  
82 {0x8d, 0x4f},  
83 {0x8e, 0x00},  
84 {0x8f, 0x00},  
85 {0x90, 0x00},  
86 {0x91, 0x00},  
87 {OV_DM_LNL, 0x00}, //0x19, //0x66  
88 {0x96, 0x00},  
89 {0x9a, 0x80},  
90 {0xb0, 0x84},  
91 {0xb1, 0x0c},  
92 {0xb2, 0x0e},  
93 {OV_THL_ST, 0x82},  
94 {0xb8, 0xa},  
95 {OV_AWBC1, 0x14},  
96 {OV_AWBC2, 0xf0},  
97 {OV_AWBC3, 0x34},  
98 {OV_AWBC4, 0x58},  
99 {OV_AWBC5, 0x28},  
100 {OV_AWBC6, 0x3a},  
101 {0x59, 0x88},  
102 {0x5a, 0x88},  
103 {0x5b, 0x44},  
104 {0x5c, 0x67},  
105 {0x5d, 0x49},  
106 {0x5e, 0x0e},  
107 {OV_LCC3, 0x04},  
108 {OV_LCC4, 0x20},  
109 {OV_LCC5, 0x05},  
110 {OV_LCC6, 0x04},  
111 {OV_LCC7, 0x08},  
112 {OV_AWBCTR3, 0xa},  
113 {OV_AWBCTR2, 0x55},  
114 {OV_AWBCTR1, 0x11},  
115 {OV_AWBCTR0, 0x9f}, //0x9e for advance AWB  
116 {OV_GGAIN, 0x40},  
117 {OV_BLUE, 0x40},  
118 {OV_RED, 0x40},  
119 {OV_COM8, 0xe7},
```

```
120 {OV_COM10, 0x02}, //VSYNC negative
121 {OV_MTX1, 0x80},
122 {OV_MTX2, 0x80},
123 {OV_MTX3, 0x00},
124 {OV_MTX4, 0x22},
125 {OV_MTX5, 0x5e},
126 {OV_MTX6, 0x80},
127 {OV_MTDX, 0x9e},
128 {OV_COM16, 0x08},
129 {OV_EDGE, 0x00},
130 {OV_REG75, 0x05},
131 {OV_REG76, 0xe1},
132 {OV_DNSTH, 0x00},
133 {OV_REG77, 0x01},
134 {OV_COM13, 0xc2}, //0xc0,
135 {OV_REG4B, 0x09},
136 {OV_SATCTR, 0x60},
137 {OV_COM16, 0x38},
138 {OV_CONTRAS, 0x40},
139 {0x34, 0x11},
140 {OV_COM11, 0x02}, //0x00, //0x02,
141 {OV_HIST5, 0x89}, //0x88,
142 {0x96, 0x00},
143 {0x97, 0x30},
144 {0x98, 0x20},
145 {0x99, 0x30},
146 {0x9a, 0x84},
147 {0x9b, 0x29},
148 {0x9c, 0x03},
149 {OV_BD50ST, 0x4c},
150 {OV_BD60ST, 0x3f},
151 {0x78, 0x04},
152 {0x79, 0x01}, //Some weird thing with reserved registers.
153 {0xc8, 0xf0},
154 {0x79, 0x0f},
155 {0xc8, 0x00},
156 {0x79, 0x10},
157 {0xc8, 0x7e},
158 {0x79, 0xa},
159 {0xc8, 0x80},
160 {0x79, 0x0b},
161 {0xc8, 0x01},
162 {0x79, 0x0c},
163 {0xc8, 0x0f},
164 {0x79, 0xd},
165 {0xc8, 0x20},
166 {0x79, 0x09},
167 {0xc8, 0x80},
168 {0x79, 0x02},
169 {0xc8, 0xc0},
170 {0x79, 0x03},
171 {0xc8, 0x40},
172 {0x79, 0x05},
173 {0xc8, 0x30},
174 {0x79, 0x26},
175 {OV_COM2, 0x03},
176 {OV_BRIGHT, 0x00},
177 {OV_CONTRAS, 0x40},
```

```

178 {OV_COM11, 0x42}, //0x82, //0xc0, //0xc2, //night mode
180 };

```

H.1.1.11 PCA9542A.h

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

```

1  /*
2   *  PCA9542A.h
3   *
4   *  Created: 15/02/2013 12:21:46
5   *  Author: hslovett
6   */
7
8
9 #ifndef PCA9542A_H_
10 #define PCA9542A_H_
11
12 #define A0 0
13 #define A1 0
14 #define A2 1
15 #define PCA9542A_ADDR (0x70 | (A2 << 2) | (A1 << 1) | A0)
16
17 #define NO_SELECT 0x00
18 //#define ERROR 0x01
19 #define I2C_CHANNEL_0 0x04
20 #define I2C_CHANNEL_1 0x05
21
22 //Status Codes
23 #define SUCCESS 0
24 #define DEVICE_NOT_FOUND 2
25
26 typedef struct {
27     uint8_t Status;
28     uint8_t ChannelSelected;
29 } PCA9542A_t;
30
31 PCA9542A_t PCA9542A;
32 int PCA9542A_Init();
33 //void PCA9542A_Channel_Select(uint8_t Channel);
34 void PCA9542A_Chain_Sel(unsigned char Channel);
35#endif /* PCA9542A_H_ */

```

H.1.1.12 PCA9542A.c

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

```

1 /*
2  *  PCA9542A.c
3  *

```

```

4  * Created: 15/02/2013 12:21:36
5  * Author: hslovett
6  */

8 #include <asf.h>
9 #include "CustomDevices/CustomDevices.h"
10 //Camera
11 /*#include "CustomDevices/OV7670.h"*/
12 //I2C Mux
13 /*#include "CustomDevices/PCA9542A.h"*/
14 //MotorDriver
15 /*#include "CustomDevices/MotorDriver.h"*/
16 //SDCard
17 /*#include "CustomDevices/SD_Card.h"*/

19 int PCA9542A_Init()
20 {
21     int status = twim_probe(&AVR32_TWIMO, PCA9542A_ADDR);
22     if (status != STATUS_OK)
23     {
24         PCA9542A.Status = DEVICE_NOT_FOUND;
25         return DEVICE_NOT_FOUND;
26     }
27     char buff[2] = {NO_SELECT, 0};
28     status = twim_write(&AVR32_TWIMO, &buff, 1, PCA9542A_ADDR, false);
29     PCA9542A.Status = STATUS_OK;
30     PCA9542A.ChannelSelected = NO_SELECT;
31     return status;
32 }

35 void PCA9542A_Chан_Sel(unsigned char Channel)
36 {
37     int status = 0;
38     char buff[2] = {Channel, 0};
39     status = twim_write(&AVR32_TWIMO, &buff, 1, PCA9542A_ADDR, false);
40     if(status == STATUS_OK)
41     {
42         PCA9542A.ChannelSelected = Channel;
43     }
44     else
45     {
46         PCA9542A.Status = ERR_PROTOCOL;
47     }
48 }
```

H.1.1.13 SD_Card.h

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

```

1 /*
2  * SD_Card.h
3  *
4  * Created: 10/02/2013 17:11:51
```

```

5   * Author: hslovett
6   */

9 #ifndef SD_CARD_H_
10 #define SD_CARD_H_
11 #include "ImageProcessor.h"
12 #define SIGNAL_FILE "signal.bin"
13 #define TWOD_SIGNAL_FILE "signal2d.bin"

15 typedef struct {
16     uint8_t Status;
17     uint32_t Memory_size;
18 } SD_Status_t;
19 SD_Status_t SD_Status;

21 void local_pdca_init(void);
22 void sd_mmc_resources_init(void);
23 static void pdca_int_handler(void);
24 void wait();
25 void Log_Write_ulong(unsigned long n);
26 void Log_Write(char *buff, int length);
27 void SaveBuff( int * WorkingBuffer , int size);
28 int Read2DSignal( int * WorkingBuffer );
29 int ReadSignal( int * WorkingBuffer );
30 void SaveBitmap(uint16_t *Image, int width, int height, char *FileName);
31 //void ReadBitmap(char *Filename);
32 void ReadBitmap(char *Filename, Image_t *image);
33 void SaveBuff_CSV(char *Filename, int *WorkingBuffer, int size);
34 void SaveCBuff_CSV(char *Filename, dsp16_complex_t *ComplexBuffer, int size);
35 void Read_CSV(char *Filename, int *WorkingBuffer, int size);
36#endif /* SD_CARD_H_ */

```

H.1.1.14 SD_Card.c

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

```

1 /*
2  * SD_Card.c
3  *
4  * Created: 10/02/2013 17:11:58
5  * Author: hslovett
6  */
7 //Camera
8 /*#include "CustomDevices/0V7670.h"*/
9 //I2C Mux
10 /*#include "CustomDevices/PCA9542A.h"*/
11 //MotorDriver
12 /*#include "CustomDevices/MotorDriver.h"*/
13 //SDCard
14 /*#include "CustomDevices/SD_Card.h"*/
15 #include "CustomDevices/CustomDevices.h"
16 #include "conf_sd_mmc_spi.h"
17 #include <asf.h>

```

```
18 #include "stdlib.h"
19 #include "stdio.h"
20 // Dummy char table
21 const char dummy_data[] =
22 #include "dummy.h"
23 ;

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

35 void wait()
36 {
37     volatile int i;
38     for(i = 0 ; i < 5000; i++);
39 }
40 /* interrupt handler to notify if the Data reception from flash is
41 * over, in this case lunch the Memory(ram_buffer) to USART transfer and
42 * disable interrupt*/
43
44 static void pdca_int_handler(void)
45 {
46     // Disable all interrupts.
47     Disable_global_interrupt();

48     // Disable interrupt channel.
49     pdca_disable_interrupt_transfer_complete(AVR32_PDCA_CHANNEL_SPI_RX);

50     sd_mmc_spi_read_close_PDCA(); //unselects the SD/MMC memory.
51     wait();
52     // Disable unnecessary channel
53     pdca_disable(AVR32_PDCA_CHANNEL_SPI_TX);
54     pdca_disable(AVR32_PDCA_CHANNEL_SPI_RX);

55     // Enable all interrupts.
56     Enable_global_interrupt();

57     end_of_transfer = true;
58 }

59 /*! \brief Initializes SD/MMC resources: GPIO, SPI and SD/MMC.
60 */
61 void sd_mmc_resources_init(void)
62 {
63     // GPIO pins used for SD/MMC interface
64     static const gpio_map_t SD_MMC_SPI_GPIO_MAP =
65     {
66         {SD_MMC_SPI_SCK_PIN, SD_MMC_SPI_SCK_FUNCTION }, // SPI Clock.
67         {SD_MMC_SPI_MISO_PIN, SD_MMC_SPI_MISO_FUNCTION}, // MISO.
68         {SD_MMC_SPI_MOSI_PIN, SD_MMC_SPI_MOSI_FUNCTION}, // MOSI.
69         {SD_MMC_SPI_NPCS_PIN, SD_MMC_SPI_NPCS_FUNCTION} // Chip Select NPCS.
70     };
71 }
```

```

75    };
76
77    // SPI options.
78    spi_options_t spiOptions =
79    {
80        .reg          = SD_MMC_SPI_NPCS,
81        .baudrate     = SD_MMC_SPI_MASTER_SPEED, // Defined in conf_sd_mmc_spi.h.
82        .bits         = SD_MMC_SPI_BITS,           // Defined in conf_sd_mmc_spi.h.
83        .spck_delay   = 0,
84        .trans_delay  = 0,
85        .stay_act     = 1,
86        .spi_mode     = 0,
87        .modfdis      = 1
88    };
89
90    // Assign I/Os to SPI.
91    gpio_enable_module(SD_MMC_SPI_GPIO_MAP,
92                        sizeof(SD_MMC_SPI_GPIO_MAP) / sizeof(SD_MMC_SPI_GPIO_MAP
93                        [0]));
94
95    // Initialize as master.
96    spi_initMaster(SD_MMC_SPI, &spiOptions);
97
98    // Set SPI selection mode: variable_ps, pcs_decode, delay.
99    spi_selectionMode(SD_MMC_SPI, 0, 0, 0);
100
101    // Enable SPI module.
102    spi_enable(SD_MMC_SPI);
103
104    // Initialize SD/MMC driver with SPI clock (PBA).
105    sd_mmc_spi_init(spiOptions, PBA_HZ);
106}
107
107/*! \brief Initialize PDCA (Peripheral DMA Controller A) resources for the SPI
108   transfer and start a dummy transfer
109 */
110void local_pdca_init(void)
111{
112    // this PDCA channel is used for data reception from the SPI
113    pdca_channel_options_t pdca_options_SPI_RX ={ // pdca channel options
114
115        .addr = ram_buffer,
116        // memory address. We take here the address of the string dummy_data. This
117        // string is located in the file dummy.h
118
119        .size = 512,                                // transfer counter: here the
120        // size of the string
121        .r_addr = NULL,                             // next memory address after 1st
122        // transfer complete
123        .r_size = 0,                               // next transfer counter not
124        // used here
125        .pid = AVR32_PDCA_CHANNEL_USED_RX,          // select peripheral ID - data
126        // are on reception from SPI1 RX line
127        .transfer_size = PDCA_TRANSFER_SIZE_BYTE // select size of the transfer:
128        // 8,16,32 bits
129    };
130

```

```
124 // this channel is used to activate the clock of the SPI by sending a dummy
125 // variables
126 pdca_channel_options_t pdca_options_SPI_TX ={ // pdca channel options
127     .addr = (void *)&dummy_data, // memory address.
128     // We take here the address of
129     // the string dummy_data.
130     // This string is located in the
131     file dummy.h
132     .size = 512, // transfer counter: here the
133     // size of the string
134     .r_addr = NULL, // next memory address after 1st
135     // transfer complete
136     .r_size = 0, // next transfer counter not
137     // used here
138     .pid = AVR32_PDCA_CHANNEL_USED_TX, // select peripheral ID - data
139     // are on reception from SPI1 RX line
140     .transfer_size = PDCA_TRANSFER_SIZE_BYTE // select size of the transfer:
141     // 8,16,32 bits
142 };
143
144 // Init PDCA transmission channel
145 pdca_init_channel(AVR32_PDCA_CHANNEL_SPI_TX, &pdca_options_SPI_TX);
146
147 // Init PDCA Reception channel
148 pdca_init_channel(AVR32_PDCA_CHANNEL_SPI_RX, &pdca_options_SPI_RX);
149
150 //!\ brief Enable pdca transfer interrupt when completed
151 INTC_register_interrupt(&pdca_int_handler, AVR32_PDCA_IRQ_0, AVR32_INTC_INT1
152 ); // pdca_channel_spi1_RX = 0
153
154 }
155
156 #define BUFFER_FILENAME "Buffer.bin"
157 #define BUFFERCSV_FILENAME "Buffer.csv"
158 void SaveBuff( int * WorkingBuffer , int size)
159 {
160     //If the file exists, delete it
161     if(nav_filelist_findname((FS_STRING)BUFFER_FILENAME, false))
162     {
163         nav_setcwd((FS_STRING)BUFFER_FILENAME, false, false);
164         nav_file_del(false);
165     }
166     nav_file_create((FS_STRING)BUFFER_FILENAME);
167     nav_setcwd((FS_STRING)BUFFER_FILENAME, false, true);
168     file_open(FOPEN_MODE_APPEND);
169     file_write_buf(WorkingBuffer, size * sizeof(WorkingBuffer));
170     file_close();
171 }
172 void SaveBuff_CSV(char *Filename, int *WorkingBuffer, int size)
173 {
174     int i, j;
175     char Buff[16];
176     //If the file exists, delete it
177     nav_filelist_reset();
178     if(nav_filelist_findname((FS_STRING)Filename, false))
179     {
180         nav_setcwd((FS_STRING)Filename, false, false);
181     }
```

```
173     nav_file_del(false);
174 }
175 nav_file_create((FS_STRING)Filename);
176 nav_setcwd((FS_STRING)Filename, false, true);
177 file_open(FOPEN_MODE_W);
178 for(i = 0; i < size; i++)
{
180     sprintf(Buff, "%d,", WorkingBuffer[i]);
181 //itoa(WorkingBuffer[i], Buff, 10);
182     j = 0;
183     while(Buff[j++] != 0); //count the size of data to be written
184     atoi(Buff);
185     file_write_buf(Buff, j-1);
186 //file_write_buf(",", 1);
187 }

189     file_close();
190 }
191 void SaveCBuff_CSV(char *Filename, dsp16_complex_t *ComplexBuffer, int size)
192 {
193     int i, j;
194     char Buff[16];
195 //If the file exists, delete it
196     nav_filelist_reset();
197     if(nav_filelist_findname((FS_STRING)Filename, false))
198     {
199         nav_setcwd((FS_STRING)Filename, false, false);
200         nav_file_del(false);
201     }
202     nav_file_create((FS_STRING)Filename);
203     nav_setcwd((FS_STRING)Filename, false, true);
204     file_open(FOPEN_MODE_W);
205     for(i = 0; i < size; i++)
{
207         if(ComplexBuffer[i].imag >= 0)
208             sprintf(Buff, "%d%dj,", ComplexBuffer[i].real, ComplexBuffer[i].imag);
209         else
210             sprintf(Buff, "%d%dj,", ComplexBuffer[i].real, ComplexBuffer[i].imag);
211 //itoa(WorkingBuffer[i], Buff, 10);
212     j = 0;
213     while(Buff[j++] != 0); //count the size of data to be written
214     atoi(Buff);
215     file_write_buf(Buff, j-1);
216 //file_write_buf(",", 1);
217 }

219     file_close();
220 }
221 void Read_CSV(char *Filename, int *WorkingBuffer, int size)
222 {
223     char Buff[32];
224     int i, j;
225     char c;
226     nav_filelist_reset();
227     if(WorkingBuffer == NULL){
228         print_dbg("\n\rRead_CSV: Buffer not initialised");
229         return;
230 }
```

```
231 //Check file Exists
232 if(nav_filelist_findname((FS_STRING)Filename, false) == false){
233     print_dbg("\n\rRead_CSV : File doesn't exist;");
234     return;
235 }
236 nav_setcwd((FS_STRING)Filename, false, true);
237 file_open(FOPEN_MODE_R);
238 for(i = 0; i < size; i++)
239 {
240     if(file_eof())
241         break;
242     c = 0;
243     //j = 0;
244     for(j = 0; j < 32; j++)
245         Buff[j] = 0; //clear the buffer
246     j = 0;
247     while(c != ',')
248     {
249         if(file_eof())
250             break;
251         c = file_getc();
252         if(c == ',')
253             break;
254         Buff[j++] = c; //load string into buffer
255     }
256     WorkingBuffer[i] = atoi(Buff); //Convert to int and put into buffer
257 }
258 file_close();
259 return;
260 }
261 void Log_Write(char *buff, int length)
262 {
263     nav_setcwd((FS_STRING)LOG_FILE, true, false);
264     file_open(FOPEN_MODE_APPEND);
265     if(length == -1)
266         length = sizeof(buff);
267     file_write_buf(buff, length);
268     file_close();
269 }
270 void Log_Write_ulong(unsigned long n)
271 {
272     char tmp[11];
273     int i = sizeof(tmp) - 1;

275 // Convert the given number to an ASCII decimal representation.
276     tmp[i] = '\0';
277     do
278     {
279         tmp[--i] = '0' + n % 10;
280         n /= 10;
281     } while (n);

283 // Transmit the resulting string with the given USART.
284 Log_Write(tmp + i, -1);
285 }

287 int ReadSignal( int * WorkingBuffer )
288 {
```

```
289     bool status_b;
290     int Status, temp;
291     char c = 0;
292     if(Columbus_Status.SD_Card->Status != STATUS_OK)
293         return ERR_IO_ERROR;
294     nav_filelist_reset();
295     nav_setcwd((FS_STRING)SIGNAL_FILE, false, false);
296     status_b = file_open(FOPEN_MODE_R);
297     if(status_b == false)
298     {
299         print_dbg("File Open Error");
300         return ERR_IO_ERROR;
301     }
302
303
304 //Status = file_read_buf(WorkingBuffer, 16);
305 for(Status = 0; Status < FFT_SIZE; Status++)
306 {
307 //    print_dbg("\n\r Read from file: ");
308     c = 0;
309     temp = 0;
310     temp |= file_getc() << 24;
311     temp |= file_getc() << 16;
312     temp |= file_getc() << 8;
313     temp |= file_getc();
314
315 //    print_dbg_char(c);
316
317     WorkingBuffer[Status] = temp;
318 //    print_dbg(" Working Buff = ");
319 //    print_dbg_char(WorkingBuffer[Status]);
320 }
321 file_close();
322 return STATUS_OK;
323 }
324
325 int Read2DSignal( int * WorkingBuffer )
326 {
327     bool status_b;
328     int Status, temp;
329     char c = 0;
330     if(Columbus_Status.SD_Card->Status != STATUS_OK)
331         return ERR_IO_ERROR;
332     nav_filelist_reset();
333     nav_setcwd((FS_STRING)TWOD_SIGNAL_FILE, false, false);
334     status_b = file_open(FOPEN_MODE_R);
335     if(status_b == false)
336     {
337         print_dbg("File Open Error");
338         return ERR_IO_ERROR;
339     }
340
341
342 //Status = file_read_buf(WorkingBuffer, 16);
343 for(Status = 0; Status < FFT_SIZE * FFT_SIZE; Status++)
344 {
345 //    print_dbg("\n\r Read from file: ");
346     c = 0;
```

```
347     temp = 0;
348     temp |= file_getc() << 24;
349     temp |= file_getc() << 16;
350     temp |= file_getc() << 8;
351     temp |= file_getc();

353 //      print_dbg_char(c);

355 WorkingBuffer[Status] = temp;
356 //      print_dbg("  Working Buff = ");
357 //      print_dbg_char(WorkingBuffer[Status]);
358 }
359 file_close();
360 return STATUS_OK;
361 }

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

368     nav_filelist_reset();
369     if(nav_filelist_findname((FS_STRING)FileName, false))
370     {
371         nav_setcwd((FS_STRING)FileName, true, false);
372         nav_file_del(false);
373     }
374     nav_file_create((FS_STRING)FileName);
375     file_open(FOPEN_MODE_W);
376     //write a modified bitmap header
377     //Calculate which is the biggest:
378     i = width * 2;
379     if(height > i)
380         i = height;
381     if(DIBHEADERSIZE > i)
382         i = DIBHEADERSIZE;

384     Buffer = malloc(i);

386     for(i = 0; i < BMPHEADERSIZE; i++)//copy all the header
387     {
388         Buffer[i] = BMPHeader[i];
389     }
390     //edit the size field
391     j = width * height * 2 + BMPHEADERSIZE + DIBHEADERSIZE;
392     for(i = 0; i < 4; i++)
393     {
394         Buffer[i + 2] = (uint8_t)(j >> 8*i);
395     }

397     file_write_buf(Buffer, BMPHEADERSIZE);

399 //DIB Header
400     for(i = 0; i < DIBHEADERSIZE; i++)
401     {
402         Buffer[i] = DIBHead[i];
403     }
404     Buffer[4] = (uint8_t)(width & 0xFF);
```

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

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

414     file_write_buf(Buffer, DIBHEADERSIZE);

416     for(i = 0; i < height ; i++)
417     {
418         for(j = 0; j < width ; j++)
419         {
420             //Copy the data across.

422             /*Buffer[j] = Image[i*width + j];*/
423             Buffer[(2 * j) + 1] = (uint8_t)(Image[i*width + j]);
424             Buffer[(2 * j)] = (uint8_t)(Image[i*width + j] >> 8);
425         }
426         if(file_write_buf(Buffer, width * 2) != (width * 2))
427         {
428             print_dbg("\n\rFile write error.");
429         }

431         //    j = width % 4;
432         //    if(j != 0)
433         //        {//Padding is needed to make things 4 byte aligned
434         //            file_write_buf(Buffer, j);
435         //        }
436     }

440     free(Buffer);
441     file_close();
442 }

444 #define BMP_HEADER_FILESIZE_OFFSET      2
445 #define BMP_HEADER_OFFSETTOARRAY_OFFSET 10
446 #define DIB_V5_WIDTH_OFFSET           4
447 #define DIB_V5_HEIGHT_OFFSET          8
448 #define DIB_V5_BITCOUNT_OFFSET        14
449 #define DIB_V5_IMAGESIZE_OFFSET       20

451 int ReadBigEndian(uint8_t *Buffer, int Offset, uint size)
452 {
453     int retVal, i;
454     retVal = 0; //initialise value
455     for(i = 0; i < size; i++)
456     {
457         retVal |= Buffer[Offset + i] << (i * 8);
458     }
459     return (Buffer[Offset]) | (Buffer[Offset + 1] << 8) | (Buffer[Offset + 2] << 16) | (Buffer[Offset + 3] << 24);
460 }
461 void ReadBitmap(char *Filename, Image_t *image)
```

```
462 {
463 // Image_t image;
464 int i, j, FileSize, OffsetToArray, temp, BitCount, ImageSize;
465 uint8_t Buffer[128];
466 nav_filelist_reset();
467 if(nav_filelist_findname((FS_STRING)Filename, false) == false)//if the file
468 doesn't exist
469 {
470 print_dbg("\n\rFile ");
471 print_dbg(Filename);
472 print_dbg("\n\r does not exist;");
473 return;
474 }
475 nav_setcwd((FS_STRING)Filename, false, false);
476 file_open(FOPEN_MODE_R);
477 //Read Header
478 file_read_buf(Buffer, BMPHEADERSIZE);
479 //Check for BM to confirm it is a Bitmap
480 if((Buffer[0] != 'B') || (Buffer[1] != 'M'))
481 {
482 print_dbg("\n\rBitmap Parse Fail 'BM';");
483 return;
484 }
485 //Extract file size and offset to pixel array
486 FileSize = ReadBigEndian(Buffer, BMP_HEADER_FILESIZE_OFFSET, 4);
487 OffsetToArray = ReadBigEndian(Buffer, BMP_HEADER_OFFSETTOARRAY_OFFSET, 4);

488 file_read_buf(Buffer, DIBHEADERSIZE);
489 temp = ReadBigEndian(Buffer, 0, 4);
490 if(temp != 0x7C) //check it is a V5 BMP DIB Header
491 {
492 print_dbg("\n\rBMP Parse: DIB Header not V5;");
493 return;
494 }
495 image->Width= ReadBigEndian(Buffer, DIB_V5_WIDTH_OFFSET, 4);
496 image->Height = ReadBigEndian(Buffer, DIB_V5_HEIGHT_OFFSET, 4);
497 BitCount = ReadBigEndian(Buffer, DIB_V5_BITCOUNT_OFFSET, 2);
498 ImageSize = ReadBigEndian(Buffer, DIB_V5_IMAGESIZE_OFFSET, 4);
499 print_dbg("\n\rBitmap Width = ");
500 print_dbg_ulong(image->Width);
501 print_dbg("\n\rBitmap Height = ");
502 print_dbg_ulong(image->Height);
503 print_dbg("\n\rBitmap File Size = ");
504 print_dbg_ulong(FileSize);
505 print_dbg("\n\rBitmap Offset to Array = ");
506 print_dbg_ulong(OffsetToArray);
507 print_dbg("\n\rBitmap Image Bitcount = ");
508 print_dbg_ulong(BitCount);
509 print_dbg("\n\rBitmap Image Size = ");
510 print_dbg_ulong(ImageSize);

512 file_seek(OffsetToArray, FS_SEEK_SET);
513 j = 0;
514 image->ImagePtr = mspace_malloc(sdram_msp, image->Height * image->Width);
515 for(i = 0; i < ImageSize; i += 2)
516 {
517 image->ImagePtr[j++] = (file_getc()<<8) | (file_getc());
518 }
```

```

519     file_close();
520     nav_filelist_reset();
521     return;
522 }
```

H.1.1.15 TWI.c

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

```

1  /*
2   *  TWI.c
3   *
4   *  Created: 27/02/2013 10:51:19
5   *  Author: hslovett
6   */
7
8 #include "CustomDevices/CustomDevices.h"
9 #include <asf.h>
10
11 #define TARGET_ADDRESS      0x0          //!< Target's TWI address
12 #define TARGET_ADDR_LGT      3            //!< Internal Address length
13 #define VIRTUALMEM_ADDR     0x123456    //!< Internal Address
14 #define TWIM_MASTER_SPEED    50000        //!< Speed of TWI
15
16
17 void twim_init (void)
18 {
19     int8_t status;
20
21     /**
22     * \internal
23     * PIN 2 & 3 in Header J24 can be used in EVK1104
24     * PIN 1 & 2 in Header J44 can be used in UC3C_EK
25     * \endinternal
26     */
27     const gpio_map_t TWIM_GPIO_MAP = {
28     {AVR32_TWIMSO_TWCK_0_0_PIN, AVR32_TWIMSO_TWCK_0_0_FUNCTION},
29     {AVR32_TWIMSO_TWD_0_0_PIN, AVR32_TWIMSO_TWD_0_0_FUNCTION}
30 };
31
32 // Set TWIM options
33 const twi_options_t TWIM_OPTIONS = {
34     .pba_hz = FOSCO,
35     .speed = TWIM_MASTER_SPEED,
36     .chip = TARGET_ADDRESS,
37     .smbus = false,
38 };
39 // TWIM gpio pins configuration
40 gpio_enable_module (TWIM_GPIO_MAP,
41     sizeof (TWIM_GPIO_MAP) / sizeof (TWIM_GPIO_MAP[0]));
42
43 // Initialize as master.
44 status = twim_master_init (&AVR32_TWIMO, &TWIM_OPTIONS);
45 }
```

H.2 MATLAB Code

H.2.1 Image Matching Algorithms

H.2.1.1 Match.m

..//MATLAB/Match.m

```

1  function [ MatchDiff ] = Match( left, right )
2  %UNTITLED2 Summary of this function goes here
3  % Detailed explanation goes here
4  %close all;
5  % Global Constants
6  BoxSize = 50

8  % Display Images
9  f = figure;
10 subplot(2,2,1);
11 imshow(left);
12 title('Left Image');
13 subplot(2,2,2);
14 imshow(right);
15 title('Right Image');
16 % Get Template
17 figure;
18 imshow(right);
19 title('Choose location to Cross Correlate with the Left Image');
20 rSubCoord = ginput(1);
21 [rSubCoord(2), rSubCoord(1)];
22 rSubCoord = round(rSubCoord);
23 Rx = rSubCoord(1);
24 Ry = rSubCoord(2);
25 close;
26 leftG = rgb2gray(left);
27 rightG = rgb2gray(right);
28 % Extract Template
29 RightTemplate = rightG(Ry-(BoxSize/2):Ry+(BoxSize/2), Rx-(BoxSize/2):Rx+(BoxSize/2));

31 %Cross Correlate
32 NCC = normxcorr2(RightTemplate, leftG);

34 %Display
35 figure(f);
36 subplot(2,2,3);
37 surf(NCC);
38 hold on;
39 shading interp;

41 %Find Maximum
42 [~, 1] = max(NCC(:));
43 %Return Match Coordinates
44 [~, Lx] = ind2sub(size(NCC),1);
45 Lx = Lx-BoxSize/2;
```

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

48 MatchDiff = abs(Lx - Rx);
49
50 end
```

H.2.1.2 MatchAll.m

..../MATLAB/MatchAll.m

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

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

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

7
8 end
```

References

- Atmel Corporation. *AVR311: TWI Slave*, 2007.
- 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.
- PCB Cart. [Pcb manufacturer](#), 2013.
- 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. *Electronics 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.
- 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.