

UNIVERSITY OF SOUTHAMPTON
FACULTY OF PHYSICAL AND APPLIED SCIENCES
Electronics and Computer Science

Two Dimensional Stereoscopic Mapping 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 December, 2012

Turn off iNotes!

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL AND APPLIED SCIENCES
Electronics and Computer Science

A project report submitted for the award of MEng Electronic Engineering

TWO DIMENSIONAL STEREOSCOPIC MAPPING ROBOT

by Henry S. Lovett

This paper describes the research, design, build and testing of a stereoscopic mapping robot. Two cameras are used to locate objects and calculate distances to solid items. This allows an occupancy map to be built up of the unknown area. The object detection is done by a normalised cross correlation and uses derived equations to find distances. The camera used is an OV7670 by OmniVision delivering QVGA sized images. The end product will be a mobile wheeled robot that explores an unknown area. Multiple modules of the design have been developed and tested and all aspects will be brought together onto an AVR32 for the final product.

Contents

List of Symbols	xi
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 Initial Hardware and Firmware Development	9
3.1 Camera	9
3.1.1 Single Camera Operation	10
3.1.2 Dual Camera Operation	11
3.2 SD Card	12
3.2.1 Storing Images	13
3.2.2 User Interface	14
3.3 Circuit and PCB Development	15
3.3.1 Il Matto Development	15
4 Investigation into Vision Algorithms	17
4.1 Comparison	17
4.1.1 Sum of Absolute Differences	18
4.1.2 Sum of Squared Differences	18
4.1.3 NCC	18
4.1.4 Comparison	19
4.1.5 Conclusion	21
4.2 Range Finding	21
4.2.1 Derivations	21
4.2.1.1 Object is between the Cameras	21
4.2.1.2 Object is to the same side in each camera	23
4.2.1.3 Object is in front of a camera	24
4.2.1.4 Summary	24

5 Conclusions and Further Work	27
A Gantt Chart	29
B Circuit Diagrams	31
C Bitmap File Format	33
C.1 Bitmap File Format	33
D Source Code	37
D.1 C Code for AVR	37
D.1.1 Dual Camera Operation	37
D.1.1.1 main.c	37
D.1.1.2 Bitmap.h	41
D.1.1.3 Bitmap.c	41
D.1.1.4 Config.h	43
D.1.1.5 Config.c	44
D.1.1.6 DualCameras.h	45
D.1.1.7 DualCameras.c	50
D.1.1.8 PCA9542A.h	58
D.1.1.9 PCA9542A.c	59
D.1.1.10 TWI_Master.h	60
D.1.1.11 TWI_Master.c	63
D.1.1.12 Usart.h	68
D.1.1.13 Usart.c	69
D.1.2 Dual Camera User Interface	70
D.1.2.1 DualCamera_UI.c	70
D.1.2.2 TWI_slave.h	72
D.1.2.3 TWI_slave.c	75
D.2 MATLAB Code for Image Algorithm Prototyping	82
D.2.1 Image Matching Algorithms	82
D.2.1.1 loadimages.m	82
D.2.1.2 GetSubImage.m	82
D.2.1.3 SADAll.m	83
D.2.1.4 SSDAll.m	84
D.2.1.5 NCC.m	85
References	87

List of Figures

1.1	The base of the robot will use	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
4.1	Stereoscopic Test Images from MATLAB Examples	17
4.2	Result Graphs of Comparison Algorithms	20
4.3	Problem 1 - Object is between the Cameras	22
4.4	Problem 2 - Object is to the same side in both cameras	25
4.5	Problem 3 - Object is directly in front of a camera	26
A.1	Gantt Chart of how time will be spent in the areas of the project . .	30
B.1	The circuit diagram for the OV7670 breakout board	31
B.2	The circuit diagram for Dual Cameras using the Il Matto Board . .	32

List of Tables

1.1	A list of risks and the prevention steps to reduce their impact	3
2.1	Comparison Table of some common microcontrollers. Data of microcontroller 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
C.1	Format of a Bitmap file with values used, to write an image from the camera to an SD Card	33

List of Symbols

I^2C	Inter-Integrated Circuit
TWI	Two Wire Interface
SCCB	Serial Camera Control Bus
SPI	Serial Peripheral Interface
kB	KiloBytes
φ_0	Field of View of the Camera
φ_1, φ_2	Angle from camera to the object
B	Seperation 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

Chapter 1

Introduction

This report documents the designing, testing and building of a stereoscopic mapping robot. The end product will be a small, two wheeled robot with a roller ball, that will autonomously search and map an unknown area and produce an occupancy map of the searched area.

Stereoscopy in computer vision is the ability to calculate the locations and depths using images from two cameras that 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.

An occupancy map is a representation of an area where a location is given a value of either *free*, *occupied* or *unknown* [Thrun \(2003\)](#). Three dimensional occupancy maps can also be generated, such as the OctoMap ([Wurm et al. \(2010\)](#)). Objects can also be tracked using an occupancy map and statistics ([Fleuret et al. \(2007\)](#)).

The purpose of a mapping robot is to build a representation of the area around it. This then facilitates the use of any application that requires prior knowledge of the area. One algorithm that is used to build up an occupancy map is the S.L.A.M. algorithm ([Thrun and Montemerlo \(2006\)](#)) and is used in other mapping robots([Se et al. \(2002\)](#)). Accurate mapping robots tend to use laser range finders ([Ruhnke et al. \(2011\)](#)).

Stereovision is a small section of computer vision which is widely used in many applications, including Microsofts Xbox Kinect ([Microsoft \(2012\)](#)), where stereo vision is used to locate a game player in order to use their movements to control

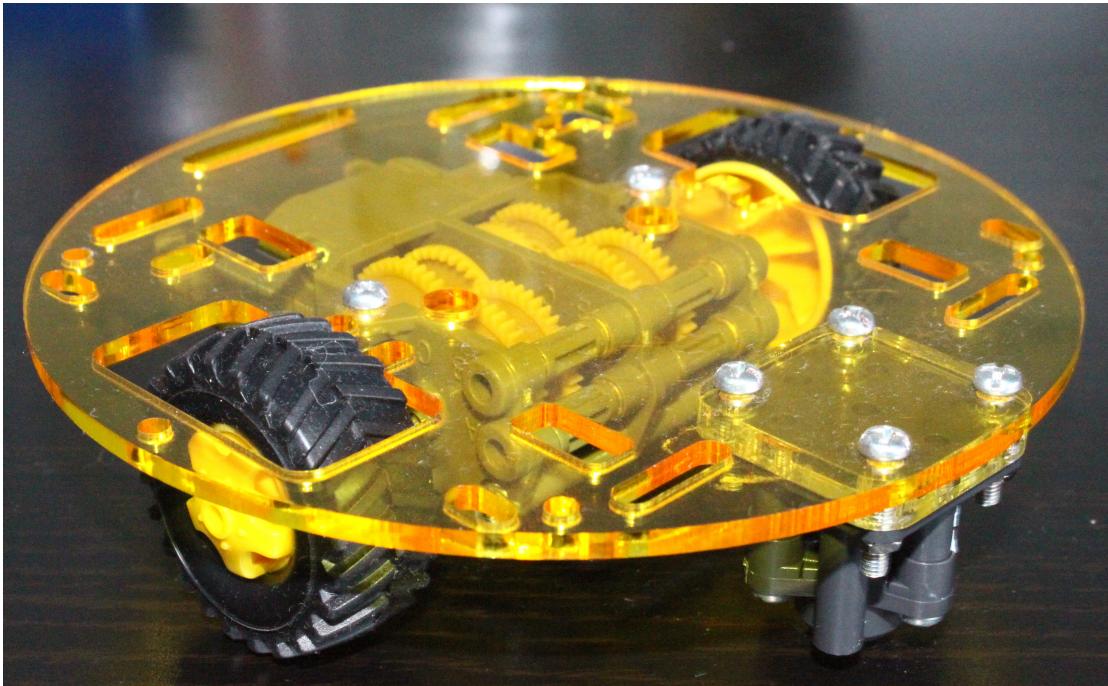


Figure 1.1: The base of the robot will use

the game. [Mrovlje and Vrančić \(2008\)](#) uses stereovision to be able to locate the distance to a marker.

The stereovision mapping 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 up to VGA format images.

1.1 Project Management

In order to reduce the risk with in the project, all aspects of potential issue are looked at and are summarised in table 1.1. A Gannt chart of how time will be spent can be seen in [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 can 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
Not completing	High	Develop in stages to obtain functionality in parts
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 to reduce their impact

Chapter 2

Research

The research done for this project is split into three sections:

1. Hardware
2. Software, broken down into:
 - (a) Firmware
 - (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.

For this comparison, the Stellaris by Texas Instruments will be examined. ARM cores are based on a RISC Harvard architecture and tend to be 32-bit with a high clock speed. ARM microcontrollers have onboard support for SPI, I^2C , PWN, ADCs and have Flash, SRAM and EEPROM memory.

Atmel have a variety of products in the microcontroller market. They range from 8-bit, low clock speed devices for the hobbyist, the ATMega and ATTiny series to an improved 8-bit variant, XMega, and a 32-bit design, AT32UC3C. XMegas and AVR32s tend to have higher clock speeds than the ATMegas. The AVR core is also a Harvard RISC architecture, but mainly 8-bit. Atmel devices often have on board peripherals such as I^2C (called TWI on AVR), SPI, ADCs as well as a number of different memories: Flash, EEPROM and SRAM. An AT32UC3C0512C, ATXmega256A3BU and ATMega644P will be compared in this section.

Table 2.1 shows a brief summary of some common ARM and AVR microcontrollers. The Stellaris offers the most power with the largest DMIPS performance. However, due to the necessity of floating point operations, the AT32 clearly has a distinct advantage by having a built-in floating point unit. The XMega and ATMega do not offer enough power and are restricted by a small amount of SRAM and Flash. All devices looked at use 3.3V supply and have basic communication protocols (SPI, I^2C). 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 and 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 II 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. It 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
SRAM 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 microcontroller 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 \(2012c\)](#). 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

Initial Hardware and Firmware Development

For initial development, the *Il Matto* board, designed by Steve Gunn, which has an ATMega644P, was used. The system is clocked at 16MHz and has an on-board SD card connector.

The following section is broken down into the following parts:

1. Camera Code
2. SD Card
3. Circuit and 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, of which I used to gain the operation required. The code streamed continuous video to a TFT screen. The operation required was to take a single photo.

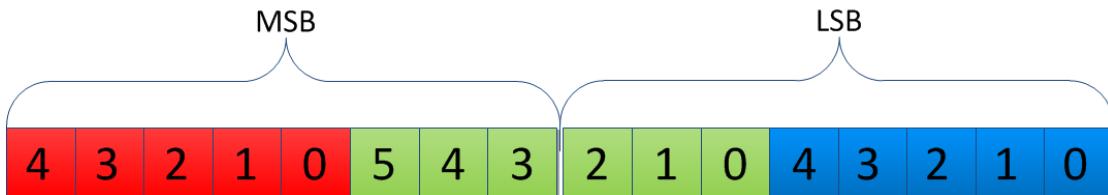


Figure 3.1: RGB565 pixel format

3.1.1 Single Camera Operation

The camera uses a SCCB Interface([OmniVision \(2007\)](#)) created by OmniVision. This is almost identical to the I^2C Interface by Phillips. The original code used a bit-banged 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 used in my application.

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 an inconsistent colour ratio of the green field.

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 s (ATMegas typically clocked at 12-16MHz) to be able to respond to the camera quick enough. This highlights the importance of 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. In order to acquire a full frame, the first VSYNC pin is set up to interrupt the AVR to enable WEN. The operation is then automatic and all the data is clocked into the buffer until the second interrupt of VSYNC where WEN is disabled. At this point, the entire frame of data is stored in the buffer.

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 ”

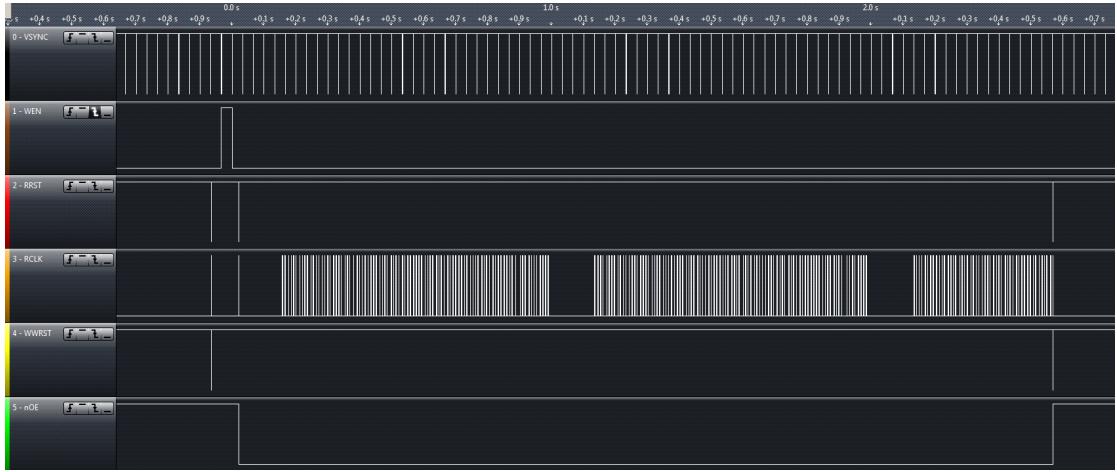


Figure 3.2: Signals generated to control the OV7670 capture and read

To obtain the data from the buffer, the AVR sets output enable and manually pulses the read clock. Valid data is available on the input port chosen. 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) 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 was decided is used as FAT file system. This allows data to 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.

The buffers have an output enable pin so the data bus can be shared by both cameras to the AVR. All buffer function pins are driven from pins, although a demultiplexer could be used if pins are short. 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 VSYNC method and 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 from one and then the other by the AVR.

A major problem now occurred with using the I^2C interface to set up both cameras. The camera has a set 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, there can be no bus contention. However, SCCB is slow and processor-hungry as it deals with the protocol bit by bit. Space for the code then has to be made and this code cannot be reused.

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^2CMUX 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 and simplify the operation and reduce the code size so an I^2CMUX will be used. A suitable multiplexer is the Phillips PCA9542A ([Phillips \(2009\)](#)).

Operation to read an image is identical to using one camera. 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, 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.

	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 \(2010\)](#))

An SD card was chosen due to it's 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 so the card only uses one extra enable pin in hardware to function.

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 much computational time to compress the image into the correct format. To avoid compression, and thereby save processing time, bitmap was chosen at the expense of using more memory. The data in a bitmap image is also stored in RGB format so can be read back easily when processing the data. Appendix C shows the make up of a Bitmap File that was used.

By writing the image in this format, the images 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.



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

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 ATMega 168 was added on the I^2C bus to act as a port extender. The ATMega168 accepts a read or write, places the written data on Port D and reads in the lower nibble of 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 [Atmel Corporation \(2007\)](#), written for IAR Compiler. This code was altered to compile with GCC under Atmel Studio. AVRs contain a hardware based I^2C protocol that is interrupt based in software. The interrupt service routine 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.

	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.

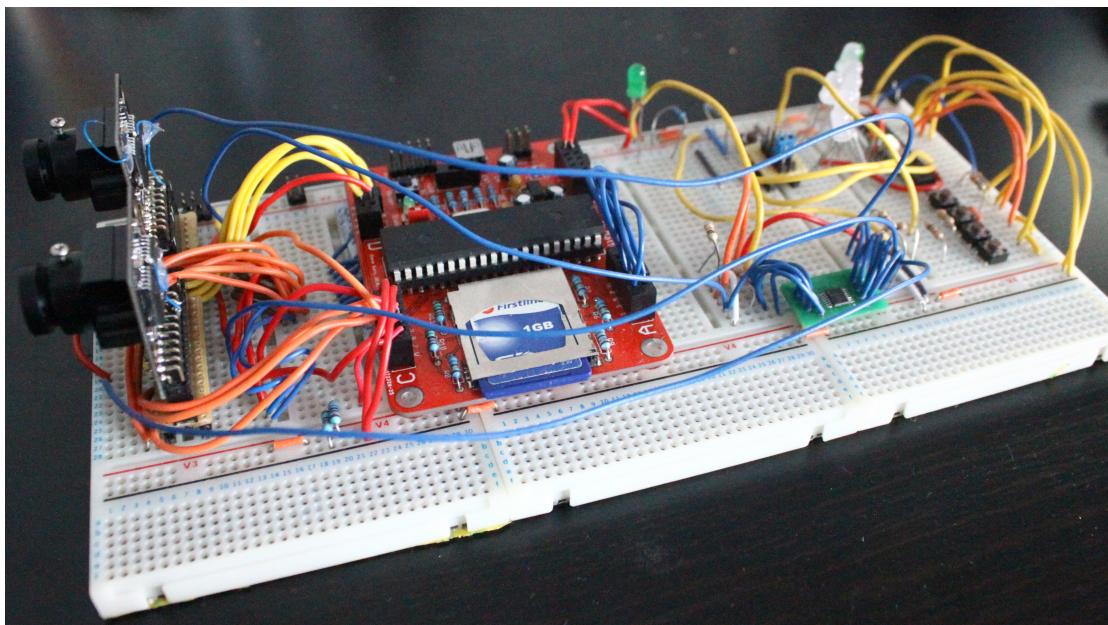


Figure 3.4: Prototype of Dual Camera operation.

3.3 Circuit and PCB Development

3.3.1 Il Matto Development

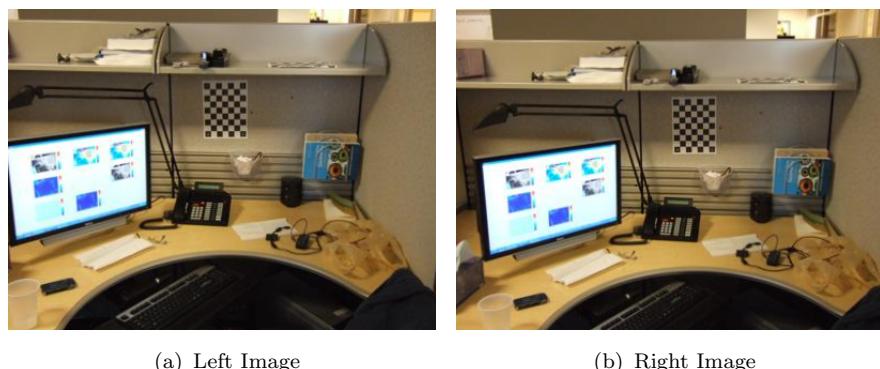
Figure B.2 shows the circuit diagram for the prototype. This uses the Il Matto development board for the main microcontroller. The prototype can be seen in figure 3.4.

Chapter 4

Investigation into Vision Algorithms

4.1 Comparison

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 of image to compare to of 50×50 pixels of the same part of the image.



(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 takes each sub image and subtracts the observed sub image 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.

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,

containing division and the necessity to square root a number in order to calculate the standard deviation. These operations are rarely implemented in hardware and are time consuming to carry out. They also require floating point registers and operates slowly on a microcontroller with a small amount of floating point registers.

4.1.4 Comparison

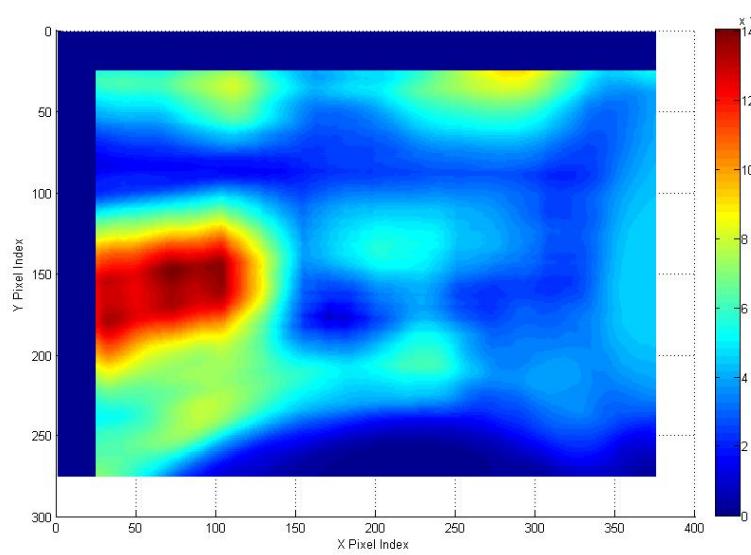
To compare these equations, a 50 by 50 image taken from the right picture was compared with the left image over the entire valid range. The coordinates on the graph give the centre pixel of the calculation.

Each of the graphs show the correct area being indentified 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 phone from the test figure. 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 \(2012\)](#)).

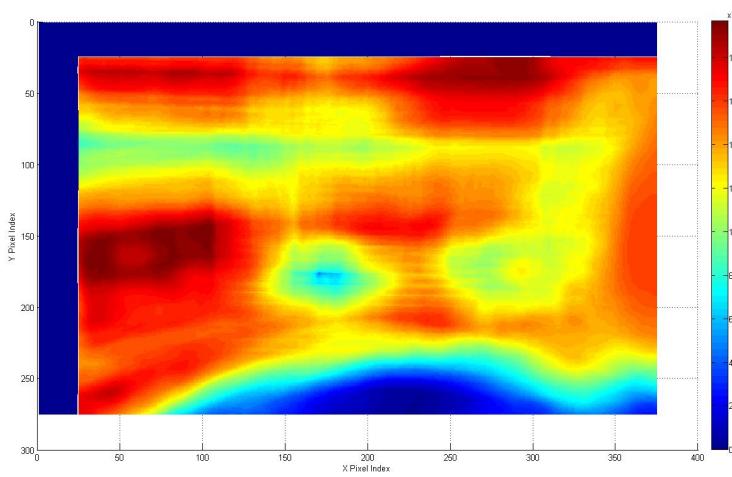
SAD results in figure 4.2(b) show large areas of matching. The actual match is at (170, 175) and a minimum does occur at this position as expected, of a value of 5.66×10^4 . However, along the bottom of the image, where a dark area occurs in the lower part of figures 4.1 below the desk, the SAD algorithm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

SSD shows matches in the same two areas: where a match should occur and the dark area beneath the desk. The minimum values 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 occurs at (225, 274). This, again, is a false match and is a downfall of this algorithm.

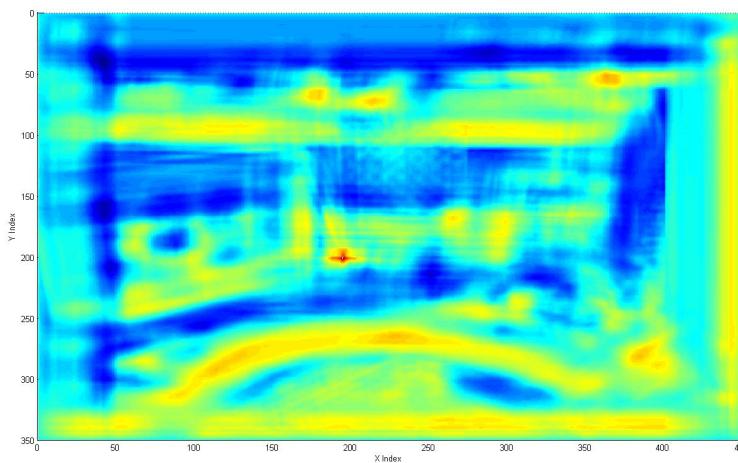
The NCC results are visible in figure 4.2(c). A match can be seen at coordinate (195, 201) with a peak value of 0.9654. The coordinate is different to the previous results because the cross correlation works over the boundary of the image creating more results. The dimensions of the image are 300×400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



(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

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 difference, the range of an object can be found given some characteristics of the camera. The following is a derivation of the equations used to calculate distance.

The problem is broken down into three parts:

1. Object is between the cameras (Figure 4.3)
2. Object is in left or right hand sides of both images (Figure 4.4)
3. Object is directly in front of a camera (Figure 4.5)

4.2.1.1 Object is between the Cameras

Derivation from [Mrovlje and Vrančić \(2008\)](#).

$$B = B_1 + B_2 = D \tan(\varphi_1) + D \tan(\varphi_2) \quad (4.4)$$

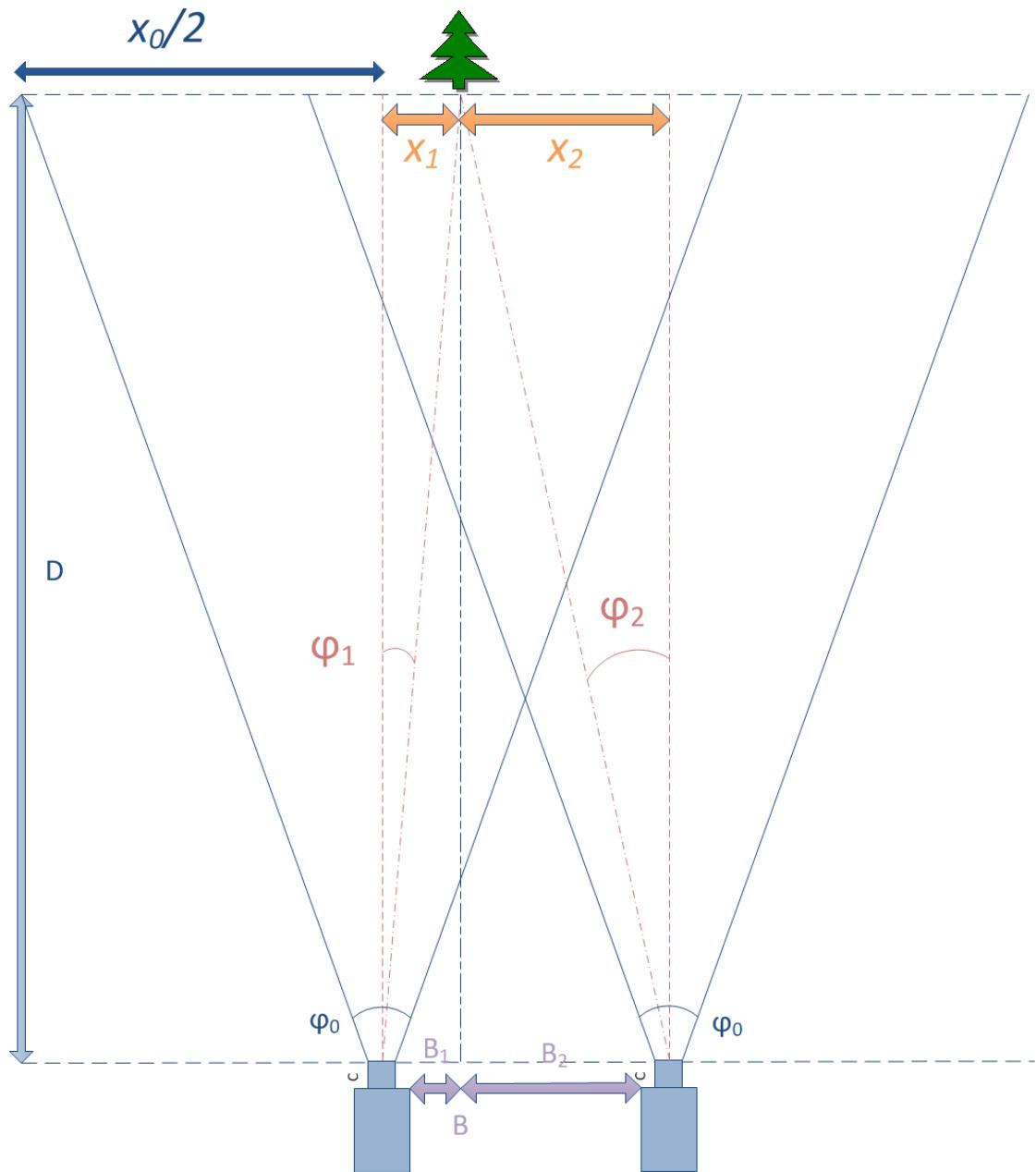


Figure 4.3: Problem 1 - Object is between the Cameras

$$D = \frac{B}{\tan(\varphi_1) + \tan(\varphi_2)} \quad (4.5)$$

$$D \tan\left(\frac{\varphi_0}{2}\right) = x_0/2 \quad (4.6)$$

$$D \tan(\varphi_1) = x_1 \quad (4.7)$$

Dividing (4.7) by (4.6)

$$\frac{\tan(\varphi_1)}{\tan(\frac{\varphi_0}{2})} = \frac{2x_1}{x_0} \quad (4.8)$$

$$\tan(\varphi_1) = \frac{2x_1 \tan(\frac{\varphi_0}{2})}{x_0} \quad (4.9)$$

This can also be shown for the right camera:

$$\tan(\varphi_2) = \frac{-2x_2 \tan(\frac{\varphi_0}{2})}{x_0} \quad (4.10)$$

Substitution equations (4.9) and (4.10) into (4.5) gives

$$D = \frac{Bx_0}{2 \tan(\frac{\varphi_0}{2})(x_1 - x_2)} \quad (4.11)$$

4.2.1.2 Object is to the same side in each camera

Derivation is based on the derivation from Tjandranegara (2005). Using figure 4.4:

$$D \cdot \tan(\varphi_1) = x_1 \quad (4.12)$$

$$D \cdot \tan\left(\frac{\varphi_0}{2}\right) = \frac{x_0}{2} \quad (4.13)$$

$$\frac{\tan(\varphi_1)}{\tan(\frac{\varphi_0}{2})} = \frac{2x_1}{x_0} \quad (4.14)$$

$$\varphi_1 = \arctan\left(\frac{2x_1}{x_0} \tan\left(\frac{\varphi_0}{2}\right)\right) \quad (4.15)$$

and similarly

$$\varphi_2 = \arctan\left(\frac{2x_2}{x_0} \tan\left(\frac{\varphi_0}{2}\right)\right) \quad (4.16)$$

$$\theta = \varphi_2 - \varphi_1 \quad (4.17)$$

Using the sine equality rule:

$$\frac{R}{\sin(\frac{\pi}{2} - \varphi_2)} = \frac{B}{\sin(\theta)} \quad (4.18)$$

$$R = B \cdot \frac{\sin(\frac{\pi}{2} - \varphi_2)}{\sin(\theta)} = B \frac{\cos(\varphi_2)}{\sin(\theta)} \quad (4.19)$$

$$D = \cos(\varphi_1) \cdot R \quad (4.20)$$

Substituting (4.17) into (4.19), and then into (4.20):

$$D = B \cdot \frac{\cos(\varphi_2) \cdot \cos(\varphi_1)}{\sin(\varphi_2 - \varphi_1)} \quad (4.21)$$

Where φ_1 is defined in equation (4.15) and φ_2 is defined in equation (4.16).

4.2.1.3 Object is in front of a camera

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

$$D = B \tan\left(\frac{\pi}{2} - \varphi_2\right) \quad (4.22)$$

Where φ_2 can be found from equation 4.16.

4.2.1.4 Summary

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

Object is between the two cameras:

$$D = \frac{Bx_0}{2 \tan(\frac{\varphi_0}{2})(x_1 - x_2)} \quad (4.23)$$

Object is to the same side in both images:

$$D = B \cdot \frac{\cos(\varphi_2) \cdot \cos(\varphi_1)}{\sin(\varphi_2 - \varphi_1)} \quad (4.24)$$

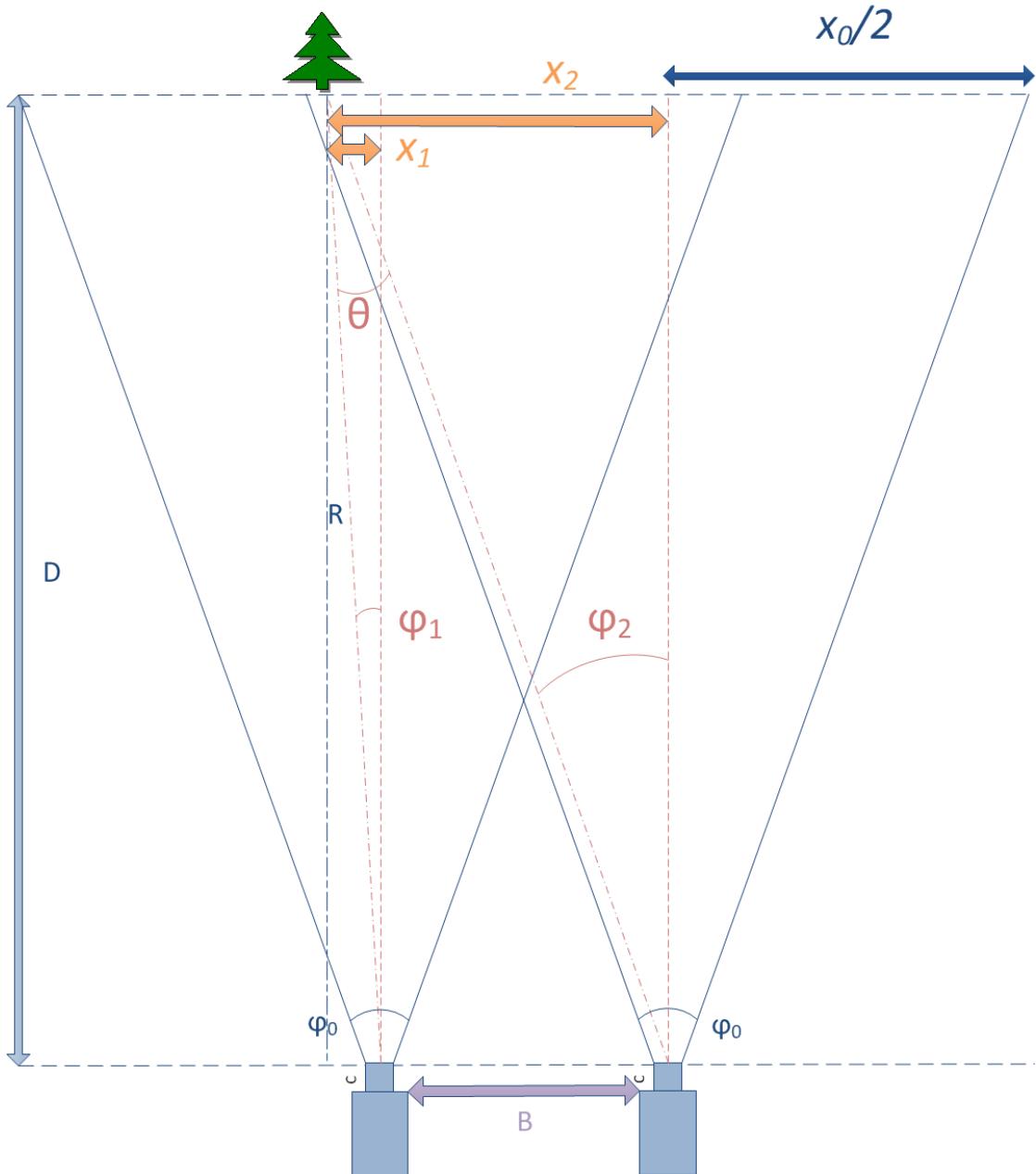


Figure 4.4: Problem 2 - Object is to the same side in both cameras

Object is directly in front of a camera:

$$D = B \tan \left(\frac{\pi}{2} - \varphi_2 \right) \quad (4.25)$$

Where φ_1 is defined in equation (4.15) and φ_2 is defined in equation (4.16).

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

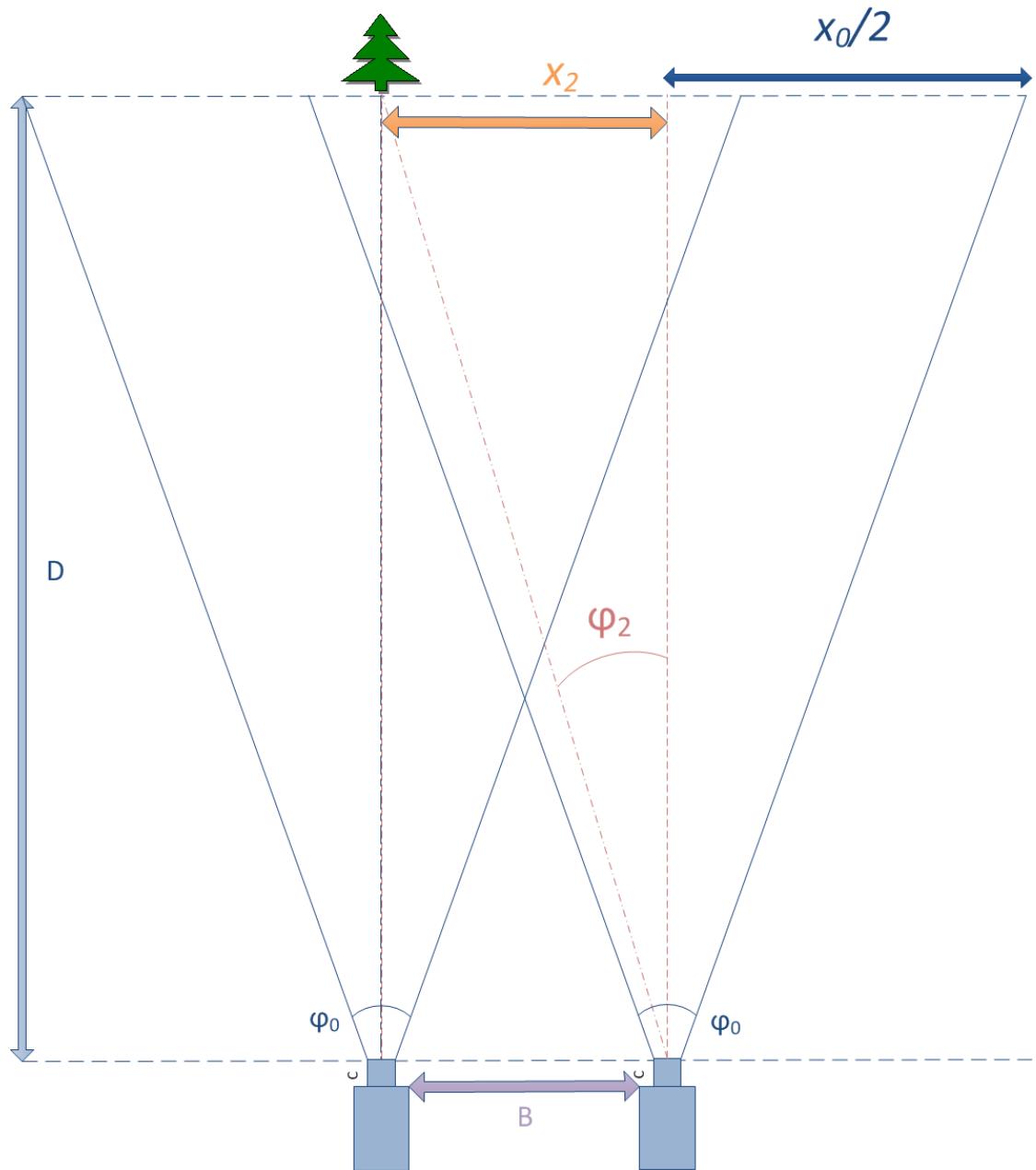


Figure 4.5: Problem 3 - Object is directly in front of a camera

Chapter 5

Conclusions and Further Work

So far, the project has addressed three aspects of the whole design: image acquisition, object matching a range finding. All these aspects have been developed separately on different platforms; on an ATMega644P or in MATLAB.

A full image can be read from a camera and stored on an SD card in a FAT32 filesystem. 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.

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

Range finding equations were then derived, which use the characteristics of the camera and the separation distance between them.

The next steps will be bringing the comparison and range equations together so as to be able to choose an object in a scene and find the range to said object. Object detection algorithms need to be looked into so that an object in view can be detected and correctly located.

Robot movement also needs development. An accurate method to move a distance will be implemented with a proportional or PID controller.

A printed circuit board will also be designed with an AVR32 as the microcontroller and all extra hardware on the board will be included. This will then mean all modules (image acquisition, comparison, range finding etc.) can be ported to the

AVR32 and the robot can be made mobile. Further parts can then be developed such as the occupancy map and search algorithms.

All modules will be tested before being implemented. The final hardware design and development is to be completed by 15th March 2013 and the project report hand-in date is 1st May 2013.

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

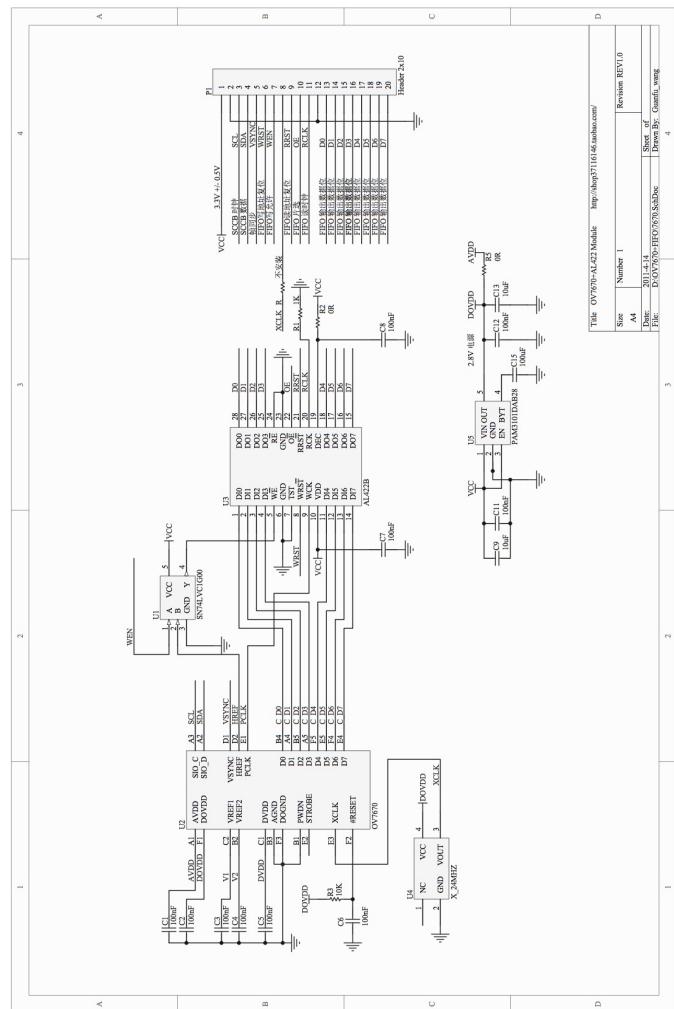


Figure B.1: The circuit diagram for the OV7670 breakout board

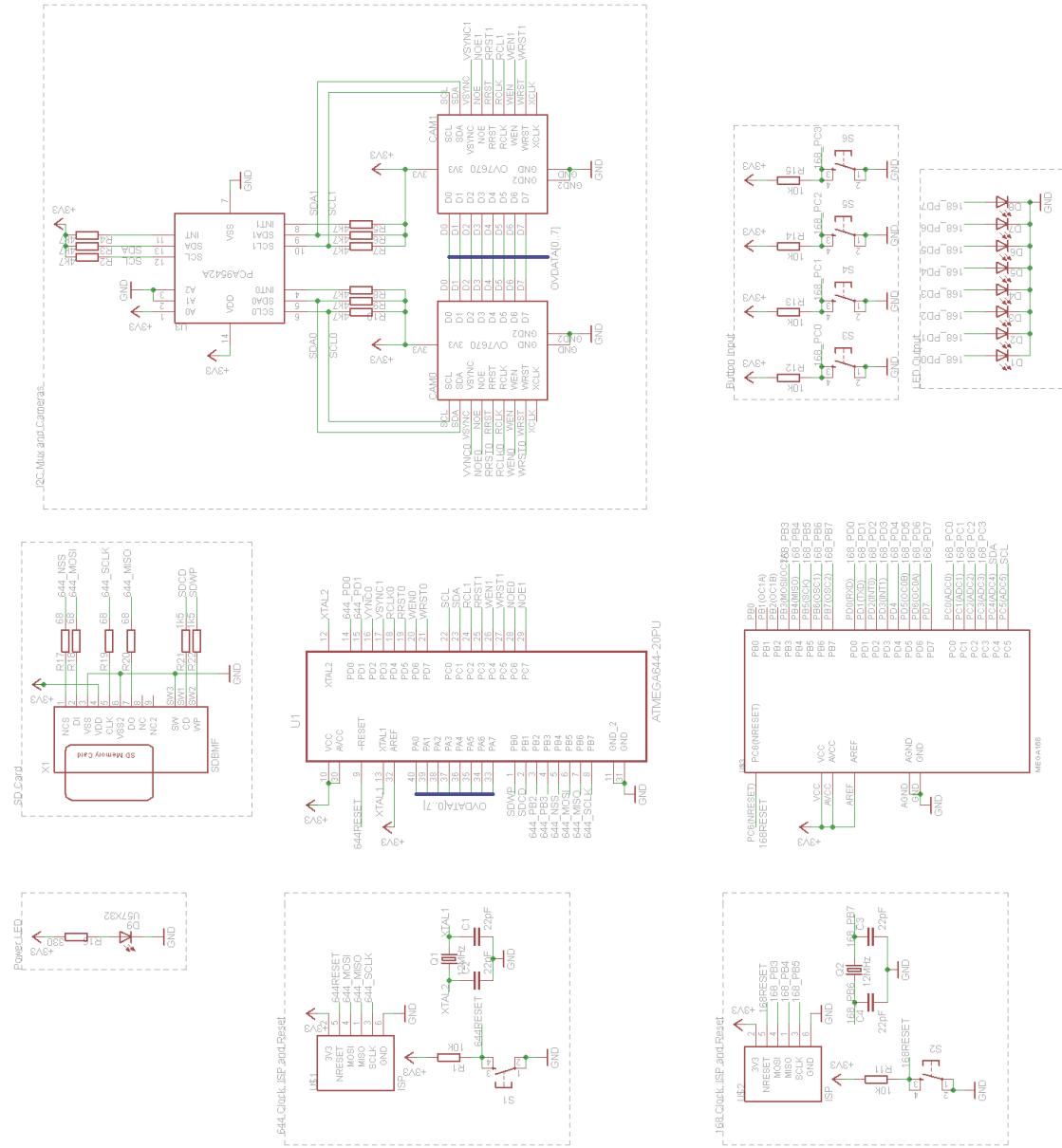


Figure B.2: The circuit diagram for Dual Cameras using the Il Matto Board

Appendix C

Bitmap File Format

C.1 Bitmap File Format

Table C.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 Bitamp 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 Independant 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 said in Table 3.1 due to omitting many optional fields

Table C.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 C.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 D

Source Code

D.1 C Code for AVR

D.1.1 Dual Camera Operation

D.1.1.1 main.c

..../Code/DualOV7670/main.c

```
1  /*
2   * DualOV7670.c
3   *
4   * Created: 09/11/2012 11:43:13
5   * Author: hl13g10
6   */
7  #include "Config.h"

8

11 //static FILE mystdout = FDEV_SETUP_STREAM(File_Write_Printf, NULL,
12     _FDEV_SETUP_WRITE);
12 //FatFS Variables
13 FILINFO Finfo;
14 FATFS Fatfs[_VOLUMES];      /* File system object for each logical drive */
15 //FIL Files[2];           /* File object */
16 uint8_t StatusReg;
17 // char Line[100];          /* Console input buffer */
18 //char Buff[100];           /* Working buffer */
19 char ImageRName[20];
20 char ImageLName[20];
21 #define STATUS_OKAY        0x01
22 #define STATUS_SDOokay     0x02
23 #define STATUS_CAM00okay    0x04
24 #define STATUS_CAM10okay    0x08
```

```

25 #define STATUS_READY      0x10
26 #define STATUS_CAPTURING  0x20
27 #define STATUS_Exit_Bad   0x80

29 #define Button_Capture    0
30 #define Button_Exit       3
31 unsigned char UI_LEDs(uint8_t LED)
32 {
33     unsigned char mesbuf[TWI_BUFFER_SIZE];
34     mesbuf[0] = (0x15 << TWI_ADR_BITS) | (FALSE << TWI_READ_BIT);
35     mesbuf[1] = 0x10;
36     mesbuf[2] = LED;
37     TWI_Start_Transceiver_With_Data(mesbuf, 3);
38     while(TWI_Transceiver_Busy()) ;
39     return TWI_statusReg.lastTransOK;
40 }
41 unsigned char UI.Buttons()
42 {
43     unsigned char messageBuf[TWI_BUFFER_SIZE]; //Initialise a buffer
44     messageBuf[0] = (0x15<<TWI_ADR_BITS) | (FALSE<<TWI_READ_BIT); // The first
        byte must always consist of General Call code or the TWI slave address.
45     messageBuf[1] = 0x20;           // The first byte is used for the command
46     TWI_Start_Transceiver_With_Data( messageBuf, 2 );
47     _delay_us(250);
48     // Request/collect the data from the Slave
49     messageBuf[0] = (0x15<<TWI_ADR_BITS) | (TRUE<<TWI_READ_BIT); // The first
        byte must always consist of General Call code or the TWI slave address.
50     TWI_Start_Transceiver_With_Data( messageBuf, 2 );

52     // Get the received data from the transceiver buffer
53     TWI_Get_Data_From_Transceiver( messageBuf, 2 );
54     return messageBuf[1];
55 }
56 ISR(TIMERO0_COMPA_vect)
57 {
58     disk_timerproc(); /* Drive timer procedure of low level disk I/O module */
59 //     if(!TWI_statusReg.lastTransOK) //if the last TWI transmission failed,
        reset the protocol
60 //     TWI_Start_Transceiver();
61 //     if(!TWI_Transceiver_Busy())
62 //         UI_LEDs(StatusReg);
63 }
64 int main(void)
65 {
66     unsigned long int a = 0;
67     uint8_t b = 0;
68     FRESULT fr;
69     uint8_t PhotoCount = 0;
70     TWI_Master_Initiate();
71     IO_Init();
72     sei();
73     PCA9542A_Init();

75     StatusReg = STATUS_OKAY;
76     UI_LEDs(StatusReg);

79     fr = f_mount(0, &Fatfs[0]);

```

```
80     if(fr != FR_OK)
81     {
82         StatusReg |= (STATUS_Exit_Bad);
83         StatusReg &= ~(STATUS_OKAY);
84         UI_LEDs(StatusReg);
85         return 0;
86     }
87     else
88         StatusReg |= STATUS_SDOkay;
89     UI_LEDs(StatusReg);

90     fr = f_open(&Files[0], "/log.txt", FA_WRITE|FA_CREATE_ALWAYS);
91     if(fr != FR_OK)
92     {
93         StatusReg |= (STATUS_Exit_Bad);
94         StatusReg &= ~(1<<STATUS_SDOkay) | (1<<STATUS_OKAY);
95         UI_LEDs(StatusReg);
96         return 0;
97     }
98     UI_LEDs(StatusReg);

99     f_close(&Files[0]);
100    f_open(&Files[0], "/log.txt", FA_WRITE);
101    //stdout = &mystdout;
102    b = MCUSR;
103    MCUSR = 0;
104    f_write(&Files[0],"Il Matto Dual Camera\n", sizeof("Il Matto Dual Camera\n")
105          , &a);

106    /*f_write(&Files[0], "System Startup Complete.\n", 26, &a);*/

107    PCA9542A_SetChannel(CH1);
108    b = OV7670_init();
109    if(b == 0)
110        StatusReg |= STATUS_CAM1Okay;
111    PCA9542A_SetChannel(NO_SELECT);
112    UI_LEDs(StatusReg);
113    sprintf(Buff, "OV7670_1 Initialise result : %d\n", b);
114    f_write(&Files[0], &Buff, 33, &a);

115    PCA9542A_SetChannel(CH0);
116    b = OV7670_init();
117    if(b == 0)
118        StatusReg |= STATUS_CAM0Okay;
119    UI_LEDs(StatusReg);
120    PCA9542A_SetChannel(NO_SELECT);
121    sprintf(Buff, "OV7670_0 Initialise result : %d\n", b);
122    f_write(&Files[0], &Buff, 33, &a);
123    FIFO_init();

124    //f_close(&Files[0]);
125    StatusReg |= STATUS_READY;
126    UI_LEDs(StatusReg);
127    _delay_ms(250);
128    uint8_t Input;

129    while(1)
130    {
131
```

```

137     Input = (~UI.Buttons() & 0x0F); //Data is received negative
138     if(Input)//if a button has been pressed
139     {
140         _delay_ms(250);
141         sprintf(Buff, "Button Received : %d\n", Input);
142         f_write(&Files[0], Buff, 21, &a);

144         StatusReg &= ~(STATUS_READY); //no longer ready

146         switch(Input)
147         {
148             case (1<<Button_Capture):
149                 StatusReg |= STATUS_CAPTURING;
150                 UI_LEDs(StatusReg);
151                 //Reset both buffers
152                 FIFO_Reset(0);
153                 FIFO_Reset(1);
154                 f_write(&Files[0], "Capturing Images...\n", 20, &a);
155                 LoadImagesToBuffer(); //Load both images

157                 //Create Bitmap for image 0
158                 //PSTR("Image_r.bmp");

160                 f_open(&Files[1], "Image_r.bmp", FA_CREATE_ALWAYS | FA_WRITE);
161                 f_write(&Files[0], "Created image0 file.\n", 22, &a);
162                 f_lseek(&Files[1], BMPFileSize);
163                 f_lseek(&Files[1], 0);
164                 f_close(&Files[1]);
165                 f_write(&Files[0], "Extended image0 file.\n", 22, &a);

167                 //Create Bitmap for image 1
168                 f_open(&Files[1], "image_1.bmp", FA_CREATE_ALWAYS | FA_WRITE);
169                 f_write(&Files[0], "Created image1 file.\n", 22, &a);
170                 f_lseek(&Files[1], BMPFileSize);
171                 f_lseek(&Files[1], 0);
172                 f_close(&Files[1]);
173                 f_write(&Files[0], "Extended image1 file.\n", 22, &a);
174                 //Get image 0
175                 f_open(&Files[1], "Image_r.bmp", FA_WRITE);
176                 while (2 == GetImageIfAvailable(&Files[1], 0)) ;
177                 f_close(&Files[1]);
178                 f_write(&Files[0], "Captured image0.\n", 17, &a);
179                 //get image 1
180                 f_open(&Files[1], "image_1.bmp", FA_WRITE);
181                 while (2 == GetImageIfAvailable(&Files[1], 1)) ;
182                 f_close(&Files[1]);
183                 f_write(&Files[0], "Captured image1.\n", 17, &a);
184                 StatusReg |= STATUS_READY;
185                 StatusReg &= ~STATUS_CAPTURING;
186                 UI_LEDs(StatusReg);
187                 break; //break case(1<<ButtonCapture)

189             case (1<<Button_Exit):
190                 f_write(&Files[0], "\nSystem Exiting...\n", 19, &a);
191                 f_close(&Files[0]); //close log file

193             StatusReg = 0x41;
194             UI_LEDs(StatusReg);

```

```
195         return 0;//Q
196     } //End switch
197 } //End if(Input)
198 else
199 {
200     StatusReg |= STATUS_READY;
201     UI_LEDs(StatusReg);
202     _delay_ms(250); //wait
203 } //end else(Input)
204 } //End while(1)
205 } //End Main
```

D.1.1.2 Bitmap.h

..../Code/DualOV7670/Bitmap.h

```
1  /*  
2   * Bitmap.h  
3   *  
4   * Created: 29/10/2012 11:31:11  
5   * Author: hslovett  
6   */  
  
9  #ifndef BITMAP_H_  
10 #define BITMAP_H_  
  
12 #define BMPHEADERSIZE 14  
13 #define DIBHEADERSIZE 124 //v5  
14 #define FILESIZE 153738  
  
16 #include "ff.h"  
17 #include "Config.h"  
  
20 FRESULT WriteBMPHeader(FIL *File);  
21 FRESULT WriteDIBHeader(FIL *File);  
  
24 #endif /* BITMAP_H_ */
```

D.1.1.3 Bitmap.c

..../Code/DualOV7670/Bitmap.c

```
1  /*
2   * Bitmap.c
3   * Contains Methods to write the Bitmap and DIB Header. File must already be
4   * open.
5   * Created: 29/10/2012 11:30:58
6   * Author: Henry Lovett (hl13g10@ecs.soton.ac.uk)
```

```

6   */
7 #include "Bitmap.h"
8
9 uint8_t DIBHead[DIBHEADERSIZE] = { 0x7C, 0x00, 0x00, 0x00, //Number of bytes
10    0x40, 0x01, 0x00, 0x00, //Width - 320
11    0xF0, 0x00, 0x00, 0x00, //Height - 240
12    0x01, 0x00, //Planes
13    0x10, 0x00, //Bits per Pixel
14    0x03, 0x00, 0x00, 0x00, //Compression
15    0x00, 0x86, 0x25, 0x00, //Size of Raw Data
16    0x13, 0x0B, 0x00, 0x00, //Horizontal Resolution
17    0x13, 0x0B, 0x00, 0x00, //Vertical Resolution
18    0x00, 0x00, 0x00, 0x00, //Colours in Palette
19    0x00, 0x00, 0x00, 0x00, //Important Colours
20    0x00, 0xF8, 0x00, 0x00, //Red Mask
21    0xE0, 0x07, 0x00, 0x00, //Green Mask
22    0x1F, 0x00, 0x00, 0x00, //Blue Mask
23    0x00, 0x00, 0x00, 0x00, //Alpha Mask
24    0x01, 0x00, 0x00, 0x00, //Colour Space Type
25    0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
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}; //Reserved
41
42 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 };
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63

```

```
64     FRESULT f;  
  
66     f_lseek(File, BMPHEADERSIZE); //place just after the bitmap header  
67     f = f_write(File, DIBHead, DIBHEADERSIZE, &p);  
68     return f;  
69 }
```

D.1.1.4 Config.h

..../Code/DualOV7670/Config.h

```

44 #define SD_WP      PB0
45 #define SD_CD      PB1
46 // #define          PB2
47 // #define          PB3
48 #define SPI_nSS_SD    PB4
49 #define SPI_MOSI     PB5
50 #define SPI_MISO     PB6
51 #define SPI_SCK      PB7
52 ///////////////////////////////////////////////////////////////////
53 //  Port C
54 ///////////////////////////////////////////////////////////////////
55 #define TWI_SCL      PC0
56 #define TWI_SDA      PC1
57 #define FIFO_RCLK_1   PC2
58 #define FIFO_nRRST_1  PC3
59 #define FIFO_WEN_1    PC4
60 #define FIFO_WRST_1   PC5
61 #define FIFO_nOE_0    PC6
62 #define FIFO_nOE_1    PC7
63 ///////////////////////////////////////////////////////////////////
64 //  Port D
65 ///////////////////////////////////////////////////////////////////
66 #define USARTO_RX    PD0
67 #define USARTO_TX    PD1
68 #define OV7670_VSYNC_0 PD2 //MUST BE AN INTERRUPT PIN
69 #define OV7670_VSYNC_1 PD3 //MUST BE AN INTERRUPT PIN
70 #define FIFO_RCLK_0   PD4
71 #define FIFO_nRRST_0  PD5
72 #define FIFO_WEN_0    PD6
73 #define FIFO_WRST_0   PD7

77 #endif /* CONFIG_H_ */

```

D.1.1.5 Config.c

..../Code/DualOV7670/Config.c

```

1 /*
2  * Config.c
3  *
4  * Contains Global Methods and initialisations
5  *
6  * Created: 25/10/2012 21:59:06
7  * Author: hslovett
8  */
9
10 #include "Config.h"
11 #include <avr/io.h>
12 void IO_Init(void)
13 {
14     //initialise timer 0 to interrupt every 10 ms
15     TIMSK0 |= (1 << OCIE0A);

```

```

16     TCCROA |= (1 << WGM01);
17     OCROA = 117; //10ms interrupt at 12MHz
18     TCCROB |= (1 << CS02) | (1 << CS00);

21     DDRA = 0x00;
22     //PORTB = 0xBF;
23     DDRC = 0xFC;
24     DDRD = 0xF2;

28 //set int0 and int1 to trigger on falling edge
29 EIMSK = (1 << INT0) | (1 << INT1);           //Enable INT0 and INT1
30 EICRA = (1 << ISC01) | (1 << ISC11);         //Trigger INT0 and INT1 on the
31     falling edge
}

```

D.1.1.6 DualCameras.h

..../Code/DualOV7670/DualCameras.h

```

1  /*
2  * DualCameras.h
3  *
4  * Created: 10/11/2012 15:19:52
5  * Author: hslovett
6  */

9 #ifndef DUALCAMERAS_H_
10 #define DUALCAMERAS_H_

12 #include "Config.h"

14 // Constants
16 // Globals
25 const char default_settings[SETTINGS_LENGTH][2];
26 volatile uint8_t VSYNC_0_Count;
27 volatile uint8_t VSYNC_1_Count;
31 unsigned char OV7670_init(void);           //Initialises Camera
32 void FIFO_init(void);                    //Initialises Buffer

```

```

33     uint8_t GetImageIfAvailable(FIL *File, uint8_t CameraID);
34     void LoadImagesToBuffer(void);
35     unsigned char rdOV7670Reg(unsigned char regID, unsigned char *regData);
36     unsigned char OV7670_SCCB_init(void);
37     void FIFO_Reset(uint8_t CameraID);
38     ///////////////////////////////////////////////////////////////////
39 // Pins & Macros
40 ///////////////////////////////////////////////////////////////////
41 #define FIFO_RCLK_1    PC2
42 #define FIFO_nRRST_1   PC3
43 #define FIFO_WEN_1     PC4
44 #define FIFO_WRST_1    PC5
45 #define FIFO_nOE_0     PC6
46 #define FIFO_nOE_1     PC7

48 #define FIFO_RCLK_1_SET { PORTC |= (1 << FIFO_RCLK_1); }
49 #define FIFO_RCLK_1_CLR { PORTC &= ~(1 << FIFO_RCLK_1); }
50 #define FIFO_nRRST_1_SET { PORTC |= (1 << FIFO_nRRST_1); }
51 #define FIFO_nRRST_1_CLR { PORTC &= ~(1 << FIFO_nRRST_1); }
52 #define FIFO_WEN_1_SET { PORTC |= (1 << FIFO_WEN_1); }
53 #define FIFO_WEN_1_CLR { PORTC &= ~(1 << FIFO_WEN_1); }
54 #define FIFO_WRST_1_SET { PORTC |= (1 << FIFO_WRST_1); }
55 #define FIFO_WRST_1_CLR { PORTC &= ~(1 << FIFO_WRST_1); }
56 #define FIFO_nOE_0_SET { PORTC |= (1 << FIFO_nOE_0); }
57 #define FIFO_nOE_0_CLR { PORTC &= ~(1 << FIFO_nOE_0); }
58 #define FIFO_nOE_1_SET { PORTC |= (1 << FIFO_nOE_1); }
59 #define FIFO_nOE_1_CLR { PORTC &= ~(1 << FIFO_nOE_1); }

62 #define FIFO_RCLK_0    PD4
63 #define FIFO_nRRST_0   PD5
64 #define FIFO_WEN_0     PD6
65 #define FIFO_WRST_0    PD7

67 #define FIFO_RCLK_0_SET { PORTD |= (1 << FIFO_RCLK_0); }
68 #define FIFO_RCLK_0_CLR { PORTD &= ~(1 << FIFO_RCLK_0); }
69 #define FIFO_nRRST_0_SET { PORTD |= (1 << FIFO_nRRST_0); }
70 #define FIFO_nRRST_0_CLR { PORTD &= ~(1 << FIFO_nRRST_0); }
71 #define FIFO_WEN_0_SET { PORTD |= (1 << FIFO_WEN_0); }
72 #define FIFO_WEN_0_CLR { PORTD &= ~(1 << FIFO_WEN_0); }
73 #define FIFO_WRST_0_SET { PORTD |= (1 << FIFO_WRST_0); }
74 #define FIFO_WRST_0_CLR { PORTD &= ~(1 << FIFO_WRST_0); }

76 ///////////////////////////////////////////////////////////////////
77 //Camera Register Address definitions
78 ///////////////////////////////////////////////////////////////////
79 #define OV_GAIN      0x00 //Gain Control Setting - ACG[7:0]
80 #define OV_BLUE     0x01 //Blue Channel Gain
81 #define OV_RED      0x02 //Red Channel Gain
82 #define OV_VREF     0x03 //Vertical Frame Control & ACG[9:8]
83 #define OV_COM1     0x04 //CCIR656 enable, AEC low bits (AECHH, AECH)
84 #define OV_BAVE     0x05 //U/B Average level - AUTO UPDATED
85 #define OV_GbAVE    0x06 //Y/Gb Average Level - AUTO UPDATED
86 #define OV_AECHH    0x07 //Exposure value [15:10] (AECH, COM1)
87 #define OV_RAVE     0x08 //V/R Average level - AUTO UPDATED
88 #define OV_COM2     0x09 //Soft Sleep, Output drive capability
89 #define OV_PID      0x0A //Product ID MSB Read only
90 #define OV_VER      0x0B //Product ID LSB Read Only

```

```

91 #define OV_COM3      0x0C //Output data MSB/LSB swap + other stuff
92 #define OV_COM4      0x0D //Average values - MUST BE SAME AS COM17
93 #define OV_COM5      0x0E //RESERVED
94 #define OV_COM6      0x0F //COM6
95 #define OV_AECH      0x10 //Exposure value [9:2] (see AECHH, COM1)
96 #define OV_CLKRC     0x11 //Internal Clock options
97 #define OV_COM7      0x12 //RESET, Output format
98 #define OV_COM8      0x13 //Common control 8
99 #define OV_COM9      0x14 //Automatic Gain Ceiling
100 #define OV_COM10     0x15 //PCLK, HREF and VSYNC options
101 #define OV_RSVD      0x16 //RESERVED
102 #define OV_HSTART    0x17 //Output format Horizontal Frame start
103 #define OV_HSTOP     0x18 //Output format Horizontal Frame end
104 #define OV_VSTRT     0x19 //Output format Vertical Frame start
105 #define OV_VSTOP     0x1A //Output format Vertical Frame Stop
106 #define OV_PSHFT     0x1B //Pixel Delay Select
107 #define OV_MIDH      0x1C //Manufacturer ID MSB - READ ONLY
108 #define OV_MIDL      0x1D //Manufacturer ID LSB - READ ONLY
109 #define OV_MVFP      0x1E //Mirror / VFlip Enable
110 #define OV_LAEC      0x1F //RESERVED
111 #define OV_ADCCTR0   0x20 //ADC Control
112 #define OV_ADCCTR1   0x21 //RESERVED
113 #define OV_ADCCTR2   0x22 //RESERVED
114 #define OV_ADCCTR3   0x23 //RESERVED
115 #define OV_AEW       0x24 //ACG/AEC Stable Operating Region Upper Limit
116 #define OV_AEB       0x25 //ACG/AEC Stable Operation Region Lower Limit
117 #define OV_VPT       0x26 //ACG/AEC Fast Mode Operation Region
118 #define OV_BBIAS     0x27 //B Channel Signal Output Bias
119 #define OV_GbBIAS    0x28 //Gb Channel Output Bias
120 #define OV_RSVD1     0x29 //RESERVED
121 #define OV_EXHCH     0x2A //Dummy Pixel Insert MSB
122 #define OV_EXHCL     0x2B //Dummy Pixel Insert LSB
123 #define OV_RBIAS     0x2C //R Channel Signal Output Bias
124 #define OV_ADVFL     0x2D //LSB of insert dummy line in vertical direction
125 #define OV_AdVFH     0x2E //MSB of insert dummy line in vertical direction
126 #define OV_YAVE      0x2F //Y/G Channel Average Value
127 #define OV_HSYST     0x30 //HSYNC Rising Edge Delay (low 8 bits)
128 #define OV_HSYEN     0x31 //HSYNCE Falling Edge Delay (low 8 bits)
129 #define OV_HREF      0x32 //HREF Control
130 #define OV_CHLF      0x33 //Array Current Control - RESERVED
131 #define OV_ARBLM     0x34 //Array Reference Control - RESERVED
132 #define OV_RSVD2     0x35 //RESERVED
133 #define OV_RSVD3     0x36 //RESERVED
134 #define OV_ADCCTRL   0x37 //ADC Control - RESERVED
135 #define OV_ACOM      0x38 //ADC and Analog Common Mode Control - RESERVED
136 #define OV_OFON      0x39 //ADC Offset Control
137 #define OV_TSLB      0x3A //Line Buffer Test Option
138 #define OV_COM11     0x3B //COM11
139 #define OV_COM12     0x3C //COM12
140 #define OV_COM13     0x3D //COM13
141 #define OV_COM14     0x3E //COM14
142 #define OV_EDGE      0x3F //Edge Detection Adjustment
143 #define OV_COM15     0x40 //COM15
144 #define OV_COM16     0x41 //COM16
145 #define OV_COM17     0x42 //COM17
146 #define OV_AWBC1     0x43
147 #define OV_AWBC2     0x44
148 #define OV_AWBC3     0x45

```

```
149 #define OV_AWBC4      0x46
150 #define OV_AWBC5      0x47
151 #define OV_AWBC6      0x48
152 #define OV_RSVD4      0x49
153 #define OV_RSVD5      0x40
154 #define OV_RSVD6      0x4A
155 #define OV_REG4B      0x4B
156 #define OV_DNSTH      0x4C
157 #define OV_RSVD7      0x4D
158 #define OV_RSVD8      0x4E
159 #define OV_MTX1       0x4F
160 #define OV_MTX2       0x50
161 #define OV_MTX3       0x51
162 #define OV_MTX4       0x52
163 #define OV_MTX5       0x53
164 #define OV_MTX6       0x54
165 #define OV_BRIGHT      0x55
166 #define OV CONTRAS     0x56
167 #define OV CONTRASCNTR 0x57
168 #define OV_MTXS        0x58
169 #define OV_RSVD9       0x59
170 #define OV_RSVD9_1     0x5A
171 #define OV_RSVD9_2     0x5B
172 #define OV_RSVD9_3     0x5C
173 #define OV_RSVD9_4     0x5D
174 #define OV_RSVD9_5     0x5E
175 #define OV_RSVD9_6     0x5F
176 #define OV_RSVD10      0x60
177 #define OV_RSVD11      0x61
178 #define OV_LCC1        0x62
179 #define OV_LCC2        0x63
180 #define OV_LCC3        0x64
181 #define OV_LCC4        0x65
182 #define OV_LCC5        0x66
183 #define OV_MANU        0x67
184 #define OV_MANV        0x68
185 #define OV_GFIX        0x69
186 #define OV_GGAIN        0x6A
187 #define OV_DBLV        0x6B
188 #define OV_AWBCTR3     0x6C
189 #define OV_AWBCTR2     0x6D
190 #define OV_AWBCTR1     0x6E
191 #define OV_AWBCTR0     0x6F
192 #define OV_SCALING_XSC 0x70
193 #define OV_SCALING_YSC 0x71
194 #define OV_SCALING_DCWCTR 0x72
195 #define OV_SCALING_PCLK_DIV 0x73
196 #define OV_REG74        0x74
197 #define OV_REG75        0x75
198 #define OV_REG76        0x76
199 #define OV_REG77        0x77
200 #define OV_RSVD12      0x78
201 #define OV_RSVD13      0x79
202 #define OV_GAM1         0x7A
203 #define OV_GAM2         0x7B
204 #define OV_GAM3         0x7C
205 #define OV_GAM4         0x7D
206 #define OV_GAM5         0x7E
```

```
207 #define OV_GAM6      0x7F
208 #define OV_GAM7      0x80
209 #define OV_GAM8      0x81
210 #define OV_GAM9      0x82
211 #define OV_GAM10     0x83
212 #define OV_GAM11     0x84
213 #define OV_GAM12     0x85
214 #define OV_GAM13     0x86
215 #define OV_GAM14     0x87
216 #define OV_GAM15     0x88
217 #define OV_GAM16     0x89
218 #define OV_RSVD14    0x8A
219 #define OV_RSVD15    0x8B
220 #define OV_RSVD16    0x8C
221 #define OV_RSVD17    0x8D
222 #define OV_RSVD18    0x8E
223 #define OV_RSVD19    0x8F
224 #define OV_RSVD20    0x90
225 #define OV_RSVD21    0x91
226 #define OV_DM_LNL    0x92
227 #define OV_DM_LNH    0x93
228 #define OV_LCC6      0x94
229 #define OV_LCC7      0x95
230 #define OV_RSVD22    0x96
231 #define OV_RSVD23    0x97
232 #define OV_RSVD24    0x98
233 #define OV_RSVD25    0x99
234 #define OV_RSVD26    0x9A
235 #define OV_RSVD27    0x9B
236 #define OV_RSVD28    0x9C
237 #define OV_BD50ST   0x9D
238 #define OV_BD60ST   0x9E
239 #define OV_HIST0     0x9F
240 #define OV_HIST1     0xA0
241 #define OV_HIST2     0xA1
242 #define OV_HIST3     0xA2
243 #define OV_HIST4     0xA3
244 #define OV_HIST5     0xA4
245 #define OV_HIST6     0xA5
246 #define OV_HIST7     0xA6
247 #define OV_HIST8     0xA7
248 #define OV_HIST9     0xA8
249 #define OV_HIST10    0xA9
250 #define OV_HIST11    0xAA
251 #define OV_HIST12    0xAB
252 #define OV_STR_OPT   0xAC
253 #define OV_STR_R     0xAD
254 #define OV_STR_G     0xAE
255 #define OV_STR_B     0xAF
256 #define OV_RSVD28_1   0xB0
257 #define OV_RSVD29    0xB1
258 #define OV_RSVD30    0xB2
259 #define OV_THL_ST    0xB3
260 #define OV_RSVD31    0xB4
261 #define OV_THL_DLT   0xB5
262 #define OV_RSVD32    0xB6
263 #define OV_RSVD33    0xB7
264 #define OV_RSVD34    0xB8
```

```

265 #define OV_RSVD35    0xB9
266 #define OV_RSVD36    0xBA
267 #define OV_RSVD37    0xBB
268 #define OV_RSVD38    0xBC
269 #define OV_RSVD39    0xBD
270 #define OV_AD_CHB    0xBE
271 #define OV_AD_CHR    0xBF
272 #define OV_AD_CHGb   0xC0
273 #define OV_AD_CHGr   0xC1
274 #define OV_RSVD40    0xC2
275 #define OV_RSVD41    0xC3
276 #define OV_RSVD42    0xC4
277 #define OV_RSVD43    0xC5
278 #define OV_RSVD44    0xC6
279 #define OV_RSVD45    0xC7
280 #define OV_RSVD46    0xC8
281 #define OV_SATCTR   0xC9

283 #endif /* DUALCAMERAS_H_ */

```

D.1.1.7 DualCameras.c

..../Code/DualOV7670/DualCameras.c

```

1  /*
2  *  DualCameras.c
3  *
4  *  Created: 10/11/2012 15:20:03
5  *  Author: hl13g10
6  */
7

8 #include "DualCameras.h"

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_VSTRT, 0x02},
19 {OV_VSTOP, 0x7b}, //0x7a,
20 {OV_VREF, 0x06}, //0xa,
21 {OV_COM3, 0x00},
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 {Oxa2, 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_MT XS, 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 {0V_BD50ST, 0x4c},  
150 {0V_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, 0x0a},  
159 {0xc8, 0x80},  
160 {0x79, 0x0b},  
161 {0xc8, 0x01},  
162 {0x79, 0x0c},  
163 {0xc8, 0x0f},  
164 {0x79, 0x0d},  
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 {0V_COM2, 0x03},  
176 {0V_BRIGHT, 0x00},  
177 {0V_CONTRAS, 0x40},  
178 {0V_COM11, 0x42}, //0x82, //0xc0, //0xc2, //night mode  
180 };  
  
184 //ISR for controlling WEN.  
185 ISR(INT0_vect)  
186 {  
187     //printf("ISR INT0 Entered\n");  
188     if (VSYNC_0_Count==1)//start a frame read  
189     {  
190         FIFO_WEN_0_SET;  
191         VSYNC_0_Count++;  
192     }  
193     else if (VSYNC_0_Count==2)//end a frame read  
194     {  
195         FIFO_WEN_0_CLR;  
196         VSYNC_0_Count++;  
197     }  
198     else if(VSYNC_0_Count == 3)  
199     {  
200         FIFO_WEN_0_CLR;  
201     }  
202     else  
203     {  
204         FIFO_WEN_0_CLR
```

```

205     VSYNC_0_Count = 0; //wait for a read to be started
206 }
207 }
208 //ISR for controlling WEN.
209 ISR(INT1_vect)
210 {
211     //printf("ISR INT1 Entered\n");
212     if (VSYNC_1_Count==1)//start a frame read
213     {
214         FIFO_WEN_1_SET;
215         VSYNC_1_Count++;
216     }
217     else if (VSYNC_1_Count==2)//end a frame read
218     {
219         FIFO_WEN_1_CLR;
220         VSYNC_1_Count++;
221     }
222     else if(VSYNC_1_Count == 3)
223     {
224         FIFO_WEN_1_CLR;
225     }
226     else
227     {
228         FIFO_WEN_1_CLR
229         VSYNC_1_Count = 0; //wait for a read to be started
230     }
231 }

233 //Write Register Method
234 unsigned char wrOV7670Reg(unsigned char regID, unsigned char regDat)
235 {
236     /* I2C Traffic Generated:
237      * S | OV_7670 + W | A | RegID | A | Data | A | P |
238      */
239 //I2C Interface
240 unsigned char messageBuf[TWI_BUFFER_SIZE];
241 messageBuf[0] = (OV7670_ADDR << TWI_ADR_BITS) | (FALSE<<TWI_READ_BIT); //The first byte must always consist of General Call code or the TWI slave address.
242 messageBuf[1] = regID; // The first byte is used for commands.
243 messageBuf[2] = regDat; // The second byte is used for the data.
244 TWI_Start_Transceiver_With_Data( messageBuf, 3 );

246 while(TWI_Transceiver_Busy()) ; //Wait for transceiver to clear

248 return TWI_statusReg.lastTransOK;
249 }

251 //Read Register Method
252 unsigned char rdOV7670Reg(unsigned char regID, unsigned char *regDat)
253 {
254     /* I2C Traffic Generated:
255      * S | OV_ADDR + W | A | RegID | A | P |
256      * S | OV_ADDR + R | A | Data | ~A | P |
257      */
258 //I2C Interface
259     unsigned char messageBuf[TWI_BUFFER_SIZE]; //Initialise a buffer

```

```
260     messageBuf[0] = (OV7670_ADDR<<TWI_ADR_BITS) | (FALSE<<TWI_READ_BIT); // The
261     first byte must always consist of General Call code or the TWI slave
262     address.
263     messageBuf[1] = regID;           // The first byte is used for Address
264     Pointer.
265     TWI_Start_Transceiver_With_Data( messageBuf, 2 );
266
267     // Request/collect the data from the Slave
268     messageBuf[0] = (OV7670_ADDR<<TWI_ADR_BITS) | (TRUE<<TWI_READ_BIT); // The
269     first byte must always consist of General Call code or the TWI slave
270     address.
271     TWI_Start_Transceiver_With_Data( messageBuf, 2 );
272
273     // Get the received data from the transceiver buffer
274     TWI_Get_Data_From_Transceiver( messageBuf, 2 );
275     *regDat = messageBuf[1];
276     return TWI_statusReg.lastTransOK;
277 }
278
279
280 unsigned char OV7670_init()
281 {
282     uint8_t i = 0;
283     if(0==wrOV7670Reg(OV_COM7, 0x80)) //Reset Camera
284     {
285         return 1;
286     }
287     _delay_ms(10);
288     for(i=0; i<SETTINGS_LENGTH; i++)
289     {
290         if( 0==wrOV7670Reg(default_settings[i][0], default_settings[i][1] ))
291         {
292             return 1;
293         }
294         _delay_ms(1);
295     }
296
297     return 0;
298 }
299
300 void FIFO_init( void )
301 {
302     //disable both outputs
303     FIFO_nOE_0_SET;
304     FIFO_nOE_1_SET;
305     //Reset Buffer 0
306     FIFO_WRST_0_CLR;
307     FIFO_RCLK_0_CLR;
308     //FIFO_nOE_0_CLR;
309     FIFO_nRRST_0_SET;
310     FIFO_WEN_0_CLR;
311     _delay_us(10);
312     FIFO_RCLK_0_SET;
313     _delay_us(10);
314     FIFO_RCLK_0_CLR;
315     FIFO_nRRST_0_CLR;
316     _delay_us(10);
317     FIFO_RCLK_0_SET;
```

```
313     _delay_us(10);
314     FIFO_RCLK_0_CLR;
315     FIFO_nRRST_0_SET;
316     _delay_us(10);
317     FIFO_WRST_0_SET;

319     //Reset Buffer 1
320     FIFO_WRST_1_CLR;
321     FIFO_RCLK_1_CLR;
322     //FIFO_nOE_1_CLR;
323     FIFO_nRRST_1_SET;
324     FIFO_WEN_1_CLR;
325     _delay_us(10);
326     FIFO_RCLK_1_SET;
327     _delay_us(10);
328     FIFO_RCLK_1_CLR;
329     FIFO_nRRST_1_CLR;
330     _delay_us(10);
331     FIFO_RCLK_1_SET;
332     _delay_us(10);
333     FIFO_RCLK_1_CLR;
334     FIFO_nRRST_1_SET;
335     _delay_us(10);
336     FIFO_WRST_1_SET;

338 }

340 //Write one pixel in AVR
341 uint16_t FIFO_TO_AVR(uint8_t ID)
342 {
343     uint16_t data = 0;

345     DDRA = 0;
346     if(ID == 1)
347     {
348         FIFO_RCLK_1_SET;
349         data = PINA;
350         FIFO_RCLK_1_CLR;
351         data <= 8;
352         FIFO_RCLK_1_SET;
353         data |= PINA;
354         FIFO_RCLK_1_CLR;
355     }
356     else
357     {
358         FIFO_RCLK_0_SET;
359         data = PINA;
360         FIFO_RCLK_0_CLR;
361         data <= 8;
362         FIFO_RCLK_0_SET;
363         data |= PINA;
364         FIFO_RCLK_0_CLR;
365     }
366     return(data);
367 }

370 //Resets both pointers
```

```
371 void FIFO_Reset(uint8_t CameraID)
372 {
373     FIFO_nOE_0_SET;
374     FIFO_nOE_1_SET;
375     if(CameraID == 0)
376     {
377         FIFO_WRST_0_CLR;
378         FIFO_nRRST_0_CLR;
379         FIFO_RCLK_0_SET;
380         FIFO_RCLK_0_CLR;
381         FIFO_nRRST_0_SET;
382         FIFO_WRST_0_SET;
383     }
384     else
385     {
386         FIFO_WRST_1_CLR;
387         FIFO_nRRST_1_CLR;
388         FIFO_RCLK_1_SET;
389         FIFO_RCLK_1_CLR;
390         FIFO_nRRST_1_SET;
391         FIFO_WRST_1_SET;
392     }
393 }
394
395 void LoadImagesToBuffer()
396 {
397     VSYNC_0_Count = 0;
398     VSYNC_1_Count = 0;
399     FIFO_Reset(0);
400     FIFO_Reset(1);
401     VSYNC_0_Count = 1;
402     VSYNC_1_Count = 1;
403 }
404
405 uint8_t GetImageIfAvailable(FIL *File, uint8_t CameraID)
406 {
407
408     if( ((CameraID == 0) && (VSYNC_0_Count == 3)) ||
409        ((CameraID == 1) && (VSYNC_1_Count == 3)) )
410     {
411
412         //Write Bitmap Headers
413         WriteBMPHeader(File);
414         WriteDIBHeader(File);
415         if (CameraID == 0)
416         {
417             //Enable output of Camera 0
418             FIFO_nOE_0_CLR;
419             //Reset Read Pointer
420             FIFO_nRRST_0_CLR;
421             FIFO_RCLK_0_SET;
422             FIFO_RCLK_0_CLR;
423             FIFO_nRRST_0_SET;
424         }
425         else
426         {
427             //Enable output of Camera 0
428             FIFO_nOE_1_CLR;
```

```

429 //Reset Read Pointer
430 FIFO_nRRST_1_CLR;
431 FIFO_RCLK_1_SET;
432 FIFO_RCLK_1_CLR;
433 FIFO_nRRST_1_SET;
434 }
435 int i, j;
436 uint32_t pointer;
437 uint16_t Temp;
438 uint32_t p;
439 FRESULT fr;
440 //for(j = HEIGHT; j>0; j--)
441 for(j = 0; j < HEIGHT; j++)
442 {
443     pointer = 0;
444     for(i = 0; i < WIDTH; i++)
445     {
446         Temp = FIFO_TO_AVR(CameraID);
447         //USART0_Senduint16(Temp);

448         Buff[pointer++] = (uint8_t)(Temp >> 8);
449         Buff[pointer++] = (uint8_t)Temp;
450     }
451     pointer = (uint32_t)j * (uint32_t)WIDTH * 2 + BMPHEADERSIZE +
DIBHEADERSIZE;
452     f_lseek(File, pointer);
453     fr = f_write(File, Buff, WIDTH * 2, &p);
454     if(fr != FR_OK)
455     {
456         //printf("Write Fail.\n");
457         VSYNC_0_Count = 0;
458         VSYNC_1_Count = 0;
459         FIFO_Reset(CameraID);
460         FIFO_nOE_0_SET;
461         FIFO_nOE_1_SET;
462         return 1;
463     }
464 }
465 FIFO_Reset(CameraID);
466 //fr = f_close(File);
467 FIFO_nOE_0_SET;
468 FIFO_nOE_1_SET;
469 return 0;
470 }
471 else
472 {
473     return 2;
474 }
475 }
476 }
```

D.1.1.8 PCA9542A.h

..../Code/DualOV7670/PCA9542A.h

```

1  /*
2   * PCA9542A.h
3   *
4   * Created: 13/11/2012 23:24:48
5   * Author: hslovett
6   */
7
8
9 #ifndef PCA9542A_H_
10#define PCA9542A_H_
11#include "Config.h"
12
13#define A0 0
14#define A1 0
15#define A2 1
16#define PCA9542A_ADDR (0x70 | (A2 << 2) | (A1 << 1) | A0)
17
18#define NO_SELECT 0x00
19#define CH0 0x04
20#define CH1 0x05
21
22unsigned char PCA9542A_Init();
23unsigned char PCA9542A_SetChannel(uint8_t Channel);
24
25#endif /* PCA9542A_H_ */

```

D.1.1.9 PCA9542A.c

..../Code/DualOV7670/PCA9542A.c

```

1 /*
2  * PCA9542A.c
3  *
4  * Created: 13/11/2012 23:24:40
5  * Author: hslovett
6  */
7#include "PCA9542A.h"
8
9
10unsigned char PCA9542A_Init()
11{
12    unsigned char messageBuf[TWI_BUFFER_SIZE];
13    messageBuf[0] = (PCA9542A_ADDR << TWI_ADR_BITS) | (FALSE << TWI_READ_BIT); // The first byte must always consist of General Call code or the TWI slave address.
14    messageBuf[1] = NO_SELECT; // The first byte is used for commands.
15    // The second byte is used for the data.
16    TWI_Start_Transceiver_With_Data(messageBuf, 2);
17
18    while(TWI_Transceiver_Busy()); //Wait for transceiver to clear
19
20    return TWI_statusReg.lastTransOK;
21}

```

```

23     unsigned char PCA9542A_SetChannel( uint8_t Channel )
24 {
25     unsigned char messageBuf[TWI_BUFFER_SIZE];
26     messageBuf[0] = (PCA9542A_ADDR <<TWI_ADR_BITS) | (FALSE<<TWI_READ_BIT); // The first byte must always consist of General Call code or the TWI slave address.
27     messageBuf[1] = Channel; // The first byte is used for commands.
28                         // The second byte is used for the data.
29     TWI_Start_Transceiver_With_Data( messageBuf, 2 );
30
31     while(TWI_Transceiver_Busy()) ; //Wait for transceiver to clear
32
33     return TWI_statusReg.lastTransOK;
34 }
35
36     unsigned char PCA9542A_ReadChannel()
37 {
38     unsigned char messageBuf[TWI_BUFFER_SIZE];
39     messageBuf[0] = (PCA9542A_ADDR <<TWI_ADR_BITS) | (TRUE<<TWI_READ_BIT); // The first byte must always consist of General Call code or the TWI slave address.
40
41     TWI_Start_Transceiver_With_Data( messageBuf, 1 );
42
43     while(TWI_Transceiver_Busy()) ; //Wait for transceiver to clear
44 // Get the received data from the transceiver buffer
45     TWI_Get_Data_From_Transceiver( messageBuf, 2 );
46     return TWI_statusReg.lastTransOK;
47 }
```

D.1.1.10 TWI_Master.h

..../Code/DualOV7670/TWI_Master.h

```
1  ****
2  *
3  * Atmel Corporation
4  *
5  * File : TWI_Master.h
6  * Compiler : IAR EWAVR 2.28a/3.10c
7  * Revision : Revision: 1.13
8  * Date : Date: 24. mai 2004 11:31:22
9  * Updated by : Author: ltwa
10 *
11 * Support mail : avr@atmel.com
12 *
13 * Supported devices : All devices with a TWI module can be used.
14 *                      The example is written for the ATmega16
15 *
16 * AppNote : AVR315 - TWI Master Implementation
17 *
18 * Description : Header file for TWI_Master.c
19 *                 Include this file in the application.
```

```

20  /*
21   ****
22   * Modified by Henry Lovett (hl13g10@ecs.soton.ac.uk) to allow SCL frequency
23   * to be specified and TWBR calculated
24   * Also allows AVR internal pull up resistors to be used.
25   */
26 #ifndef _TWI_MASTER_H
27 #define _TWI_MASTER_H
28 #include <avr/io.h>
29 #include <avr/interrupt.h>
30 #include "Config.h"
31 /**
32  * TWI Status/Control register definitions
33 */
34
35 #define INTERNAL_PULLUPS 0
36
37 #define TWI_BUFFER_SIZE 4    // Set this to the largest message size that will
38 // be sent including address byte.
39
40 #define SCL_Freq      100000    //SCL Frequency in Hertz
41 #define TWI_TWBR      (char)(F_CPU / 2 / SCL_Freq - 8) //Equation to calculate
42 //TWBR Based on SCL Frequency and Clock Frequency
43
44 //##define TWI_TWBR          0x0C //400KHz           // TWI Bit rate Register
45 //      setting.
46 //##define TWI_TWBR          0x34 //100KHz           //
47 //      Se Application note for detailed
48 //      information on setting this value.
49 // Not used defines!
50 //##define TWI_TWPS          0x00           // This driver presumes prescaler = 00
51
52 /**
53  * Global definitions
54 */
55
56 union TWI_statusReg           // Status byte holding flags.
57 {
58     unsigned char all;
59     struct
60     {
61         unsigned char lastTransOK:1;
62         unsigned char unusedBits:7;
63     };
64
65     extern union TWI_statusReg TWI_statusReg;
66
67 /**
68  * Function definitions
69 */
70
71 void TWI_Master_Initialise( void );
72 unsigned char TWI_Transceiver_Busy( void );
73 unsigned char TWI_Get_State_Info( void );
74 void TWI_Start_Transceiver_With_Data( unsigned char * , unsigned char );
75 void TWI_Start_Transceiver( void );
76 unsigned char TWI_Get_Data_From_Transceiver( unsigned char * , unsigned char );

```

```

74  **** Bit and byte definitions ****
75  Bit and byte definitions
76  ****
77  #define TWI_READ_BIT 0           // Bit position for R/W bit in "address byte".
78  #define TWI_ADR_BITS 1          // Bit position for LSB of the slave address
    bits in the init byte.

80  #define TRUE      1
81  #define FALSE     0

83  **** TWI State codes ****
84  TWI State codes
85  ****
86  // General TWI Master status codes
87  #define TWI_START          0x08 // START has been transmitted
88  #define TWI_REP_START       0x10 // Repeated START has been
    transmitted
89  #define TWI_ARB_LOST        0x38 // Arbitration lost

91  // TWI Master Transmitter status codes
92  #define TWI_MTX_ADR_ACK     0x18 // SLA+W has been transmitted and ACK
    received
93  #define TWI_MTX_ADR_NACK    0x20 // SLA+W has been transmitted and
    NACK received
94  #define TWI_MTX_DATA_ACK    0x28 // Data byte has been transmitted and
    ACK received
95  #define TWI_MTX_DATA_NACK   0x30 // Data byte has been transmitted and
    NACK received

97  // TWI Master Receiver status codes
98  #define TWI_MRX_ADR_ACK     0x40 // SLA+R has been transmitted and ACK
    received
99  #define TWI_MRX_ADR_NACK    0x48 // SLA+R has been transmitted and
    NACK received
100 #define TWI_MRX_DATA_ACK    0x50 // Data byte has been received and
    ACK transmitted
101 #define TWI_MRX_DATA_NACK   0x58 // Data byte has been received and
    NACK transmitted

103 // TWI Slave Transmitter status codes
104 #define TWI_STX_ADR_ACK     0xA8 // Own SLA+R has been received; ACK
    has been returned
105 #define TWI_STX_ADR_ACK_M_ARB_LOST 0xB0 // Arbitration lost in SLA+R/W as
    Master; own SLA+R has been received; ACK has been returned
106 #define TWI_STX_DATA_ACK    0xB8 // Data byte in TWDR has been
    transmitted; ACK has been received
107 #define TWI_STX_DATA_NACK   0xC0 // Data byte in TWDR has been
    transmitted; NOT ACK has been received
108 #define TWI_STX_DATA_ACK_LAST_BYTE 0xC8 // Last data byte in TWDR has been
    transmitted (TWEA = 0); ACK has been received

110 // TWI Slave Receiver status codes
111 #define TWI_SRX_ADR_ACK     0x60 // Own SLA+W has been received ACK
    has been returned
112 #define TWI_SRX_ADR_ACK_M_ARB_LOST 0x68 // Arbitration lost in SLA+R/W as
    Master; own SLA+W has been received; ACK has been returned

```

```
113 #define TWI_SRX_GEN_ACK           0x70 // General call address has been
114   received; ACK has been returned
115 #define TWI_SRX_GEN_ACK_M_ARB_LOST 0x78 // Arbitration lost in SLA+R/W as
116   Master; General call address has been received; ACK has been returned
117 #define TWI_SRX_ADR_DATA_ACK       0x80 // Previously addressed with own SLA+
118   W; data has been received; ACK has been returned
119 #define TWI_SRX_ADR_DATA_NACK     0x88 // Previously addressed with own SLA+
120   W; data has been received; NOT ACK has been returned
121 #define TWI_SRX_GEN_DATA_ACK       0x90 // Previously addressed with general
122   call; data has been received; ACK has been returned
123 #define TWI_SRX_GEN_DATA_NACK     0x98 // Previously addressed with general
124   call; data has been received; NOT ACK has been returned
125 #define TWI_SRX_STOP_RESTART      0xA0 // A STOP condition or repeated START
126   condition has been received while still addressed as Slave

127 // TWI Miscellaneous status codes
128 #define TWI_NO_STATE              0xF8 // No relevant state information
129   available; TWINT = 0
130 #define TWI_BUS_ERROR             0x00 // Bus error due to an illegal START
131   or STOP condition

132 #endif
```

D.1.1.11 TWI_Master.c

..../Code/DualOV7670/TWI_Master.c

```
1  ****
2  *
3  * Atmel Corporation
4  *
5  * File : TWI_Master.c
6  * Compiler : IAR EWAVR 2.28a/3.10c
7  * Revision : Revision: 1.13
8  * Date : Date: 24. mai 2004 11:31:20
9  * Updated by : Author: ltwa
10 *
11 * Support mail : avr@atmel.com
12 *
13 * Supported devices : All devices with a TWI module can be used.
14 *                      The example is written for the ATmega16
15 *
16 * AppNote : AVR315 - TWI Master Implementation
17 *
18 * Description : This is a sample driver for the TWI hardware modules.
19 *                 It is interrupt driven. All functionality is
20 *                 controlled through
21 *                 passing information to and from functions. See main.c for
22 *                 samples
23 *                 of how to use the driver.
24 * ****
```

```

27 #include "TWI_Master.h"

28 static unsigned char TWI_buf[ TWI_BUFFER_SIZE ];           // Transceiver buffer
29 static unsigned char TWI_msgSize;                         // Number of bytes to be
   transmitted.
30 static unsigned char TWI_state = TWI_NO_STATE;           // State byte. Default set
   to TWI_NO_STATE.

33 union TWI_statusReg TWI_statusReg = {0};                // TWI_statusReg is
   defined in TWI_Master.h

35 ****
36 Call this function to set up the TWI master to its initial standby state.
37 Remember to enable interrupts from the main application after initializing the
   TWI.
38 ****
39 void TWI_Master_Initialise(void)
40 {
41     #if INTERNAL_PULLUPS == 1 //enable built in pullups for I2C Lines
42         DDRC = 0x00;
43         PORTC = (1 << PC0) | (1 << PC1);
44     #else
45         #pragma message("External I2C Pull Ups Required.")
46     #endif
47     TWBR = TWI_TWBR;                                       // Set bit rate register (
   Baudrate). Defined in header file.
48     // TWSR = TWI_TWPS;                                     // Not used. Driver
   presumes prescaler to be 00.
49     TWDR = 0xFF;                                         // Default content = SDA
   released.
50     TWCR = (1<<TWEN) |                                // Enable TWI-interface
   and release TWI pins.
51     (0<<TWIE) | (0<<TWINT);                           // Disable Interrupt.
52     (0<<TWEA) | (0<<TWSTA) | (0<<TWSTO);             // No Signal requests.
53     (0<<TWWC);                                         //
54 }

56 ****
57 Call this function to test if the TWI_ISR is busy transmitting.
58 ****
59 unsigned char TWI_Transceiver_Busy( void )
60 {
61     return ( TWCR & (1<<TWIE) );                      // IF TWI Interrupt is enabled
   then the Transceiver is busy
62 }

64 ****
65 Call this function to fetch the state information of the previous operation.
   The function will hold execution (loop)
66 until the TWI_ISR has completed with the previous operation. If there was an
   error, then the function
67 will return the TWI State code.
68 ****
69 unsigned char TWI_Get_State_Info( void )
70 {
71     while ( TWI_Transceiver_Busy() );                  // Wait until TWI has
   completed the transmission.

```

```

72     return ( TWI_state );                                // Return error state.
73 }

75 ****
76 Call this function to send a prepared message. The first byte must contain the
    slave address and the
77 read/write bit. Consecutive bytes contain the data to be sent, or empty
    locations for data to be read
78 from the slave. Also include how many bytes that should be sent/read including
    the address byte.
79 The function will hold execution (loop) until the TWI_ISR has completed with
    the previous operation,
80 then initialize the next operation and return.
81 ****
82 void TWI_Start_Transceiver_With_Data( unsigned char *msg, unsigned char
    msgSize )
83 {
84     unsigned char temp;

86     while ( TWI_Transceiver_Busy() );                      // Wait until TWI is ready for
        next transmission.

88     TWI_msgSize = msgSize;                                  // Number of data to transmit.
89     TWI_buf[0] = msg[0];                                    // Store slave address with R/
        W setting.
90     if ( !( msg[0] & (TRUE<<TWI_READ_BIT) ) )           // If it is a write operation,
        then also copy data.
91     {
92         for ( temp = 1; temp < msgSize; temp++ )
93             TWI_buf[ temp ] = msg[ temp ];
94     }
95     TWI_statusReg.all = 0;
96     TWI_state       = TWI_NO_STATE ;
97     TWCR = (1<<TWEN)|                                // TWI Interface enabled.
98          (1<<TWIE)|(1<<TWINT)|                     // Enable TWI Interrupt and
        clear the flag.
99     (0<<TWEA)|(1<<TWSTA)|(0<<TWSTO)|            // Initiate a START condition.
100    (0<<TWWC);                                     //
101 }

103 ****
104 Call this function to resend the last message. The driver will reuse the data
    previously put in the transceiver buffers.
105 The function will hold execution (loop) until the TWI_ISR has completed with
    the previous operation,
106 then initialize the next operation and return.
107 ****
108 void TWI_Start_Transceiver( void )
109 {
110     while ( TWI_Transceiver_Busy() );                  // Wait until TWI is ready for
        next transmission.
111     TWI_statusReg.all = 0;
112     TWI_state       = TWI_NO_STATE ;
113     TWCR = (1<<TWEN)|                                // TWI Interface enabled.
114          (1<<TWIE)|(1<<TWINT)|                     // Enable TWI Interrupt and
        clear the flag.
115     (0<<TWEA)|(1<<TWSTA)|(0<<TWSTO)|            // Initiate a START condition.
116     (0<<TWWC);                                     //

```

```

117 }

119 ****
120 Call this function to read out the requested data from the TWI transceiver
121 buffer. I.e. first call
122 TWI_Start_Transceiver to send a request for data to the slave. Then Run this
123 function to collect the
124 data when they have arrived. Include a pointer to where to place the data and
125 the number of bytes
126 requested (including the address field) in the function call. The function
127 will hold execution (loop)
128 until the TWI_ISR has completed with the previous operation, before reading
129 out the data and returning.
130 If there was an error in the previous transmission the function will return
131 the TWI error code.
132 ****
133 unsigned char TWI_Get_Data_From_Transceiver( unsigned char *msg, unsigned char
134 msgSize )
135 {
136     unsigned char i;
137
138     while ( TWI_Transceiver_Busy() );                                // Wait until TWI is ready for
139     next transmission.
140
141     if( TWI_statusReg.lastTransOK )                                     // Last transmission competed
142     successfully.
143     {
144         for ( i=0; i<msgSize; i++ )                                     // Copy data from Transceiver
145         buffer.
146         {
147             msg[ i ] = TWI_buf[ i ];
148         }
149     }
150     return( TWI_statusReg.lastTransOK );
151 }

152 // ***** Interrupt Handlers **** //
153 ****
154 This function is the Interrupt Service Routine (ISR), and called when the TWI
155 interrupt is triggered;
156 that is whenever a TWI event has occurred. This function should not be called
157 directly from the main
158 application.
159 ****

160 ISR(TWI_vect)
161 {
162     static unsigned char TWI_bufPtr;

163     switch (TWSR)
164     {
165         case TWI_START:           // START has been transmitted
166         case TWI_REP_START:      // Repeated START has been transmitted
167             TWI_bufPtr = 0;          // Set buffer
168             pointer to the TWI Address location
169         case TWI_MTX_ADR_ACK:    // SLA+W has been transmitted and ACK received
170         case TWI_MTX_DATA_ACK:   // Data byte has been transmitted and ACK
171             received

```

```

161     if (TWI_bufPtr < TWI_msgSize)
162     {
163         TWDR = TWI_buf[TWI_bufPtr++];
164         TWCR = (1<<TWEN) | // TWI Interface
165         enabled
166             (1<<TWIE)|(1<<TWINT) | // Enable TWI
167             Interrupt and clear the flag to send byte
168             (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO) |
169             (0<<TWWC); // //
170     }else // Send STOP after last byte
171     {
172         TWI_statusReg.lastTransOK = TRUE; // Set status bits
173         to completed successfully.
174         TWCR = (1<<TWEN) | // TWI Interface
175         enabled
176             (0<<TWIE)|(1<<TWINT) | // Disable TWI
177             Interrupt and clear the flag
178             (0<<TWEA)|(0<<TWSTA)|(1<<TWSTO) | // Initiate a STOP
179             condition. // (0<<TWWC);
180     }
181     break;
182 case TWI_MRX_DATA_ACK: // Data byte has been received and ACK
183     transmitted
184     TWI_buf[TWI_bufPtr++] = TWDR;
185 case TWI_MRX_ADR_ACK: // SLA+R has been transmitted and ACK received
186     if (TWI_bufPtr < (TWI_msgSize-1) ) // Detect the last
187     byte to NACK it.
188     {
189         TWCR = (1<<TWEN) | // TWI Interface
190         enabled
191             (1<<TWIE)|(1<<TWINT) | // Enable TWI
192             Interrupt and clear the flag to read next byte
193             (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO) | // Send ACK after
194             reception // (0<<TWWC);
195     }else // Send NACK after next reception
196     {
197         TWCR = (1<<TWEN) | // TWI Interface
198         enabled
199             (1<<TWIE)|(1<<TWINT) | // Enable TWI
199             Interrupt and clear the flag
199             (0<<TWEA)|(0<<TWSTA)|(1<<TWSTO) | // Initiate a STOP
199             condition.

```

```

200         (0<<TWWC);                                //
201     break;
202 case TWI_ARB_LOST:           // Arbitration lost          //
203     TWCR = (1<<TWEN)|                           // TWI Interface
204     enabled
205         (1<<TWIE)|(1<<TWINT)|                  // Enable TWI Interrupt
206         and clear the flag
207         (0<<TWEA)|(1<<TWSTA)|(0<<TWSTO)|        // Initiate a (RE)
208 START condition.
209         (0<<TWWC);                                //
210     break;
211 case TWI_MTX_ADDR_NACK:    // SLA+W has been transmitted and NACK
212 received
213 case TWI_MRX_ADDR_NACK:    // SLA+R has been transmitted and NACK
214 received
215 case TWI_MTX_DATA_NACK:   // Data byte has been transmitted and NACK
216 received
217 //    case TWI_NO_STATE                // No relevant state information
218 available; TWINT = 0
219 case TWI_BUS_ERROR:        // Bus error due to an illegal START or STOP
220 condition
221 default:
222     TWI_state = TWSR;                          // Store TWSR and
223 automatically sets clears noErrors bit.
224
225     TWCR = (1<<TWEN)|                      // Reset TWI Interface
226     interface and release TWI pins           // Enable TWI-
227         (0<<TWIE)|(0<<TWINT)|              // Disable Interrupt
228         (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|  // No Signal requests
229         (0<<TWWC);                            //
230 }
231 }

```

D.1.1.12 Usart.h

.../Code/DualOV7670/Usart.h

```

1  /*
2  *  Usart.h
3  *
4  *  Created: 25/10/2012 22:25:14
5  *  Author: hslovett
6  */
7
8
9 #ifndef USART_H_
10#define USART_H_
11
12#include "Config.h"
13#include <stdio.h>
14#include <avr/io.h>
15#define USART0_BITRATE 57600
16#define UBR_F_CPU/16/USART0_BITRATE-1

```

```

18 void USART0_Init ();
19 void Usart_SendChar(char data);
20 unsigned char Usart_Receive( void );
21 int Usart_printf(char var, FILE *stream);
22 void Usart_get_line (char *buff, int len);
23 void USART0_Senduint16 (uint16_t Data);
24 // void USART0_SendChar( unsigned char data );
25 // unsigned char USART0_Receive( void );
26 // void USART0_SendString(char str[]);

28 #endif /* USART_H */

```

D.1.1.13 Usart.c

..../Code/DualOV7670/Usart.c

```

1 /*
2  * Usart.c
3  *
4  * Created: 25/10/2012 22:25:04
5  * Author: hl13g10@ecs.soton.ac.uk
6  */

9 #include "Usart.h"

12 void USART0_Init()
13 {
14     uint16_t ubrr = UBBR;
15     //Set baud rate
16     UBRROH = (unsigned char)(ubrr >>8);
17     UBRROL = (unsigned char)ubrr ;
18     //Enable receiver and transmitter
19     UCSROB = (1<<RXENO)|(1<<TXENO);

21     UCSROC = 0x06; //set asynchronous, no parity, one stop bit, 8 bit transfer.

23     //UCSROB |= (1 << RXCIE0) | (1 << TXCIE0); //set RX and TX interrupt on
24 }
25 void Usart_SendChar(char data)
26 {
27     // Wait for empty transmit buffer
28     while ( !(UCSROA & (1 << UDRE0)) );
29     // Start transmission
30     UDRO = data;
31 }
32 unsigned char Usart_Receive( void )
33 {
34     /* Wait for data to be received */
35     while ( !(UCSROA & (1<<RXCO)) )
36     ;
37     /* Get and return received data from buffer */
38     //Usart_SendChar(UDRO);

```

```

39     return UDRO;
40 }

42 //to use this copy the following as a global-
43 // static FILE mystdout = FDEV_SETUP_STREAM(Usart_printf, NULL,
44 // _FDEV_SETUP_WRITE);
45 // and add this line at the beginning of main:
46 // stdout = &mystdout;
47 // stdio.h must be used.
48 int Usart_printf(char var, FILE *stream) {
49     // translate \n to \r for br@y++ terminal
50     if (var == '\n') Usart_SendChar('\r');
51     Usart_SendChar(var);
52     return 0;
53 }

54 void Usart_get_line (char *buff, int len)
55 {
56     cli();
57     char c;
58     int i = 0;

59     for (;;) {
60         c = Usart_Receive();
61         if (c == '\r') break;
62         if ((c == '\b') && i) {
63             i--;
64             Usart_SendChar(c);
65             continue;
66         }
67         if (c >= ' ' && i < len - 1) { /* Visible chars */
68             buff[i++] = c;
69             Usart_SendChar(c);
70         }
71     }
72     buff[i] = 0;
73     Usart_SendChar('\n');
74     sei();
75 }
76 void USART0_Senduint16 (uint16_t Data)
77 {
78     Usart_SendChar(Data >> 8);
79     Usart_SendChar(Data & 0xFF);
80 }
81

```

D.1.2 Dual Camera User Interface

D.1.2.1 DualCamera_UI.c

..../Code/DualCamera_UI/DualCamera_UI.c

1 /*

```
2  * DualCamera_UI.c
3  *
4  * Created: 12/11/2012 08:32:27
5  * Author: hslovett
6  */
7
8
9 #include <avr/io.h>
10 #include <avr/interrupt.h>
11 #include "TWI_slave.h"
12
13 #define ButtonMask 0x0F
14
15 #define TWI_CMD_MASTER_WRITE 0x10
16 #define TWI_CMD_MASTER_READ 0x20
17
18 // When there has been an error, this function is run and takes care of it
19 unsigned char TWI_Act_On_Failure_In_Last_Transmission ( unsigned char
20             TWIErrorMsg );
21
22 int main(void)
23 {
24
25     char ButtonStatus = 0xFF;
26     unsigned char TWI_slaveAddress;
27     unsigned char messageBuff[TWI_BUFFER_SIZE];
28     DDRD = 0xFF; // Port D is the LED output
29     DDRC = 0x00; //PortC is the switch input
30     //PORTC = 0xFF;
31     TWI_slaveAddress = 0x15;
32     TWI_Slave_Initialise( (unsigned char)((TWI_slaveAddress<<TWI_ADR_BITS) | (
33         TRUE<<TWI_GEN_BIT) ) );
34     sei();
35     TWI_Start_Transceiver();
36     while(1)
37     {
38
39         ButtonStatus = (ButtonStatus & PINC) & ButtonMask;
40         //PORTD = ButtonStatus;
41         // Check if the TWI Transceiver has completed an operation.
42         if ( ! TWI_Transceiver_Busy() )
43         {
44             // Check if the last operation was successful
45             if ( TWI_statusReg.lastTransOK )
46             {
47                 // Check if the last operation was a reception
48                 if ( TWI_statusReg.RxDataInBuf )
49                 {
50                     TWI_Get_Data_From_Transceiver(messageBuff, 2);
51                     // Check if the last operation was a reception as General Call
52                     if ( TWI_statusReg.genAddressCall )
53                     {
54                         // Put data received out to PORTB as an example.
55                         PORTB = messageBuff[0];
56                     }
57                     else // Ends up here if the last operation was a reception as
58                     Slave Address Match
59                 }
60             }
61         }
62     }
63 }
```

```

57         // Example of how to interpret a command and respond.
58
59         // TWI_CMD_MASTER_WRITE stores the data to PORTB
60         if (messageBuff[0] == TWI_CMD_MASTER_WRITE)
61         {
62             PORTD = messageBuff[1];
63         }
64         // TWI_CMD_MASTER_READ prepares the data from PINB in the
65         transceiver buffer for the TWI master to fetch.
66         if (messageBuff[0] == TWI_CMD_MASTER_READ)
67         {
68             messageBuff[0] = ButtonStatus;
69             TWI_Start_Transceiver_With_Data( messageBuff, 1 );
70             ButtonStatus = ButtonMask; //clear all logged button presses
71         }
72     }
73     else // Ends up here if the last operation was a transmission
74     {
75         //__no_operation(); // Put own code here.
76     }
77     // Check if the TWI Transceiver has already been started.
78     // If not then restart it to prepare it for new receptions.
79     if ( ! TWI_Transceiver_Busy() )
80     {
81         TWI_Start_Transceiver();
82     }
83 }
84 else // Ends up here if the last operation completed unsuccessfully
85 {
86     //TWI_Act_On_Failure_In_Last_Transmission( TWI_Get_State_Info() );
87 }
88 }
89 }
90 }
91
92 unsigned char TWI_Act_On_Failure_In_Last_Transmission ( unsigned char
93     TWIerrorMsg )
94 {
95     // A failure has occurred, use TWIerrorMsg to determine the nature of the
96     // failure
97     // and take appropriate actions.
98     // See header file for a list of possible failures messages.
99
100    // This very simple example puts the error code on PORTB and restarts the
101    // transceiver with
102    // all the same data in the transmission buffers.
103    //PORTB = TWIerrorMsg;
104    TWI_Start_Transceiver();

105    return TWIerrorMsg;
106 }
```

D.1.2.2 TWI_slave.h

..//Code/DualCamera_UI/TWI_slave.h

```
1  ****
2  *
3  * Atmel Corporation
4  *
5  * File : TWI_Slave.h
6  * Compiler : IAR EWAAVR 2.28a/3.10c
7  * Revision : Revision: 2475
8  * Date : Date: 2007-09-20 12:00:43 +0200 (to, 20 sep 2007)
9  * Updated by : Author: mlarsson
10 *
11 * Support mail : avr@atmel.com
12 *
13 * Supported devices : All devices with a TWI module can be used.
14 *                      The example is written for the ATmega16
15 *
16 * AppNote : AVR311 - TWI Slave Implementation
17 *
18 * Description : Header file for TWI_slave.c
19 *                 Include this file in the application.
20 *
21 ****
22 /*! \page MISRA
23 *
24 * General disabling of MISRA rules:
25 * * (MISRA C rule 1) compiler is configured to allow extensions
26 * * (MISRA C rule 111) bit fields shall only be defined to be of type
27 *   unsigned int or signed int
28 * * (MISRA C rule 37) bitwise operations shall not be performed on signed
29 *   integer types
30 * As it does not work well with 8bit architecture and/or IAR
31 *
32 * Other disabled MISRA rules
33 * * (MISRA C rule 109) use of union - overlapping storage shall not be used
34 * * (MISRA C rule 61) every non-empty case clause in a switch statement shall
35 *   be terminated with a break statement
36 */
37 ****
38 #define TWI_BUFFER_SIZE 4      // Reserves memory for the drivers transceiver
39           // buffer.
40           // Set this to the largest message size that
41           // will be sent including address byte.
42 ****
43 Global definitions
44 ****
45
46 union TWI_statusReg_t          // Status byte holding flags.
47 {
48     unsigned char all;
49     struct
50 {
```

```

51     unsigned char lastTransOK:1;
52     unsigned char RxDataInBuf:1;
53     unsigned char genAddressCall:1;                                // TRUE =
54     General call, FALSE = TWI Address;
55     unsigned char unusedBits:5;
56 };
57 };

58 extern union TWI_statusReg_t TWI_statusReg;

59 //static unsigned char dont_sleep = 0;

60 /****** Function definitions ******/
61
62 void TWI_Slave_Initialise( unsigned char );
63 unsigned char TWI_Transceiver_Busy( void );
64 unsigned char TWI_Get_State_Info( void );
65 void TWI_Start_Transceiver_With_Data( unsigned char * , unsigned char );
66 void TWI_Start_Transceiver( void );
67 unsigned char TWI_Get_Data_From_Transceiver( unsigned char *, unsigned char );

68 ISR( TWI_vect );

69 /****** Bit and byte definitions ******/
70
71 #define TWI_READ_BIT    0    // Bit position for R/W bit in "address byte".
72 #define TWI_ADR_BITS   1    // Bit position for LSB of the slave address bits in
73     the init byte.
74 #define TWI_GEN_BIT    0    // Bit position for LSB of the general call bit in
75     the init byte.

76 #define TRUE          1
77 #define FALSE         0

78 /****** TWI State codes ******/
79
80 // General TWI Master staus codes
81 #define TWI_START           0x08 // START has been transmitted
82 #define TWI_REP_START        0x10 // Repeated START has been
83     transmitted
84 #define TWI_ARB_LOST         0x38 // Arbitration lost

85 // TWI Master Transmitter staus codes
86 #define TWI_MTX_ADR_ACK      0x18 // SLA+W has been tramsmitted and ACK
87     received
88 #define TWI_MTX_ADR_NACK     0x20 // SLA+W has been tramsmitted and
89     NACK received
90 #define TWI_MTX_DATA_ACK      0x28 // Data byte has been tramsmitted and
91     ACK received
92 #define TWI_MTX_DATA_NACK     0x30 // Data byte has been tramsmitted and
93     NACK received

94 // TWI Master Receiver staus codes
95 #define TWI_MRX_ADR_ACK      0x40 // SLA+R has been tramsmitted and ACK
96     received

```

```

100 #define TWI_MRX_ADR_NACK          0x48 // SLA+R has been transmitted and
101   NACK received
102 #define TWI_MRX_DATA_ACK         0x50 // Data byte has been received and
103   ACK transmitted
104 #define TWI_MRX_DATA_NACK        0x58 // Data byte has been received and
105   NACK transmitted
106
107 // TWI Slave Transmitter status codes
108 #define TWI_STX_ADR_ACK           0xA8 // Own SLA+R has been received; ACK
109   has been returned
110 #define TWI_STX_ADR_ACK_M_ARB_LOST 0xB0 // Arbitration lost in SLA+R/W as
111   Master; own SLA+R has been received; ACK has been returned
112 #define TWI_STX_DATA_ACK          0xB8 // Data byte in TWDR has been
113   transmitted; ACK has been received
114 #define TWI_STX_DATA_NACK         0xC0 // Data byte in TWDR has been
115   transmitted; NOT ACK has been received
116 #define TWI_STX_DATA_ACK_LAST_BYTE 0xC8 // Last data byte in TWDR has been
117   transmitted (TWEA = 0); ACK has been received
118
119 // TWI Slave Receiver status codes
120 #define TWI_SRX_ADR_ACK           0x60 // Own SLA+W has been received ACK
121   has been returned
122 #define TWI_SRX_ADR_ACK_M_ARB_LOST 0x68 // Arbitration lost in SLA+R/W as
123   Master; own SLA+W has been received; ACK has been returned
124 #define TWI_SRX_GEN_ACK            0x70 // General call address has been
125   received; ACK has been returned
126 #define TWI_SRX_GEN_ACK_M_ARB_LOST 0x78 // Arbitration lost in SLA+R/W as
127   Master; General call address has been received; ACK has been returned
128 #define TWI_SRX_ADR_DATA_ACK       0x80 // Previously addressed with own SLA+
129   W; data has been received; ACK has been returned
130 #define TWI_SRX_ADR_DATA_NACK      0x88 // Previously addressed with own SLA+
131   W; data has been received; NOT ACK has been returned
132 #define TWI_SRX_GEN_DATA_ACK       0x90 // Previously addressed with general
133   call; data has been received; ACK has been returned
134 #define TWI_SRX_GEN_DATA_NACK      0x98 // Previously addressed with general
135   call; data has been received; NOT ACK has been returned
136 #define TWI_SRX_STOP_RESTART        0xA0 // A STOP condition or repeated START
137   condition has been received while still addressed as Slave
138
139 // TWI Miscellaneous status codes
140 #define TWI_NO_STATE              0xF8 // No relevant state information
141   available; TWINT = 0
142 #define TWI_BUS_ERROR              0x00 // Bus error due to an illegal START
143   or STOP condition

```

D.1.2.3 TWI_slave.c

..../Code/DualCamera_UI/TWI_slave.c

```

1 ****
2 *
3 * Atmel Corporation
4 *
5 * File : TWI_Slave.c

```

```
6 * Compiler : IAR EWAAVR 2.28a/3.10c
7 * Revision : Revision: 2475
8 * Date : Date: 2007-09-20 12:00:43 +0200 (to, 20 sep 2007)
9 * Updated by : Author: mlarsson
10 *
11 * Support mail : avr@atmel.com
12 *
13 * Supported devices : All devices with a TWI module can be used.
14 *                      The example is written for the ATmega16
15 *
16 * AppNote : AVR311 - TWI Slave Implementation
17 *
18 * Description : This is sample driver to AVR's TWI module.
19 *                 It is interrupt driven. All functionality is controlled
20 *                 through
21 *                 samples
22 *                 passing information to and from functions. See main.c for
23 *                 of how to use the driver.
24 */
25 *
26 * General disabling of MISRA rules:
27 * * (MISRA C rule 1) compiler is configured to allow extensions
28 * * (MISRA C rule 111) bit fields shall only be defined to be of type
29 *   unsigned int or signed int
30 * * (MISRA C rule 37) bitwise operations shall not be performed on signed
31 *   integer types
32 * As it does not work well with 8bit architecture and/or IAR
33 *
34 * Other disabled MISRA rules
35 * * (MISRA C rule 109) use of union - overlapping storage shall not be used
36 * * (MISRA C rule 61) every non-empty case clause in a switch statement shall
37 *   be terminated with a break statement
38 */
39
40 #include <avr/io.h>
41 #include <avr/interrupt.h>
42 #include "TWI_slave.h"
43
44 static unsigned char TWI_buf[TWI_BUFFER_SIZE];      // Transceiver buffer. Set
45     the size in the header file
46 static unsigned char TWI_msgSize = 0;                // Number of bytes to be
47     transmitted.
48 static unsigned char TWI_state = TWI_NO_STATE; // State byte. Default set
49     to TWI_NO_STATE.
50
51 // This is true when the TWI is in the middle of a transfer
52 // and set to false when all bytes have been transmitted/received
53 // Also used to determine how deep we can sleep.
54 static unsigned char TWI_busy = 0;
55
56 union TWI_statusReg_t TWI_statusReg = {0};          // TWI_statusReg is
57     defined in TWI_Slave.h
58
59 ****
60 Call this function to set up the TWI slave to its initial standby state.
```

```

54 Remember to enable interrupts from the main application after initializing the
55 TWI.
56 Pass both the slave address and the requirements for triggering on a general
57 call in the
58 same byte. Use e.g. this notation when calling this function:
59 TWI_Slave_Initialise( (TWI_slaveAddress<<TWI_ADR_BITS) | (TRUE<<TWI_GEN_BIT) )
60 ;
61 The TWI module is configured to NACK on any requests. Use a
62 TWI_Start_Transceiver function to
63 start the TWI.
64 ****
65 void TWI_Slave_Initialise( unsigned char TWI_ownAddress )
66 {
67     TWAR = TWI_ownAddress;                                // Set own TWI slave
68     address. Accept TWI General Calls.
69     TWCR = (1<<TWEN)|                                     // Enable TWI-interface
70     and release TWI pins.
71     (0<<TWIE)|(0<<TWINT)|                                // Disable TWI Interrupt.
72     (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|                  // Do not ACK on any
73     requests, yet.
74     (0<<TWWC);                                         //
75     TWI_busy = 0;
76 }
77 ****
78 Call this function to test if the TWI_ISR is busy transmitting.
79 ****
80 unsigned char TWI_Transceiver_Busy( void )
81 {
82     return TWI_busy;
83 }
84 ****
85 Call this function to fetch the state information of the previous operation.
86 The function will hold execution (loop)
87 until the TWI_ISR has completed with the previous operation. If there was an
88 error, then the function
89 will return the TWI State code.
90 ****
91 unsigned char TWI_Get_State_Info( void )
92 {
93     while ( TWI_Transceiver_Busy() ) {}                      // Wait until TWI has
94     completed the transmission.
95     return ( TWI_state );                                    // Return error state.
96 }
97 ****
98 Call this function to send a prepared message, or start the Transceiver for
99 reception. Include
100 a pointer to the data to be sent if a SLA+W is received. The data will be
101 copied to the TWI buffer.
102 Also include how many bytes that should be sent. Note that unlike the similar
103 Master function, the
104 Address byte is not included in the message buffers.
105 The function will hold execution (loop) until the TWI_ISR has completed with
106 the previous operation,
107 then initialize the next operation and return.
108 ****

```

```

98 void TWI_Start_Transceiver_With_Data( unsigned char *msg, unsigned char
99   msgSize )
100 {
101   unsigned char temp;
102
103   while ( TWI_Transceiver_Busy() ) {} // Wait until TWI is ready
104   for next transmission.
105
106   TWI_msgSize = msgSize; // Number of data to transmit.
107   for ( temp = 0; temp < msgSize; temp++ ) // Copy data that may be
108     transmitted if the TWI Master requests data.
109   {
110     TWI_buf[ temp ] = msg[ temp ];
111   }
112   TWI_statusReg.all = 0;
113   TWI_state = TWI_NO_STATE;
114   TWCR = (1<<TWEN) | // TWI Interface enabled.
115     (1<<TWIE)|(1<<TWINT)| // Enable TWI Interrupt and
116     clear the flag.
117     (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)| // Prepare to ACK next time
118     the Slave is addressed.
119     (0<<TWWC); // /
120   TWI_busy = 1;
121 }
122
123 ****
124 Call this function to start the Transceiver without specifying new transmission
125   data. Useful for restarting
126 a transmission, or just starting the transceiver for reception. The driver
127   will reuse the data previously put
128 in the transceiver buffers. The function will hold execution (loop) until the
129   TWI_ISR has completed with the
130 previous operation, then initialize the next operation and return.
131 ****
132 void TWI_Start_Transceiver( void )
133 {
134   while ( TWI_Transceiver_Busy() ) {} // Wait until TWI is ready
135   for next transmission.
136   TWI_statusReg.all = 0;
137   TWI_state = TWI_NO_STATE;
138   TWCR = (1<<TWEN) | // TWI Interface enabled.
139     (1<<TWIE)|(1<<TWINT)| // Enable TWI Interrupt and
140     clear the flag.
141     (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)| // Prepare to ACK next time
142     the Slave is addressed.
143     (0<<TWWC); // /
144   TWI_busy = 0;
145 }
146 ****
147 Call this function to read out the received data from the TWI transceiver
148   buffer. I.e. first call
149 TWI_Start_Transceiver to get the TWI Transceiver to fetch data. Then Run this
150   function to collect the
151 data when they have arrived. Include a pointer to where to place the data and
152   the number of bytes
153 to fetch in the function call. The function will hold execution (loop) until
154   the TWI_ISR has completed
155 with the previous operation, before reading out the data and returning.

```

```

141     If there was an error in the previous transmission the function will return
142     the TWI State code.
143     ****
143     unsigned char TWI_Get_Data_From_Transceiver( unsigned char *msg, unsigned char
144         msgSize )
144     {
145         unsigned char i;
146
147         while ( TWI_Transceiver_Busy() ) {} // Wait until TWI is ready
148             for next transmission.
149
149         if( TWI_statusReg.lastTransOK ) // Last transmission completed
150             successfully.
150         {
151             for ( i=0; i<msgSize; i++ ) // Copy data from Transceiver
152                 buffer.
152             {
153                 msg[ i ] = TWI_buf[ i ];
154             }
155             TWI_statusReg.RxDataInBuf = FALSE; // Slave Receive data has been
156                 read from buffer.
156         }
157         return( TWI_statusReg.lastTransOK );
158     }
158
158
161 // ***** Interrupt Handlers **** //
162 ****
163 This function is the Interrupt Service Routine (ISR), and called when the TWI
164 interrupt is triggered;
164 that is whenever a TWI event has occurred. This function should not be called
165 directly from the main
165 application.
166 ****
167 ISR(TWI_vect)
168 {
169     static unsigned char TWI_bufPtr;
170
171     switch (TWSR)
172     {
173         case TWI_STX_ADDR_ACK: // Own SLA+R has been received; ACK has
174             been returned
174 //         case TWI_STX_ADDR_ACK_M_ARB_LOST: // Arbitration lost in SLA+R/W as
175             Master; own SLA+R has been received; ACK has been returned
175             TWI_bufPtr = 0; // Set buffer pointer
176             to first data location
176         case TWI_STX_DATA_ACK: // Data byte in TWDR has been transmitted
177             ; ACK has been received
177             TWDR = TWI_buf[TWI_bufPtr++];
178             TWCR = (1<<TWEN)| // TWI Interface
179             enabled
179             (1<<TWIE)|(1<<TWINT)| // Enable TWI Interrupt
180             and clear the flag to send byte
180             (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)| // 
181             (0<<TWWC);
182             TWI_busy = 1;
183             break;

```

```

184     case TWI_STX_DATA_NACK:           // Data byte in TWDR has been transmitted
185     ; NACK has been received.
186                           // I.e. this could be the end of the
187                           transmission.
188     if (TWI_bufPtr == TWI_msgSize) // Have we transceived all expected data?
189     {
190         TWI_statusReg.lastTransOK = TRUE;           // Set status bits to
191         completed successfully.
192     }
193     else                         // Master has sent a NACK before all data
194     where sent.
195     {
196         TWI_state = TWSR;                   // Store TWI State as
197         errormessage.
198     }

199
200     TWCR = (1<<TWEN)|               // Enable TWI-
201     interface and release TWI pins
202             (1<<TWIE)|(1<<TWINT)|           // Keep interrupt
203     enabled and clear the flag
204             (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|           // Answer on next
205     address match
206             (0<<TWWC);                  //

207
208     TWI_busy = 0;      // Transmit is finished, we are not busy anymore
209     break;
210     case TWI_SRX_GEN_ACK:           // General call address has been received
211     ; ACK has been returned
212 //     case TWI_SRX_GEN_ACK_M_ARB_LOST: // Arbitration lost in SLA+R/W as
213 //     Master; General call address has been received; ACK has been returned
214 //     TWI_statusReg.genAddressCall = TRUE;
215 //     case TWI_SRX_ADR_ACK:          // Own SLA+W has been received ACK has
216     been returned
217 //     case TWI_SRX_ADR_ACK_M_ARB_LOST: // Arbitration lost in SLA+R/W as
218 //     Master; own SLA+W has been received; ACK has been returned
219 //                           // Dont need to clear
220 //     TWI_StatusRegister.generalAddressCall due to that it is the default
221 //     state.

222     TWI_statusReg.RxDataInBuf = TRUE;
223     TWI_bufPtr    = 0;                // Set buffer pointer
224     to first data location

225
226                           // Reset the TWI
227                           Interrupt to wait for a new event.
228     TWCR = (1<<TWEN)|           // TWI Interface
229     enabled
230             (1<<TWIE)|(1<<TWINT)|           // Enable TWI Interrupt
231     and clear the flag to send byte
232             (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|           // Expect ACK on this
233     transmission
234             (0<<TWWC);
235     TWI_busy = 1;

236
237     break;
238     case TWI_SRX_ADR_DATA_ACK:      // Previously addressed with own SLA+W;
239     data has been received; ACK has been returned
240     case TWI_SRX_GEN_DATA_ACK:      // Previously addressed with general call
241     ; data has been received; ACK has been returned

```

```

221     TWI_buf[TWI_bufPtr++] = TWDR;
222     TWI_statusReg.lastTransOK = TRUE; // Set flag
223     transmission successfull.
224                                         // Reset the TWI
225
226     Interrupt to wait for a new event.
227     TWCR = (1<<TWEN)| // TWI Interface
228     enabled
229     (1<<TWIE)|(1<<TWINT)| // Enable TWI Interrupt
230     and clear the flag to send byte
231     (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)| // Send ACK after next
232     reception
233     (0<<TWWC); // TWI_busy = 1;
234     break;
235
236     case TWI_SRX_STOP_RESTART: // A STOP condition or repeated START
237     condition has been received while still addressed as Slave
238                                         // Enter not addressed
239     mode and listen to address match
240     TWCR = (1<<TWEN)| // Enable TWI-
241     interface and release TWI pins
242     (1<<TWIE)|(1<<TWINT)| // Enable interrupt
243     and clear the flag
244     (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)| // Wait for new
245     address match
246     (0<<TWWC); // TWI_busy = 0; // We are waiting for a new address match, so we are not
247     busy
248
249     break;
250     case TWI_SRX_ADR_DATA_NACK: // Previously addressed with own SLA+W;
251     data has been received; NOT ACK has been returned
252     case TWI_SRX_GEN_DATA_NACK: // Previously addressed with general call
253     ; data has been received; NOT ACK has been returned
254     case TWI_STX_DATA_ACK_LAST_BYTE: // Last data byte in TWDR has been
255     transmitted (TWEA = 0); ACK has been received
256
257     // case TWI_NO_STATE // No relevant state information
258     available; TWINT = 0
259     case TWI_BUS_ERROR: // Bus error due to an illegal START or STOP
260     condition
261     TWI_state = TWSR; // Store TWI State as errormessage,
262     operation also clears noErrors bit
263     TWCR = (1<<TWSTO)|(1<<TWINT); // Recover from TWI_BUS_ERROR, this
264     will release the SDA and SCL pins thus enabling other devices to use the
265     bus
266
267     break;
268     default:
269     TWI_state = TWSR; // Store TWI State as
270     errormessage, operation also clears the Success bit.
271     TWCR = (1<<TWEN)| // Enable TWI-
272     interface and release TWI pins
273     (1<<TWIE)|(1<<TWINT)| // Keep interrupt
274     enabled and clear the flag
275     (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)| // Acknowledge on any
276     new requests.
277     (0<<TWWC); //
```

```

255     TWI_busy = 0; // Unknown status, so we wait for a new address match that
256     // might be something we can handle
257 }
}

```

D.2 MATLAB Code for Image Algorithm Prototyping

D.2.1 Image Matching Algorithms

D.2.1.1 loadimages.m

..//MATLAB/loadimages.m

```

1 left = imread('viprectification_deskLeft.png');
2 right = imread('viprectification_deskRight.png');

4 % left = imread('battery_left.bmp');
5 % right = imread('battery_right.bmp');

7 % left = imread('square_left.bmp');
8 % right = imread('square_right.bmp');

10 % left = imread('fiftycm_left.bmp');
11 % right = imread('fiftycm_right.bmp');

13 % left = imread('2objs_left.bmp');
14 % right = imread('2objs_right.bmp');

```

D.2.1.2 GetSubImage.m

..//MATLAB/GetSubImage.m

```

1 function [ SubImage ] = GetSubImage( Image, BoxSize, StartCoordinates )
2 %GETSUBIMAGE Returns a sub section of the image according to the other
3 %inputs
4 % Image - The image of which a subimage is to be taken from
5 % BoxSize - A 2x1 matrix containing the size of the subImage
6 % StartCoordinates - A 2x1 matrix with the start point of the image
7 % Dimensions - How many planes - 3 for colour, 1 for grey scale

9 XLow = StartCoordinates(1)-(BoxSize(1)/2);
10 YLow = StartCoordinates(2)-(BoxSize(2)/2);
11 if(XLow<1)
12     XLow = 1;

```

```

13    end

15    if(YLow < 1)
16        YLow = 1;
17    end

19    XHigh = XLow + BoxSize(1);
20    YHigh = YLow + BoxSize(2);
21    [~, ~, LZ] = size(Image);

23    %SubImage = zeros(BoxSize);
24    for i = XLow:XHigh
25        for j = YLow:YHigh
26            if LZ == 3
27                for z = 1:3
28                    SubImage(i-XLow+1,j-YLow+1,z) = Image(i,j,z);
29                end
30            elseif LZ == 1
31                SubImage(i-XLow+1,j-YLow+1) = Image(i,j);
32            else
33                error('Number of Dimensions "%d" are not supported', LZ);
34            end
35        end
36    end

38 end

```

D.2.1.3 SADAll.m

..../MATLAB/SADAll.m

```

1 %function [ Results ] = SADAll( Left, Right )
2 %SADALL Function to compute all SADs of an image
3 % The sum of absolute differences is calculated and returned on a mesh
4 % graph to show how well matched the sub image is to the image. A box out
5 % of the right image is taken and compared with the left image.
6 loadimages;
7 BoxSize = [50,50];
8 [~,~,C] = size(right);
9 [I,J,D] = size(left);
10 if C ~= D
11     error('Images have different number of colour planes');
12 end

14 RightSub = GetSubImage(right, BoxSize, [190,190]);

16 for i = 25:(I-25)
17     for j = 25:(J-25)
18         LeftSub = GetSubImage(left, BoxSize, [i, j]);
19         Diff = LeftSub - RightSub;

21         Results(i,j) = sum(Diff(:));
22     end
23 end

```

```

25 %Display
26 figure;
27 subplot(2,2,1);
28 imshow(left);
29 title('Left Image');

31 subplot(2,2,2);
32 imshow(right);
33 title('Right Image');

35 subplot(2,2,3);
36 imshow(RightSub);
37 title('Right Sub');

39 figure;
40 surf(Results);
41 shading flat;
42 %end

```

D.2.1.4 SSDAll.m

..../MATLAB/SSDAll.m

```

1 %function [ Results ] = SADAll( Left, Right )
2 %SADALL Function to compute all SADs of an image
3 % The sum of absolute differences is calculated and returned on a mesh
4 % graph to show how well matched the sub image is to the image. A box out
5 % of the right image is taken and compared with the left image.
6 loadimages;
7 BoxSize = [50,50];
8 [~,~,C] = size(right);
9 [I,J,D] = size(left);
10 if C ~= D
11     error('Images have different number of colour planes');
12 end

14 RightSub = GetSubImage(right, BoxSize, [190,190]);

16 for i = 25:(I-25)
17     for j = 25:(J-25)
18         LeftSub = GetSubImage(left, BoxSize, [i, j]);
19         Diff = LeftSub - RightSub;
20         Diff = Diff.^2;
21         Results(i,j) = sum(Diff(:));
22     end
23 end

25 %Display
26 figure;
27 subplot(2,2,1);
28 imshow(left);
29 title('Left Image');

```

```

31 subplot(2,2,2);
32 imshow(right);
33 title('Right Image');

35 subplot(2,2,3);
36 imshow(RightSub);
37 title('Right Sub');

39 figure;
40 surf(Results);
41 shading flat;
42 %end

```

D.2.1.5 NCC.m

..../MATLAB/NCC.m

```

2 loadimages;
3 show;
4 BoxSize = [50,50];
5 MaxConfMatches = 20;
6 %SubCoord = [145, 300];
7 figure(1);
8 %[rightSub, rect_Sub] = imcrop(right);
9 figure(2);
10 imshow(right);
11 rSubCoord = ginput(1);
12 rSubCoord = [190,190]; [rSubCoord(2), rSubCoord(1)];
13 rSubCoord = round(rSubCoord);
14 close;
15 tic;
16 rightSub = GetSubImage(right, BoxSize, rSubCoord);
17 %imshow(rightSub);
18 rightSubGray = rgb2gray(rightSub);
19 leftGray = rgb2gray(left);
20 rightGray = rgb2gray(right);
21 cL = normxcorr2(rightSubGray(:,:,1), leftGray(:,:,1));
22 figure(2);
23 % subplot(1,2,1);
24 surf(cL), shading flat;
25 title('Normalised Cross Correlation of Right Sub and Left Image');
26 toc;
27 % cR = normxcorr2(rightSubGray(:,:,1), rightGray(:,:,1));
28 % subplot(1,2,2);
29 % surf(cR), shading flat;
30 % title('Normalised Cross Correlation of Right Sub and Right Image');

32 % cD = cL - cR;
33 %
34 % figure;
35 % surf(cD), shading flat;
36 % title('Differences of the Normalised Cross Correlation of Right and Left');

```

```

38 %Find coordinates of best match.
39 [Y,X] = size(cL);
40 maxValue = 0;
41 LeftMatchCoord = [0,0];
42 NumConfidentMatches = 0;

44 for i = 1:X
45     for j = 1:Y
46         Val = cL(j,i);
47         if Val > 0.9
48             NumConfidentMatches = NumConfidentMatches + 1;
49         end
50         if Val > maxValue
51             maxValue = Val;
52             LeftMatchCoord = [j-(BoxSize(1) / 2 ), i-(BoxSize(2) / 2 )];
53         end
54     end
55 end

57 Result = [maxValue, LeftMatchCoord];
58 figure(1);
59 if NumConfidentMatches >= 1 && NumConfidentMatches < MaxConfMatches
60     left(LeftMatchCoord(1)-(BoxSize(1)/2):LeftMatchCoord(1)+(BoxSize(1)/2),
61     LeftMatchCoord(2)-(BoxSize(2)/2)=255;
62     left(LeftMatchCoord(1)-(BoxSize(1)/2):LeftMatchCoord(1)+(BoxSize(1)/2),
63     LeftMatchCoord(2)+(BoxSize(2)/2)=255;
64     left(LeftMatchCoord(1)-(BoxSize(1)/2),LeftMatchCoord(2)-(BoxSize(2)/2):
65     LeftMatchCoord(2)+(BoxSize(2)/2)=255;
66     left(LeftMatchCoord(1)+(BoxSize(1)/2),LeftMatchCoord(2)-(BoxSize(2)/2):
67     LeftMatchCoord(2)+(BoxSize(2)/2))=255;

68     right(rSubCoord(1)-(BoxSize(1)/2):rSubCoord(1)+(BoxSize(1)/2),rSubCoord(2)
69     -(BoxSize(2)/2))=255;
70     right(rSubCoord(1)-(BoxSize(1)/2):rSubCoord(1)+(BoxSize(1)/2),rSubCoord(2)
71     +(BoxSize(2)/2))=255;
72     right(rSubCoord(1)-(BoxSize(1)/2),rSubCoord(2)-(BoxSize(2)/2):rSubCoord(2)
73     +(BoxSize(2)/2))=255;
74     right(rSubCoord(1)+(BoxSize(1)/2),rSubCoord(2)-(BoxSize(2)/2):rSubCoord(2)
75     +(BoxSize(2)/2))=255;

76 subplot(1,2,1);
77 imshow(left);
78 subplot(1,2,2);
79 imshow(right);
80 % LeftMatchCoord
81 % rSubCoord
82 % NumConfidentMatches
83 Distance = Range(rSubCoord(2), LeftMatchCoord(2));
84 sprintf('Distance to Object = %d metres', Distance)
85 elseif NumConfidentMatches >= MaxConfMatches
86     title(sprintf('Too many matches found : %d', NumConfidentMatches));
87 else
88     title(sprintf('No Reliable Match Found'));
89 end

```

References

- Atmel Corporation. *AVR311: TWI Slave*, 2007.
- Atmel Corporation. *AT32UC3C0512C Datasheet*, 2012a.
- Atmel Corporation. *ATMega644P Datasheet*, 2012b.
- Atmel Corporation. [Atmel software framewrok](#), 2012c.
- Atmel Corporation. *ATXMega256A3BU Datasheet*, 2012d.
- Electronic Lives Manufacturing. [Fatfs - generic fat file system module](#), 2012.
- Farnell. [Farnell online store](#), 2012.
- François Fleuret, Jérôme Berclaz, Richard Lengagne, and Pascal Fua. Multi-camera people tracking with a probabilistic occupancy map. Technical report, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 2007.
- Wayne Fulton. [Image file formats - jpg, tif, png, gif. which to use?](#), 2010.
- 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.
- Microsoft. [Xbox 360 kinect](#), 2012.
- Jernej Mrovlje and Damir Vrančić. Distance measuring based on stereoscopic pictures. Technical report, University of Ljubljana, 2008.
- OmniVision. *OmniVision Serial Camera Control Bus (SCCB) Functional Specification*, 2007.

- Phillips. *PCA9542A : 2-channel I²C-bus multiplexer and interrupt logic*, 2009.
- M. Ruhnke, R. Kummerle, G. Grisetti, and W. Burgard. Highly accurate maximum likelihood laser mapping by jointly optimizing laser points and robot poses. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2812–2817. IEEE, 2011.
- 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.
- Sebastian Thrun and Michael Montemerlo. The graphslam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, pages 403–428, 2006.
- 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.
- K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, Anchorage, AK, USA, May 2010. Software available at <http://octomap.sf.net/>.
- 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.