UNIVERSITY OF SOUTHAMPTON

FACULTY OF PHYSICAL AND APPLIED SCIENCES

Electronics and Computer Science

Supervisor: Prof. Steve Gunn

Second Examiner: Prof. Mark Zwolinski

Two Dimensional Stereoscopic Mapping Robot

by

Henry S. Lovett

A project report for a Part Three Project for the award of BEng Electronic Engineering

December 5, 2012

Turn off iNotes!

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL AND APPLIED SCIENCES
Electronics and Computer Science

TWO DIMENSIONAL STEREOSCOPIC MAPPING ROBOT

by Henry S. Lovett

This paper describes the research, designing and building of a stereoscopic mapping robot. Mapping robots usually utilise Infra-red or laser range finders to do the distance calculations. By using two cameras, distances to objects can be calculated. The end goal is to build up an occupancy map which shows the state of an explored area as either unknown, free or occupied.

Contents

List of Figures

List of Tables

Listings

Nomenclature

I^2CInter-Integrated Circuit
TWI Two Wire Interface
SCCB Serial Camera Control Bus
kB KiloBytes
φ_0 Field of View of the Camera
B Seperation Distance of two Cameras
i, j Pixel Index of an Image

Chapter 1

Introduction

The Introduction to my Report . . .

The initial idea of the project was taken from Pirobot(Goebel (2012)).

what it will do. Define everything. Use. Very general

General - mapping robots.

stereovision - uses etc.

other similar projects

why mine is important

This report documents the design, test and build 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 unkown area and build up an occupancy map of the area.

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

An Occupancy map is a representation of an area where a location is given a value of either *free, occupied* or *unkown.* CITE THIS. Three dimensional occupancy maps can also be generated, such as the OctoMapWurm et al. (2010) and 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 leads on to be able to conduct an application that requires knowledge of the area. An algorithm used to build up the occupancy map is the S.L.A.M. algorithm Thrun and

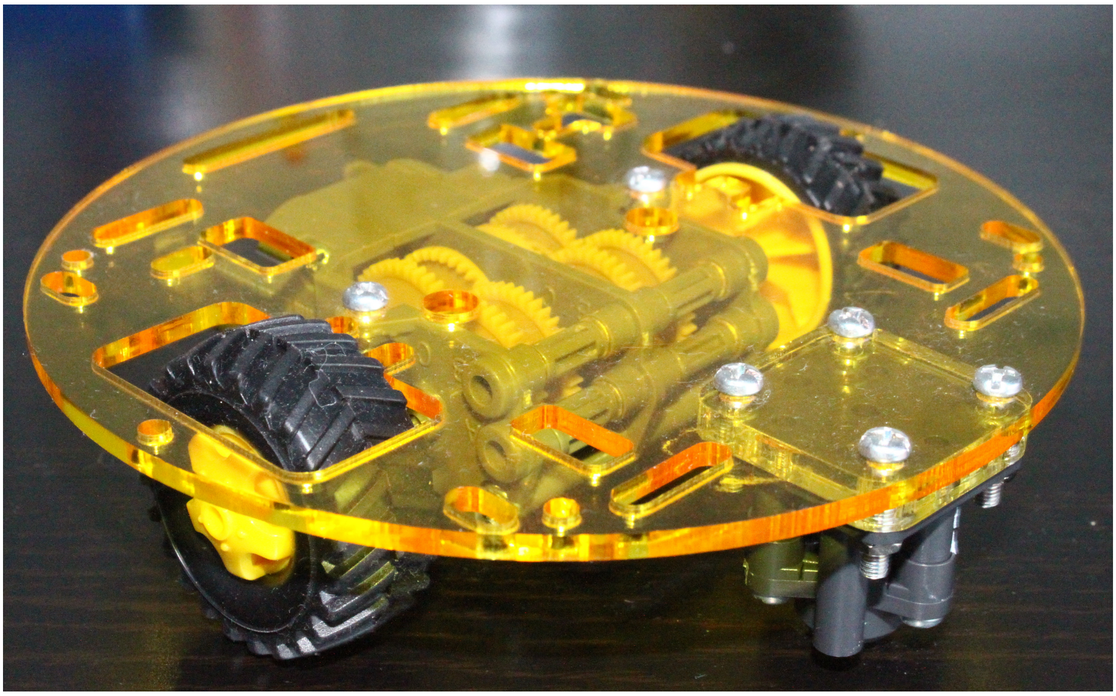


Figure 1.1: The base of the robot will use

Montemerlo (2006) and is used in Se et al. (2002). Accurate mapping robots tend to used laser range finders Ruhnke et al. (2011).

Stereovision is a small section of Computer Vision which is widely used in many applications including the Kinect Microsoft (2012) where stereo vision is used to locate a game player to involve them more in 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 discussed will use the base seen in figure 1.1 and use two OmniVision OV7670 cameras delivering up to VGA format.

Chapter 2

Research

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

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

2.1 Hardware Research

Talk about why I chose to develop with AVRs, comparison with other uControllers. Why I used the OV7670 Camera etc etc

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 three main brands of microcontrollers present in the consumer market: ARM, Atmel AVRs and PICs.

Do more research into ARM stuff

ARM is an architecture which is developed by ARM Holdings. ARM devices come in a many varieties, ARM9, ARM7, Strong ARM, ARM Cortex etc. While ARM Holdings do not fabricate and sell the devices themselves, many companies such as Texas Instruments, use the architecture and fabricate their own devices. For this comparison, the ARM7 and ARM9 will be looked at. ARM cores tend to be 32-bit and have a high clock speed. They are a RISC Harvard architecture.

Atmel have a variety of products in the microcontroller market. They range from 8-bit, low clock speed packages for the hobbyist (ATMega, ATTiny), an improved 8-bit variant, XMega, and a 32-bit design, AT32UC3C. XMegas and AVR32s tend to have higher clock speeds than the ATMegas, around 30MHz for Xmegas and 60MHz for AVR32s. The AVR core is also a Harvard RISC architecture, but mainly 8-bit. Atmel devices have on board hardware implementations of common uses - I^2C, SPI, ADCs as well as a number of different memories: Flash, EEPROM and SRAM. An AT32UC3C, ATXmegaA3BU and ATMega644P will be compared in this section.

Much point wasting words on PIC?

PICs are useless...

2.2 Image Algorithms

2.2.1 Comparison Algorithms

2.2.2 Detection Algorithms

It will be necessary to be able to work out what in the image are objects. For this, a number of detection algorithms can be used:

1. Corner Detection
2. Edge Detection

2.2.3 Corner Detection

A common edge detection algorithm is the Harris Corner Detector Nixon (2005). This works by placing a window over the image and measuring the intensity. If the intensity changes in both vertical and horizontal directions, there is a corner. If the window is on an edge, the intensity will only change in the direction perpendicular to the edge. This method will be useful to detect an object. However, this may not be able to detect a tall, vertical object, such as the edge of a wall.

2.2.4 Edge Detection

There are many edge detection algorithms. The

Chapter 3

Initial Hardware and Firmware Development

For initial development, an *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 parts listed below:

1. Camera Code
2. SD Card
3. Motor Control
4. PCB Development

3.1 Camera

The camera that was used was an OV7670 by OmniVision. It is mounted onto a break out board and connected to a AL422B FIFO Buffer. The breakout board also had all passive components and a 24MHz clock mounted. The schematic for the device can be seen in Appendix A.

Original code for the camera operation was given to me by Steve Gunn, of which I used to gain the operation required.

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 used of the built in interrupt driven I^2C interface (named TWI in Atmel AVRs)[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 contain green intensity and 11:15 represent the red intensity. 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 frame. This makes it difficult for an AVRs (ATMegas typically clocked at 12-16MHz) to be able to respond to the camera quick enough. This highlights the importance of the necessity of the 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.

To obtain the data from the buffer, the AVR manually pulses the read clock and stores the data on the input port. All the data is then read in one pixel at a time.

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

Firstly, data was sent straight to a desktop computer via a COM Port. A simple desktop program written in C# to store all the data and convert binary into a Bitmap image. This method was slow, taking around 30 seconds to transmit one uncompressed image.

The second option then was to use extra memory connected to the microcontroller. An SD card was decided to be used in as a FAT file system. This will allow data to be looked at by a user on a computer of image files and log files. This is discussed in section 2.

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

3.1.2 Dual Camera Operation

In order for stereovision to be successful, two cameras separated by a horizontal distance will need to be driven at the same time to obtain photos of the same time frame.

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. Two 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 is no contention on the bus. 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 MUX and select whether to pass the bus to bus 0, bus 1 or not allow the data to be transferred. This saves processor time, but means a write operation has to be done to select the camera bus before being able to write to the camera. This slows down the operation but not as much as using SCCB. The main disadvantage to the I^2C MUX is the extra hardware needed. Firstly, the MUX itself, but also 7 extra resistors to pull up the two extra buses and the three interrupt lines must be added.

Overall, the disadvantages posed by using a MUX are small and simplify the operation and reduce the code size so an I^2C MUX 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 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

Fix the FatFS Reference

	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))

Sort Reference Out

To use the SD card, the FATFS library (Manufacturing (2012)) was used. The library supplies all the functions for writing a FAT File System in the files *ff.c*, *ff.h*, *ffconf.h*, *diskio.c*, *diskio.h* and *integer.h*. The *diskio.h* functions control what device is being used - SD/MMC Card, USB drive etc. The *ff.h* header contains all the functions to write to in a FAT File system.
An SD card was chosen to be used 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 to obtain the correct format. To avoid compression, and thereby save computational time, Bitmap was decided to be used 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 B 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 protoyping of image algorithms in a more powerful environment. Figure 3.1 shows a photo taken by the OV7670 and stored on a SD card.

Figure 3.1: An Example Image taken using the OV7670 and stored as a Bitmap on the SD Card

	Port A	Port B	Port C	Port D
0	Data 0	SD Write Protect	I^2CSCL	No Connection
1	Data 1	SD Card Detect	I^2CSDA	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 0	Write Reset 0

Table 3.2: Pin Connections of the ATMega 644P for Dual Camera Operation.

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 Input / Output pins remaining, an ATMega 168 was added on the I^2Cbus to act as a port extender. The 168 accepts a read or write, places the write data on Port D and reads in the lower nibble of Port C. When a button is pressed, this is stored in the 168 until a read has been done. This is so the master doesn't miss any button presses while busy doing lengthy operations such as writing an image. The code is based on Corporation (2007) written for IAR Compiler. This code was altered to compile with GCC under Atmel Studio. AVRs contain a hardware based I^2Cprotocol 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.

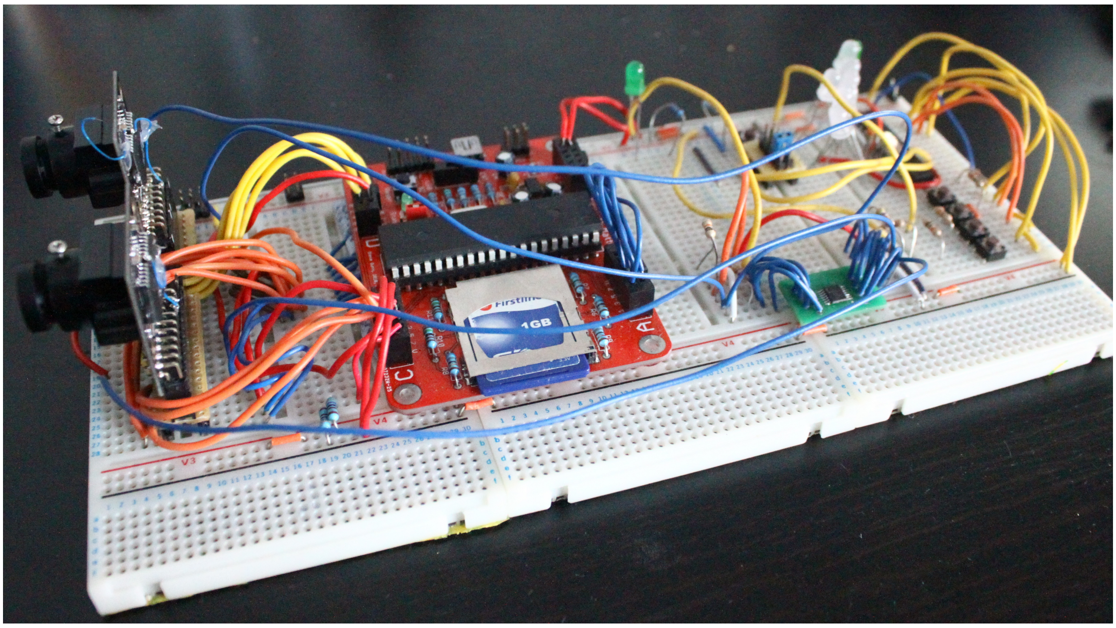


Figure 3.2: Prototype of Dual Camera operation.

3.3 Motor Control

do something of this SOON

3.4 Circuit and PCB Development

3.4.1 Il Matto Development

Figure A.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 3.2.

Chapter 4

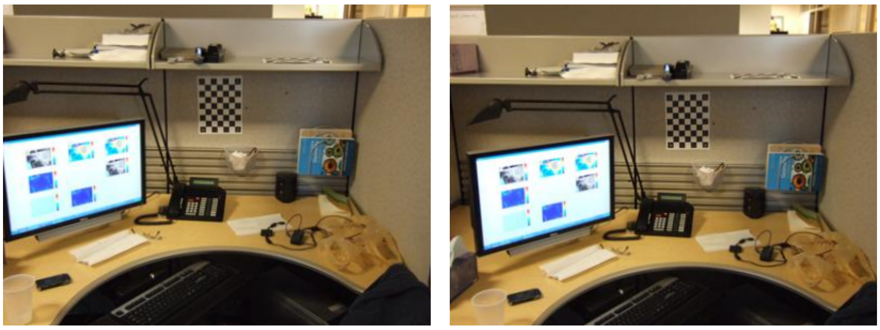
Investigation into Vision Algorithms

4.1 Comparison

find some references to back these claims up

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.) and normalised cross correlation (N.C.C.). 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 $50\times$ 50 pixels of the same part of the image.

Maybe do a basic 5x5 example for each?



(a) Left Image

(b) Right Image

Figure 4.1: Stereoscopic Test Images from MATLAB Examples

4.1.1 Sum of Absolue Differences

Given two indentically sized matricies, A, B of dimensions I, J, SAD is defined as

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

This method 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

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

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). Again, a low result is a match in this case.

4.1.3 NCC

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

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

Find a source for this equation

No date on Reference

NCC is very similar to cross correlation, but normalised to reduce the error if one image is brighter than the other. It is common in computer vision (Tsai and Lin (2003)) as cross correlation is a common operation in DSP 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 with division in it and a square root 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 operations slow 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. Fi

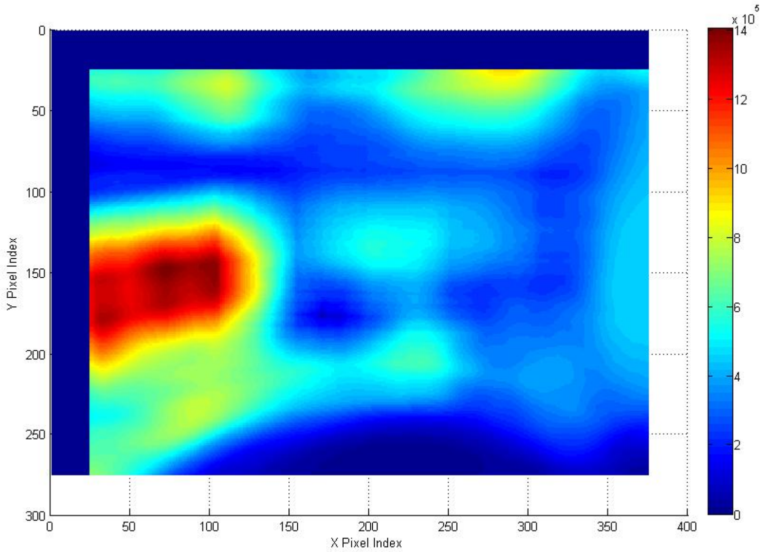
Each of the graphs show the correct area being indentified as a match, but this also highlights the downfalls of the SAD and SSD. The figures in figure 4.2 are orientated to match the orientation of the images in figure 4.1. Each of the images is tested by attempting to match the phone from the test 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.

reference for sub pixel problem?

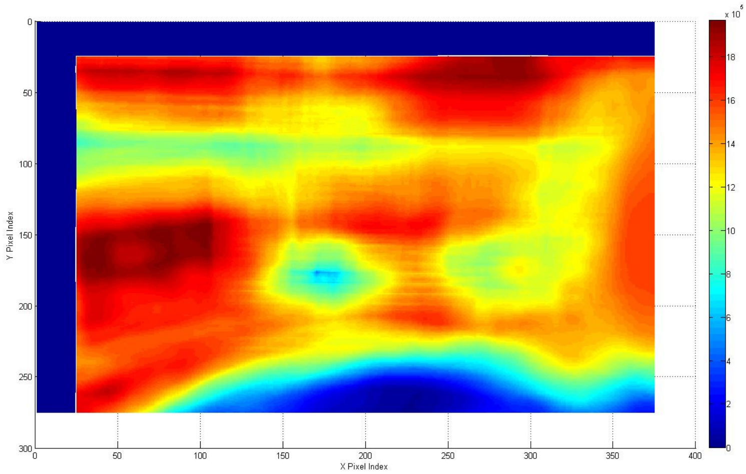
SAD results in figure 4.2(b) show large areas of matching. The actual match is at (170, 175) and a minimum does occur at this position as expected of a value of 5.66×10^4. However, along the bottom of the image where a dark area occurs in the lower part of figures 4.1 below the desk, the SAD algorithm 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, again, there is thought to be a large match correlation between the dark area under the desk where the actual lowest value of 2.768 is at $(225, 274)$. This, again, is a false match and is a downfall of this algorithm.

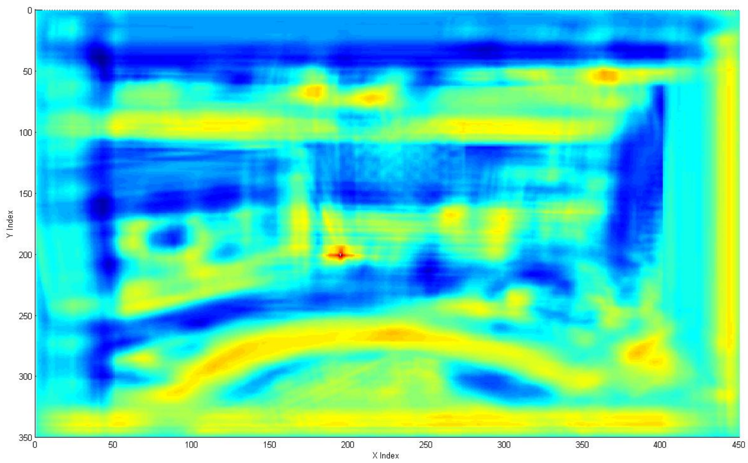
The NCC results are visible in figure 4.2(c). A match can be seen at coordinate $(195, 201)$ with a peak value of 0.9654. The coordinate is different to the previous results because the cross correlation works over the boundary of the image creating more results. The dimensions of the image are 300 $\times$ 400, but the NCC returns an data set of dimensions 350 $\times$ 450 when using a box size of 50 $\times$ 50. To get the actual match, half of the box size must be subtracted from the returned coordinate. This means the match occurs at $(170, 176)$.



(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

4.1.5 Conclusion

It can be seen 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 guaranteed to always be the case. Therefore further development of the matching algorithm will start with using the Normalised Cross Correlation. There is a compromise of complexity for reliability, of which reliability is more desirable. Cross correlation is also a large area of research, so optimised algorithms do exist.

4.2 Range Finding

Derive the range finding equations and test them

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 3

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

4.2.1.1 Object is between the Cameras

Derivation from Mrovlje and Vrančić (2008).

$$B = B_1 + B_2 = D\tan(\varphi_1) + D\tan(\varphi_2) \tag{4.4}$$

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

$$D\tan(\frac{\varphi_0}{2}) = x_0/2 \tag{4.6}$$

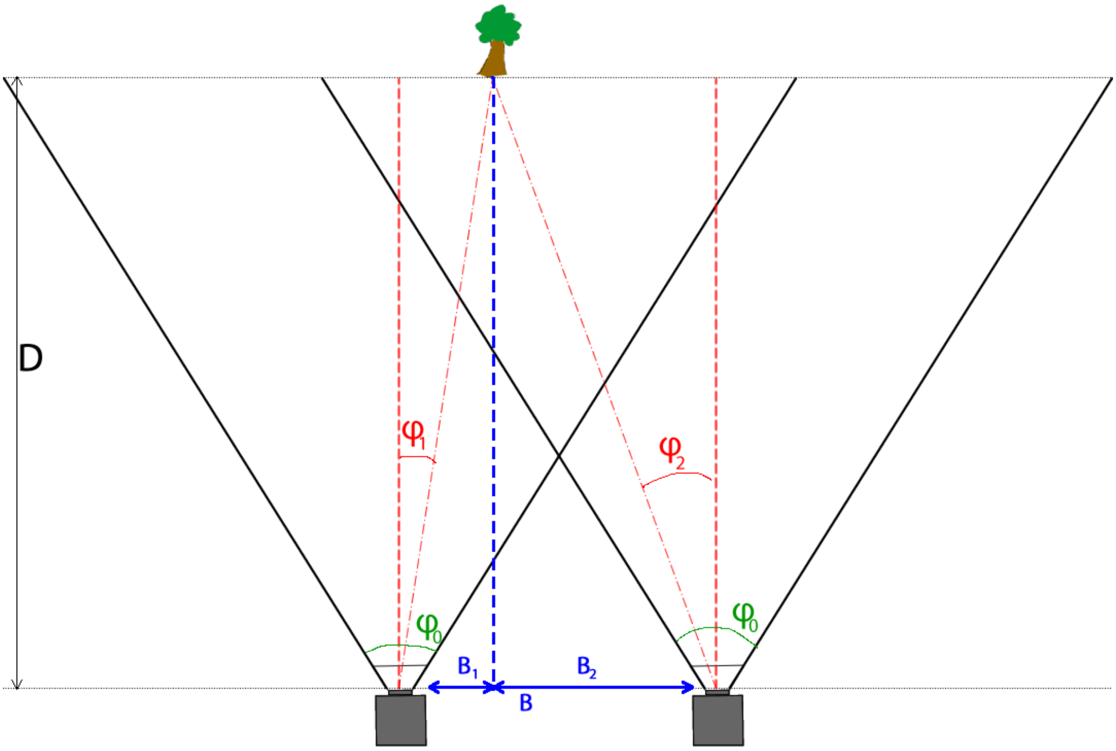


Figure 4.3: Problem 1 - Object is between the Cameras

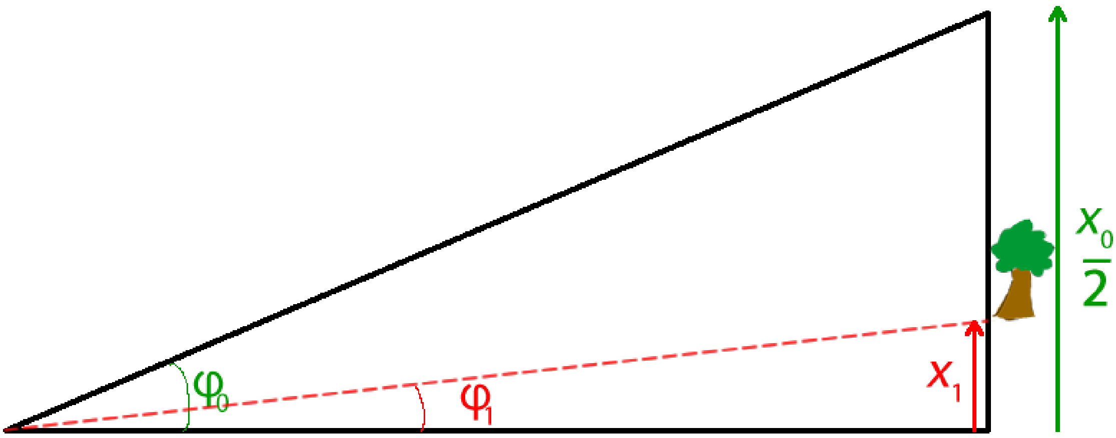


Figure 4.4: Problem 1 : Left Camera Simplified

$$D\tan(\varphi_1) = x_1 \tag{4.7}$$

Dividing (4.7) by (4.6)

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

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

It can also be shown that for the right camera:

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

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

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

4.2.1.2 Object is to the same side in each camera

Derivation is based on the derivation from Tjandranegara (2005). Using figure 4.5:

$$D.\tan(\varphi_1) = x_1 \tag{4.12}$$

$$D.\tan(\frac{\varphi_1}{2}) = \frac{x_0}{2} \tag{4.13}$$

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

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

and similarly

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

$$\theta = \varphi_2 - \varphi_1 \tag{4.17}$$

Using the sine equality rule:

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

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

$$D = \cos(\varphi_1).R \tag{4.20}$$

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

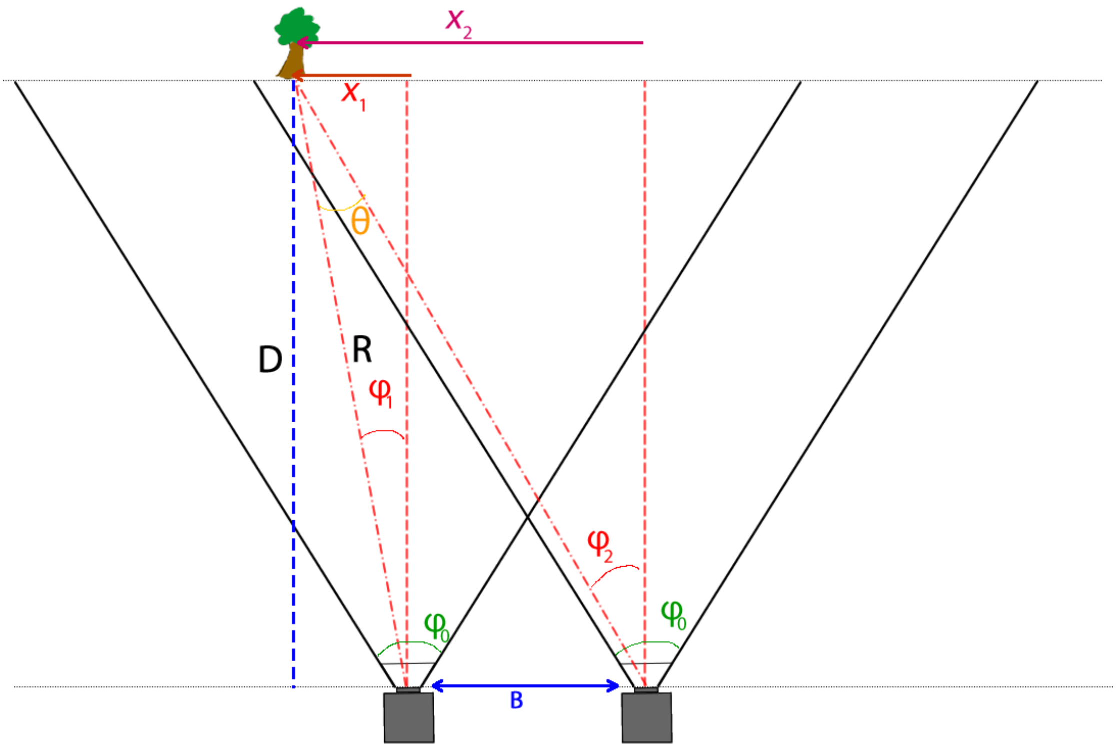


Figure 4.5: Problem 2 - Object is to the same side in both cameras

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

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:

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

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:

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

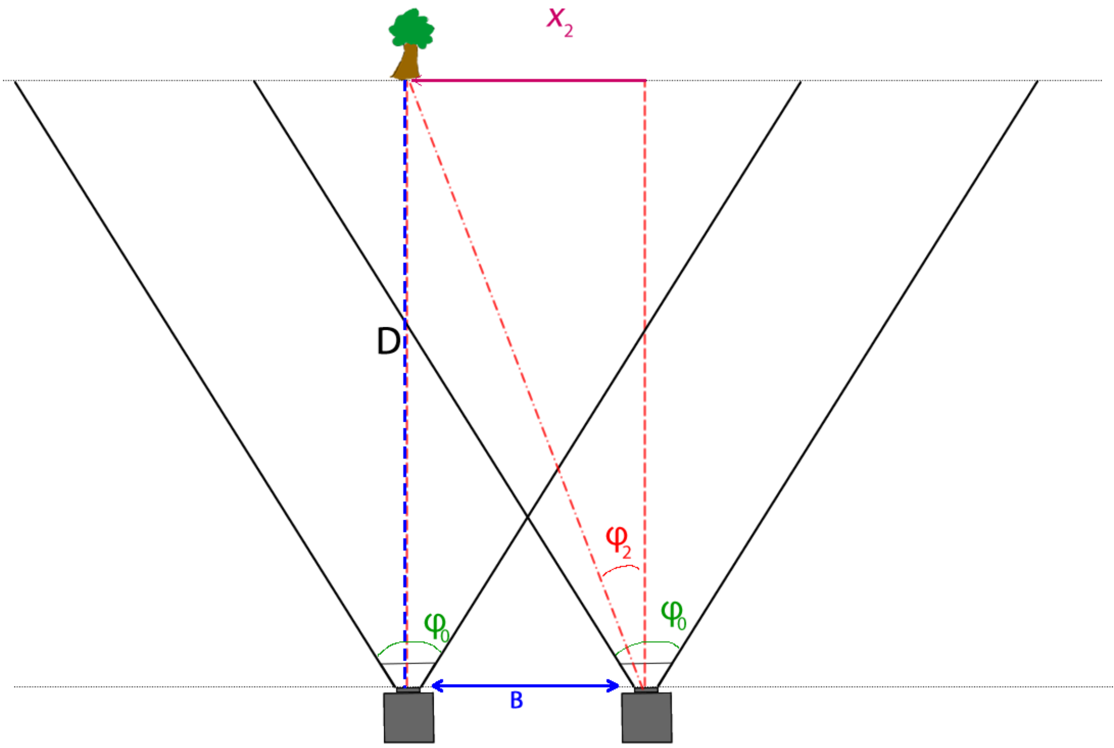


Figure 4.6: Problem 3 - Object is directly in front of a camera

Object is to the same side in both images:

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

Object is directly in front of a camera:

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

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

Chapter 5

Conclusions and Further Work

It works.

Appendix A

Circuit Diagrams

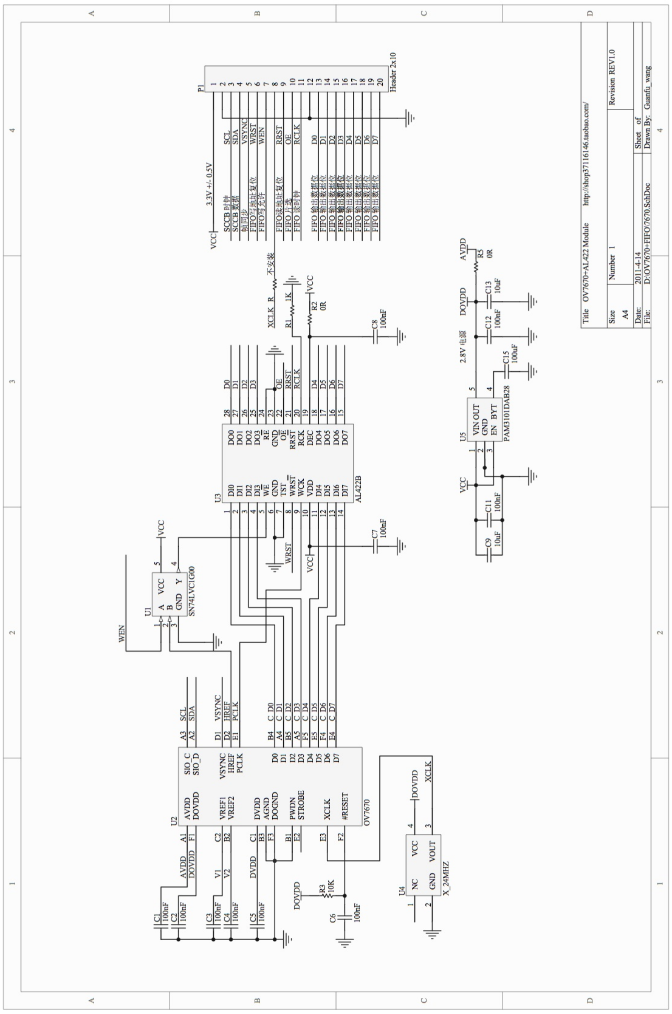


Figure A.1: The circuit diagram for the OV7670 breakout board

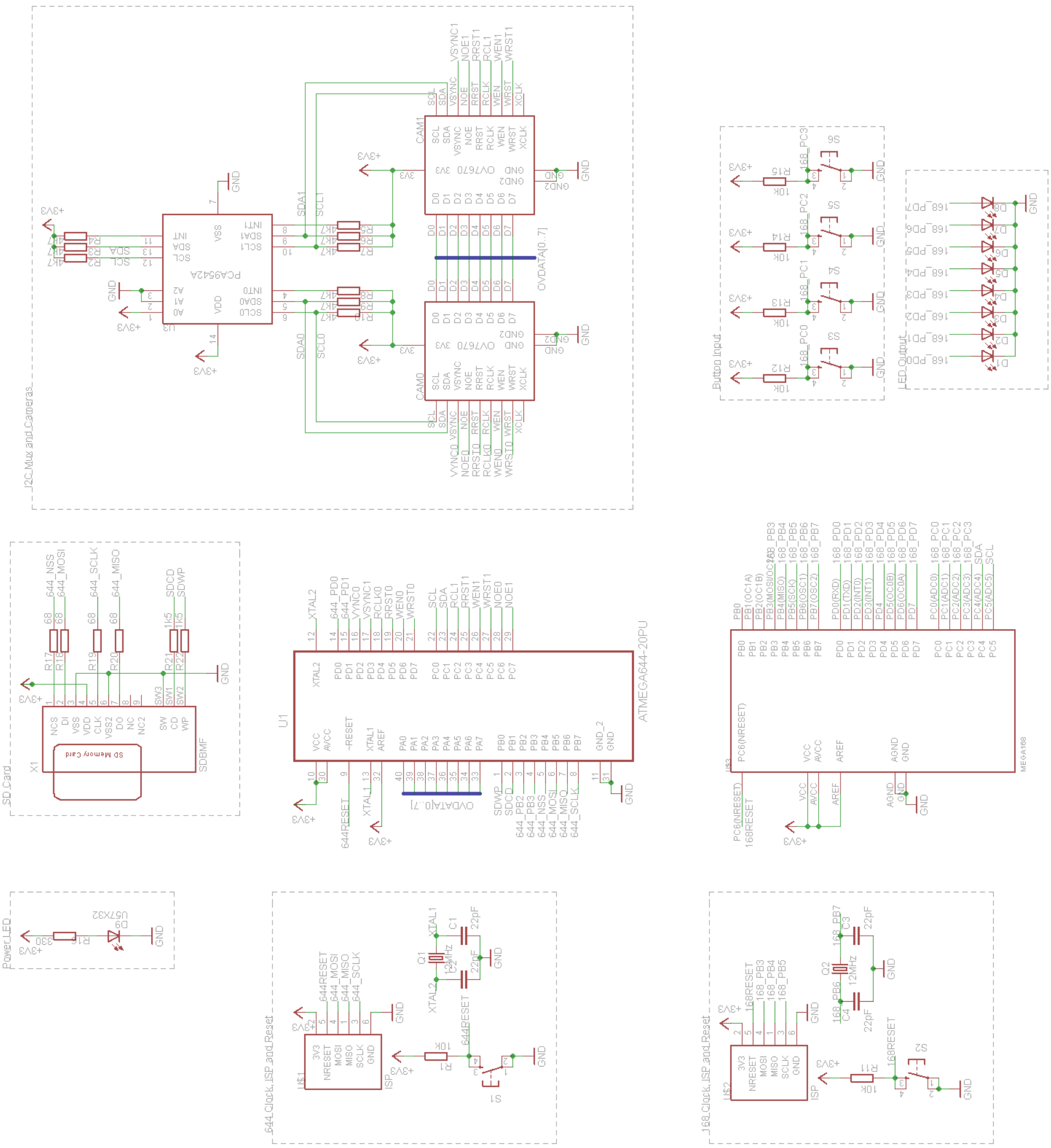


Figure A.2: The circuit diagram for Dual Cameras using the Il Matto Board

Appendix B

Bitmap File Format

B.1 Bitmap File Format

Table B.1: Feasible triples for highly variable Grid, MLMMH.

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
	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

Continued on next page

[1]This is different to the 225kB said in Table 3.1 due to ommitting many optional fields

Table B.1 – continued from previous page

Section	Field	Description	Size (Bytes)	Value (hex)
	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
	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

Continued on next page

Table B.1 – continued from previous page

Section	Field	Description	Size (Bytes)	Value (hex)
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 C

Source Code

C.1 C Code for AVR

should I include FatFS Files?

C.1.1 Dual Camera Operation

C.1.1.1 main.c

../Code/DualOV7670/main.c

```
/*
 * DualOV7670.c
 *
 * Created: 09/11/2012 11:43:13
 *  Author: hl13g10
 */
#include "Config.h"



//static FILE mystdout = FDEV_SETUP_STREAM(File_Write_Printf, NULL,
    _FDEV_SETUP_WRITE);
//FatFS Variables
FILINFO Finfo;
FATFS Fatfs[_VOLUMES];    /* File system object for each logical drive */
//FIL Files[2];       /* File object */
uint8_t StatusReg;
// char Line[100];        /* Console input buffer */
//char Buff[100];     /* Working buffer */
char ImageRName[20];
char ImageLName[20];
#define STATUS_OKAY     0x01
#define STATUS_SDOkay   0x02
#define STATUS_CAM0Okay   0x04
#define STATUS_CAM1Okay   0x08
```

```c
#define STATUS_READY     0x10
#define STATUS_CAPTURING   0x20
#define STATUS_Exit_Bad    0x80

#define Button_Capture     0
#define Button_Exit     3
unsigned char UI_LEDs(uint8_t LED)
{
  unsigned char mesbuf[TWI_BUFFER_SIZE];
  mesbuf[0] = (0x15 << TWI_ADR_BITS) | (FALSE << TWI_READ_BIT);
  mesbuf[1] = 0x10;
  mesbuf[2] = LED;
  TWI_Start_Transceiver_With_Data(mesbuf, 3);
  while(TWI_Transceiver_Busy()) ;
  return TWI_statusReg.lastTransOK;
}
unsigned char UI_Buttons()
{
  unsigned char messageBuf[TWI_BUFFER_SIZE]; //Initialise a buffer
  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.
  messageBuf[1] = 0x20;             // The first byte is used for the command
  TWI_Start_Transceiver_With_Data( messageBuf, 2 );
  _delay_us(250);
  // Request/collect the data from the Slave
  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.
  TWI_Start_Transceiver_With_Data( messageBuf, 2 );

  // Get the received data from the transceiver buffer
  TWI_Get_Data_From_Transceiver( messageBuf, 2 );
  return messageBuf[1];
}
ISR(TIMER0_COMPA_vect)
{
  disk_timerproc(); /* Drive timer procedure of low level disk I/O module */
//  if(!TWI_statusReg.lastTransOK) //if the last TWI transmission failed, reset the protocol
//    TWI_Start_Transceiver();
//  if(!TWI_Transceiver_Busy())
//    UI_LEDs(StatusReg);
}
int main(void)
{
  unsigned long int a = 0;
  uint8_t b = 0;
  FRESULT fr;
  uint8_t PhotoCount = 0;
  TWI_Master_Initialise();
  IO_Init();
  sei();
  PCA9542A_Init();

  StatusReg = STATUS_OKAY;
  UI_LEDs(StatusReg);


  fr = f_mount(0, &Fatfs[0]);
  if(fr != FR_OK)
```

```
  {
    StatusReg |= (STATUS_Exit_Bad);
    StatusReg &= ~(STATUS_OKAY);
    UI_LEDs(StatusReg);
    return 0;
  }
  else
    StatusReg |= STATUS_SDOkay;
  UI_LEDs(StatusReg);

  fr = f_open(&Files[0], "/log.txt", FA_WRITE|FA_CREATE_ALWAYS);
  if(fr != FR_OK)
  {
    StatusReg |= (STATUS_Exit_Bad);
    StatusReg &= ~(1<<STATUS_SDOkay) | (1<<STATUS_OKAY);
    UI_LEDs(StatusReg);
    return 0;
  }
  UI_LEDs(StatusReg);

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

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

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

  PCA9542A_SetChannel(CH0);
  b = OV7670_init();
  if(b == 0)
    StatusReg |= STATUS_CAM0Okay;
  UI_LEDs(StatusReg);
  PCA9542A_SetChannel(NO_SELECT);
  sprintf(Buff, "OV7670_0 Initialise result : %d\n", b);
  f_write(&Files[0], &Buff, 33, &a);
  FIFO_init();

  //f_close(&Files[0]);
  StatusReg |= STATUS_READY;
  UI_LEDs(StatusReg);
  _delay_ms(250);
  uint8_t Input;

    while(1)
    {
    Input = (~UI_Buttons() & 0x0F);//Data is received negative
    if(Input)//if a button has been pressed
```

```
      {
        _delay_ms(250);
        sprintf(Buff, "Button Received : %d\n", Input);
        f_write(&Files[0], Buff, 21, &a);

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

        switch(Input)
        {
          case (1<<Button_Capture):
            StatusReg |= STATUS_CAPTURING;
            UI_LEDs(StatusReg);
            //Reset both buffers
            FIFO_Reset(0);
            FIFO_Reset(1);
            f_write(&Files[0], "Capturing Images...\n", 20, &a);
            LoadImagesToBuffer();//Load both images

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

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

            //Create Bitmap for image 1
            f_open(&Files[1],"image_l.bmp", FA_CREATE_ALWAYS | FA_WRITE);
            f_write(&Files[0], "Created image1 file.\n", 22, &a);
            f_lseek(&Files[1], BMPFileSize);
            f_lseek(&Files[1], 0);
            f_close(&Files[1]);
            f_write(&Files[0], "Extended image1 file.\n", 22, &a);
            //Get image 0
            f_open(&Files[1], "Image_r.bmp", FA_WRITE);
            while (2 == GetImageIfAvailiable(&Files[1], 0)) ;
            f_close(&Files[1]);
            f_write(&Files[0], "Captured image0.\n", 17, &a);
            //get image 1
            f_open(&Files[1], "image_l.bmp", FA_WRITE);
            while (2 == GetImageIfAvailiable(&Files[1], 1)) ;
            f_close(&Files[1]);
            f_write(&Files[0], "Captured image1.\n", 17, &a);
            StatusReg |= STATUS_READY;
            StatusReg &= ~STATUS_CAPTURING;
            UI_LEDs(StatusReg);
            break;//break case(1<<ButtonCapture)

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

            StatusReg = 0x41;
            UI_LEDs(StatusReg);
            return 0;//Q
        }//End switch
      }//End if(Input)
```

```
    else
    {
      StatusReg |= STATUS_READY;
      UI_LEDs(StatusReg);
      _delay_ms(250);//wait
    }//end else(Input)
    }//End while(1)
}//End Main
```

C.1.1.2 Bitmap.h

../Code/DualOV7670/Bitmap.h

```
/*
 * Bitmap.h
 *
 * Created: 29/10/2012 11:31:11
 *  Author: hslovett
 */


#ifndef BITMAP_H_
#define BITMAP_H_

#define BMPHEADERSIZE 14
#define DIBHEADERSIZE 124 //v5
#define FILESIZE 153738

#include "ff.h"
#include "Config.h"


FRESULT WriteBMPHeader(FIL *File);
FRESULT WriteDIBHeader(FIL *File);


#endif /* BITMAP_H_ */
```

C.1.1.3 Bitmap.c

../Code/DualOV7670/Bitmap.c

```
/*
 * Bitmap.c
 * Contains Methods to write the Bitmap and DIB Header. File must already be
   open.
 * Created: 29/10/2012 11:30:58
 *  Author: Henry Lovett (hl13g10@ecs.soton.ac.uk)
 */
#include "Bitmap.h"

uint8_t DIBHead[DIBHEADERSIZE] = {  0x7C, 0x00, 0x00, 0x00, //Number of bytes
```

```
                   0x40, 0x01, 0x00, 0x00, //Width - 320
                   0xF0, 0x00, 0x00, 0x00, //Height - 240
                   0x01, 0x00,       //Planes
                   0x10, 0x00,       //Bits per Pixel
                   0x03, 0x00, 0x00, 0x00, //Compression
                   0x00, 0x86, 0x25, 0x00, //Size of Raw Data
                   0x13, 0x0B, 0x00, 0x00, //Horizontal Resolution
                   0x13, 0x0B, 0x00, 0x00, //Vertical Resolution
                   0x00, 0x00, 0x00, 0x00, //Colours in Palette
                   0x00, 0x00, 0x00, 0x00, //Important Colours
                   0x00, 0xF8, 0x00, 0x00, //Red Mask
                   0xE0, 0x07, 0x00, 0x00, //Green Mask
                   0x1F, 0x00, 0x00, 0x00, //Blue Mask
                   0x00, 0x00, 0x00, 0x00, //Alpha Mask
                   0x01, 0x00, 0x00, 0x00, //Colour Space Type
                   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                   0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
                   0x00, 0x00, 0x00, 0x00, //Gamma Red
                   0x00, 0x00, 0x00, 0x00, //Gamma Green
                   0x00, 0x00, 0x00, 0x00, //Gamma Blue
                   0x03, 0x00, 0x00, 0x00, //Intent - Photo
                   0x00, 0x00, 0x00, 0x00, //ICC Profile Data
                   0x00, 0x00, 0x00, 0x00, //ICC Profile Size
                   0x00, 0x00, 0x00, 0x00}; //Reserved

uint8_t BMPHeader[BMPHEADERSIZE] = {  0x42, 0x4D,
                     0x8A, 0x58, 0x02, 0x00, //Size
                     0x00, 0x00, 0x00, 0x00, //Reserved
                     0x8A, 0x00, 0x00, 0x00 //Offset to Pixel Array
                     };



FRESULT WriteBMPHeader(FIL *File)
{
  uint32_t p;
  FRESULT f;

  f_lseek(File, 0);
  f = f_write(File, BMPHeader, BMPHEADERSIZE, &p);

  return f;
}

FRESULT WriteDIBHeader(FIL *File)
{
  uint32_t p;
  FRESULT f;

  f_lseek(File, BMPHEADERSIZE);//place just after the bitmap header
  f = f_write(File, DIBHead, DIBHEADERSIZE, &p);
  return f;
```

```
}
```

C.1.1.4 Config.h

../Code/DualOV7670/Config.h

```
/*
 * Config.h
 *
 * Created: 25/10/2012 21:58:56
 *  Author: hslovett
 */
#define F_CPU 12000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <util/delay.h>
#include "TWI_Master.h"
#include "ff.h"
#include "diskio.h"
#include "Bitmap.h"
#include "DualCameras.h"
#include "PCA9542A.h"
#ifndef CONFIG_H_
#define CONFIG_H_


void IO_Init(void);

#define TRUE 1
#define FALSE 0

FIL Files[2];
char Buff[1024];

#define BMPFileSize   153738
#define RGBFileSize   153600
//////////////////////////////////////////////////////////////////////////
//  Port A
//////////////////////////////////////////////////////////////////////////
#define FIFO_AVR_DPRT   DDRA
#define FIFO_AVR_PORT   PORTA
#define FIFO_AVR_PINP   PINA
//////////////////////////////////////////////////////////////////////////
//  Port B
//////////////////////////////////////////////////////////////////////////
#define SD_WP     PB0
#define SD_CD     PB1
//#define         PB2
//#define         PB3
#define SPI_nSS_SD    PB4
#define SPI_MOSI    PB5
#define SPI_MISO    PB6
```

```
#define SPI_SCK      PB7
//////////////////////////////////////////////////////////////////////////
//  Port C
//////////////////////////////////////////////////////////////////////////
#define TWI_SCL      PC0
#define TWI_SDA      PC1
#define FIFO_RCLK_1    PC2
#define FIFO_nRRST_1   PC3
#define FIFO_WEN_1     PC4
#define FIFO_WRST_1    PC5
#define FIFO_nOE_0     PC6
#define FIFO_nOE_1     PC7
//////////////////////////////////////////////////////////////////////////
//  Port D
//////////////////////////////////////////////////////////////////////////
#define USART0_RX    PD0
#define USART0_TX    PD1
#define OV7670_VSYNC_0   PD2 //MUST BE AN INTERRUPT PIN
#define OV7670_VSYNC_1   PD3 //MUST BE AN INTERRUPT PIN
#define FIFO_RCLK_0    PD4
#define FIFO_nRRST_0   PD5
#define FIFO_WEN_0     PD6
#define FIFO_WRST_0    PD7



#endif /* CONFIG_H_ */
```

C.1.1.5 Config.c

../Code/DualOV7670/Config.c

```
/*
 * Config.c
 *
 * Contains Global Methods and initialisations
 *
 * Created: 25/10/2012 21:59:06
 *  Author: hslovett
 */

#include "Config.h"
#include <avr/io.h>
void IO_Init(void)
{
  //initialise timer 0 to interrupt every 10 ms
  TIMSK0 |= (1 << OCIE0A);
  TCCR0A |= (1 << WGM01);
  OCR0A = 117; //10ms interrupt at 12MHz
  TCCR0B |= (1 << CS02) | (1 << CS00);


  DDRA = 0x00;
  //PORTB = 0xBF;
  DDRC = 0xFC;
  DDRD = 0xF2;
```

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

C.1.1.6 DualCameras.h

../Code/DualOV7670/DualCameras.h

```
/*
 * DualCameras.h
 *
 * Created: 10/11/2012 15:19:52
 *  Author: hslovett
 */


#ifndef DUALCAMERAS_H_
#define DUALCAMERAS_H_

#include "Config.h"

////////////////////////////////////////////////////////////////////////
//  Constants
////////////////////////////////////////////////////////////////////////
#define HEIGHT        240
#define WIDTH       320
#define PIXELSIZE     2
#define SETTINGS_LENGTH   167
#define OV7670_ADDR     0x21
////////////////////////////////////////////////////////////////////////
//  Globals
////////////////////////////////////////////////////////////////////////
const char default_settings[SETTINGS_LENGTH][2];
volatile uint8_t VSYNC_0_Count;
volatile uint8_t VSYNC_1_Count;
////////////////////////////////////////////////////////////////////////
//  Methods
////////////////////////////////////////////////////////////////////////
unsigned char OV7670_init(void);              //Initialises Camera
void FIFO_init(void);          //Initialises Buffer
uint8_t GetImageIfAvailiable(FIL *File, uint8_t CameraID);
void LoadImagesToBuffer(void);
unsigned char rdOV7670Reg(unsigned char regID, unsigned char *regDat);
unsigned char OV7670_SCCB_init(void);
void FIFO_Reset(uint8_t CameraID);
////////////////////////////////////////////////////////////////////////
//  Pins & Macros
////////////////////////////////////////////////////////////////////////
#define FIFO_RCLK_1   PC2
#define FIFO_nRRST_1  PC3
#define FIFO_WEN_1    PC4
```

```
#define FIFO_WRST_1    PC5
#define FIFO_nOE_0     PC6
#define FIFO_nOE_1     PC7

#define FIFO_RCLK_1_SET   { PORTC |= (1 <<  FIFO_RCLK_1); }
#define FIFO_RCLK_1_CLR   { PORTC &= ~(1 << FIFO_RCLK_1); }
#define FIFO_nRRST_1_SET  { PORTC |= (1 <<  FIFO_nRRST_1);  }
#define FIFO_nRRST_1_CLR  { PORTC &= ~(1 << FIFO_nRRST_1);  }
#define FIFO_WEN_1_SET    { PORTC |= (1 <<  FIFO_WEN_1);  }
#define FIFO_WEN_1_CLR    { PORTC &= ~(1 << FIFO_WEN_1);  }
#define FIFO_WRST_1_SET   { PORTC |= (1 <<  FIFO_WRST_1); }
#define FIFO_WRST_1_CLR   { PORTC &= ~(1 << FIFO_WRST_1); }
#define FIFO_nOE_0_SET    { PORTC |= (1 << FIFO_nOE_0);   }
#define FIFO_nOE_0_CLR    { PORTC &= ~(1 << FIFO_nOE_0);  }
#define FIFO_nOE_1_SET    { PORTC |= (1 << FIFO_nOE_1);   }
#define FIFO_nOE_1_CLR    { PORTC &= ~(1 << FIFO_nOE_1);  }


#define FIFO_RCLK_0    PD4
#define FIFO_nRRST_0   PD5
#define FIFO_WEN_0     PD6
#define FIFO_WRST_0    PD7

#define FIFO_RCLK_0_SET   { PORTD |= (1 <<  FIFO_RCLK_0); }
#define FIFO_RCLK_0_CLR   { PORTD &= ~(1 << FIFO_RCLK_0); }
#define FIFO_nRRST_0_SET  { PORTD |= (1 <<  FIFO_nRRST_0);  }
#define FIFO_nRRST_0_CLR  { PORTD &= ~(1 << FIFO_nRRST_0);  }
#define FIFO_WEN_0_SET    { PORTD |= (1 <<  FIFO_WEN_0);  }
#define FIFO_WEN_0_CLR    { PORTD &= ~(1 << FIFO_WEN_0);  }
#define FIFO_WRST_0_SET   { PORTD |= (1 <<  FIFO_WRST_0); }
#define FIFO_WRST_0_CLR   { PORTD &= ~(1 << FIFO_WRST_0); }

///////////////////////////////////////////////////////////////////////////
//Camera Register Address definitions
///////////////////////////////////////////////////////////////////////////
#define OV_GAIN     0x00  //Gain Control Setting - ACG[7:0]
#define OV_BLUE     0x01  //Blue Channel Gain
#define OV_RED      0x02  //Red Channel Gain
#define OV_VREF     0x03  //Vertical Frame Control & ACG[9:8]
#define OV_COM1     0x04  //CCIR656 enable, AEC low bits (AECHH, AECH)
#define OV_BAVE     0x05  //U/B Average level - AUTO UPDATED
#define OV_GbAVE    0x06  //Y/Gb Average Level - AUTO UPDATED
#define OV_AECHH    0x07  //Exposure value [15:10] (AECH, COM1)
#define OV_RAVE     0x08  //V/R Average level - AUTO UPDATED
#define OV_COM2     0x09  //Soft Sleep, Output drive capability
#define OV_PID      0x0A  //Product ID MSB  Read only
#define OV_VER      0x0B  //Product ID LSB  Read Only
#define OV_COM3     0x0C  //Output data MSB/LSB swap + other stuff
#define OV_COM4     0x0D  //Average values - MUST BE SAME AS COM17
#define OV_COM5     0x0E  //RESERVED
#define OV_COM6     0x0F  //COM6
#define OV_AECH     0x10  //Exposure value [9:2] (see AECHH, COM1)
#define OV_CLKRC    0x11  //Internal Clock options
#define OV_COM7     0x12  //RESET, Output format
#define OV_COM8     0x13  //Common control 8
#define OV_COM9     0x14  //Automatic Gain Ceiling
#define OV_COM10    0x15  //PCLK, HREF and VSYNC options
#define OV_RSVD     0x16  //RESERVED
#define OV_HSTART   0x17  //Output format Horizontal Frame start
```

```
#define OV_HSTOP     0x18  //Output format Horizontal Frame end
#define OV_VSTRT     0x19  //Output format Vertical Frame start
#define OV_VSTOP     0x1A  //Output format Vertical Frame Stop
#define OV_PSHFT     0x1B  //Pixel Delay Select
#define OV_MIDH      0x1C  //Manufacturer ID MSB -   READ ONLY
#define OV_MIDL      0x1D  //Manufacturer ID LSB -   READ ONLY
#define OV_MVFP      0x1E  //Mirror / VFlip Enable
#define OV_LAEC      0x1F  //RESERVED
#define OV_ADCCTR0     0x20  //ADC Control
#define OV_ADCCTR1     0x21  //RESERVED
#define OV_ADCCTR2     0x22  //RESERVED
#define OV_ADCCTR3     0x23  //RESERVED
#define OV_AEW       0x24  //ACG/AEC Stable Operating Region Upper Limit
#define OV_AEB       0x25  //ACG/AEC Stable Operation Region Lower Limit
#define OV_VPT       0x26  //ACG/AEC Fast Mode Operation Region
#define OV_BBIAS     0x27  //B Channel Signal Output Bias
#define OV_GbBIAS    0x28  //Gb Channel Output Bias
#define OV_RSVD1     0x29  //RESERVED
#define OV_EXHCH     0x2A  //Dummy Pixel Insert MSB
#define OV_EXHCL     0x2B  //Dummy Pixel Insert LSB
#define OV_RBIAS     0x2C  //R Channel Signal Output Bias
#define OV_ADVFL     0x2D  //LSB of insert dummy line in vertical direction
#define OV_ADVFH     0x2E  //MSB of insert dummy line in vertical direction
#define OV_YAVE      0x2F  //Y/G Channel Average Value
#define OV_HSYST     0x30  //HSYNC Rising Edge Delay (low 8 bits)
#define OV_HSYEN     0x31  //HSYNCE Falling Edge Delay (low 8 bits)
#define OV_HREF      0x32  //HREF Control
#define OV_CHLF      0x33  //Array Current Control - RESERVED
#define OV_ARBLM     0x34  //Array Reference Control - RESERVED
#define OV_RSVD2     0x35  //RESERVED
#define OV_RSVD3     0x36  //RESERVED
#define OV_ADCCTRL     0x37  //ADC Control - RESERVED
#define OV_ACOM      0x38  //ADC and Analog Common Mode Control - RESERVED
#define OV_OFON      0x39  //ADC Offset Control
#define OV_TSLB      0x3A  //Line Buffer Test Option
#define OV_COM11     0x3B  //COM11
#define OV_COM12     0x3C  //COM12
#define OV_COM13     0x3D  //COM13
#define OV_COM14     0x3E  //COM14
#define OV_EDGE      0x3F  //Edge Detection Adjustment
#define OV_COM15     0x40  //COM15
#define OV_COM16     0x41  //COM16
#define OV_COM17     0x42  //COM17
#define OV_AWBC1     0x43
#define OV_AWBC2     0x44
#define OV_AWBC3     0x45
#define OV_AWBC4     0x46
#define OV_AWBC5     0x47
#define OV_AWBC6     0x48
#define OV_RSVD4     0x49
#define OV_RSVD5     0x40
#define OV_RSVD6     0x4A
#define OV_REG4B     0x4B
#define OV_DNSTH     0x4C
#define OV_RSVD7     0x4D
#define OV_RSVD8     0x4E
#define OV_MTX1      0x4F
#define OV_MTX2      0x50
#define OV_MTX3      0x51
```

```
#define OV_MTX4      0x52
#define OV_MTX5      0x53
#define OV_MTX6      0x54
#define OV_BRIGHT    0x55
#define OV_CONTRAS     0x56
#define OV_CONTRASCNTR  0x57
#define OV_MTXS      0x58
#define OV_RSVD9     0x59
#define OV_RSVD9_1     0x5A
#define OV_RSVD9_2     0x5B
#define OV_RSVD9_3     0x5C
#define OV_RSVD9_4     0x5D
#define OV_RSVD9_5     0x5E
#define OV_RSVD9_6     0x5F
#define OV_RSVD10    0x60
#define OV_RSVD11    0x61
#define OV_LCC1      0x62
#define OV_LCC2      0x63
#define OV_LCC3      0x64
#define OV_LCC4      0x65
#define OV_LCC5      0x66
#define OV_MANU      0x67
#define OV_MANV      0x68
#define OV_GFIX      0x69
#define OV_GGAIN     0x6A
#define OV_DBLV      0x6B
#define OV_AWBCTR3     0x6C
#define OV_AWBCTR2     0x6D
#define OV_AWBCTR1     0x6E
#define OV_AWBCTR0     0x6F
#define OV_SCALING_XSC  0x70
#define OV_SCALING_YSC  0x71
#define OV_SCALING_DCWCTR 0x72
#define OV_SCALING_PCLK_DIV 0x73
#define OV_REG74     0x74
#define OV_REG75     0x75
#define OV_REG76     0x76
#define OV_REG77     0x77
#define OV_RSVD12    0x78
#define OV_RSVD13    0x79
#define OV_GAM1      0x7A
#define OV_GAM2      0x7B
#define OV_GAM3      0x7C
#define OV_GAM4      0x7D
#define OV_GAM5      0x7E
#define OV_GAM6      0x7F
#define OV_GAM7      0x80
#define OV_GAM8      0x81
#define OV_GAM9      0x82
#define OV_GAM10     0x83
#define OV_GAM11     0x84
#define OV_GAM12     0x85
#define OV_GAM13     0x86
#define OV_GAM14     0x87
#define OV_GAM15     0x88
#define OV_GAM16     0x89
#define OV_RSVD14    0x8A
#define OV_RSVD15    0x8B
#define OV_RSVD16    0x8C
```

```
#define OV_RSVD17    0x8D
#define OV_RSVD18    0x8E
#define OV_RSVD19    0x8F
#define OV_RSVD20    0x90
#define OV_RSVD21    0x91
#define OV_DM_LNL    0x92
#define OV_DM_LNH    0x93
#define OV_LCC6      0x94
#define OV_LCC7      0x95
#define OV_RSVD22    0x96
#define OV_RSVD23    0x97
#define OV_RSVD24    0x98
#define OV_RSVD25    0x99
#define OV_RSVD26    0x9A
#define OV_RSVD27    0x9B
#define OV_RSVD28    0x9C
#define OV_BD50ST    0x9D
#define OV_BD60ST    0x9E
#define OV_HIST0     0x9F
#define OV_HIST1     0xA0
#define OV_HIST2     0xA1
#define OV_HIST3     0xA2
#define OV_HIST4     0xA3
#define OV_HIST5     0xA4
#define OV_HIST6     0xA5
#define OV_HIST7     0xA6
#define OV_HIST8     0xA7
#define OV_HIST9     0xA8
#define OV_HIST10    0xA9
#define OV_HIST11    0xAA
#define OV_HIST12    0xAB
#define OV_STR_OPT     0xAC
#define OV_STR_R     0xAD
#define OV_STR_G     0xAE
#define OV_STR_B     0xAF
#define OV_RSVD28_1    0xB0
#define OV_RSVD29    0xB1
#define OV_RSVD30    0xB2
#define OV_THL_ST    0xB3
#define OV_RSVD31    0xB4
#define OV_THL_DLT     0xB5
#define OV_RSVD32    0xB6
#define OV_RSVD33    0xB7
#define OV_RSVD34    0xB8
#define OV_RSVD35    0xB9
#define OV_RSVD36    0xBA
#define OV_RSVD37    0xBB
#define OV_RSVD38    0xBC
#define OV_RSVD39    0xBD
#define OV_AD_CHB    0xBE
#define OV_AD_CHR    0xBF
#define OV_AD_CHGb     0xC0
#define OV_AD_CHGr     0xC1
#define OV_RSVD40    0xC2
#define OV_RSVD41    0xC3
#define OV_RSVD42    0xC4
#define OV_RSVD43    0xC5
#define OV_RSVD44    0xC6
#define OV_RSVD45    0xC7
```

```
#define OV_RSVD46   0xC8
#define OV_SATCTR   0xC9

#endif /* DUALCAMERAS_H_ */
```

C.1.1.7 DualCameras.c

../Code/DualOV7670/DualCameras.c

```
/*
 * DualCameras.c
 *
 * Created: 10/11/2012 15:20:03
 *  Author: hl13g10
 */

#include "DualCameras.h"

const char default_settings[SETTINGS_LENGTH][2]=
{
{OV_TSLB, 0x04},
{OV_COM15, 0xd0},//RGB565 / RGB555
{OV_COM7, 0x14},
{OV_HREF, 0x80},
{OV_HSTART, 0x16},
{OV_HSTOP, 0x04},
{OV_VSTRT, 0x02},
{OV_VSTOP, 0x7b},//0x7a,
{OV_VREF, 0x06},//0x0a,
{OV_COM3, 0x00},
{OV_COM14, 0x00},//
{OV_SCALING_XSC, 0x00},
{OV_SCALING_YSC, 0x00},
{OV_SCALING_DCWCTR, 0x11},
{OV_SCALING_PCLK_DIV, 0x00},//
{0xa2, 0x02},
{OV_CLKRC, 0x01},
{OV_GAM1, 0x20},
{OV_GAM2, 0x1c},
{OV_GAM3, 0x28},
{OV_GAM4, 0x3c},
{OV_GAM5, 0x55},
{OV_GAM6, 0x68},
{OV_GAM7, 0x76},
{OV_GAM8, 0x80},
{OV_GAM9, 0x88},
{OV_GAM10, 0x8f},
{OV_GAM11, 0x96},
{OV_GAM12, 0xa3},
{OV_GAM13, 0xaf},
{OV_GAM14, 0xc4},
{OV_GAM15, 0xd7},
{OV_GAM16, 0xe8},
{OV_COM8, 0xe0},
{OV_GAIN, 0x00},//AGC
{OV_AECH, 0x00},
```

```
{OV_COM4 , 0x00},
{OV_COM9 , 0x20},//0x38, limit the max gain
{OV_HIST6 , 0x05},
{OV_HIST12 , 0x07},
{OV_AEW , 0x75},
{OV_AEB , 0x63},
{OV_VPT , 0xA5},
{OV_HIST0 , 0x78},
{OV_HIST1 , 0x68},
{OV_HIST2 , 0x03},//0x0b,
{OV_HIST7 , 0xdf},//0xd8,
{OV_HIST8 , 0xdf},//0xd8,
{OV_HIST9 , 0xf0},
{OV_HIST10 , 0x90},
{OV_HIST11 , 0x94},
{OV_COM8 , 0xe5},
{OV_COM5 , 0x61},
{OV_COM6 , 0x4b},
{0x16 , 0x02},
{OV_MVFP , 0x27},//0x37,
{0x21 , 0x02},
{0x22 , 0x91},
{0x29 , 0x07},
{0x33 , 0x0b},
{0x35 , 0x0b},
{0x37 , 0x1d},
{0x38 , 0x71},
{OV_OFON , 0x2a},//
{OV_COM12 , 0x78},
{0x4d , 0x40},
{0x4e , 0x20},
{OV_GFIX , 0x0c},///////////////////////
{OV_DBLV , 0x60},//PLL
{OV_REG74 , 0x19},
{0x8d , 0x4f},
{0x8e , 0x00},
{0x8f , 0x00},
{0x90 , 0x00},
{0x91 , 0x00},
{OV_DM_LNL , 0x00},//0x19,//0x66
{0x96 , 0x00},
{0x9a , 0x80},
{0xb0 , 0x84},
{0xb1 , 0x0c},
{0xb2 , 0x0e},
{OV_THL_ST , 0x82},
{0xb8 , 0x0a},
{OV_AWBC1 , 0x14},
{OV_AWBC2 , 0xf0},
{OV_AWBC3 , 0x34},
{OV_AWBC4 , 0x58},
{OV_AWBC5 , 0x28},
{OV_AWBC6 , 0x3a},
{0x59 , 0x88},
{0x5a , 0x88},
{0x5b , 0x44},
{0x5c , 0x67},
{0x5d , 0x49},
{0x5e , 0x0e},
```

```
{OV_LCC3, 0x04},
{OV_LCC4, 0x20},
{OV_LCC5, 0x05},
{OV_LCC6, 0x04},
{OV_LCC7, 0x08},
{OV_AWBCTR3, 0x0a},
{OV_AWBCTR2, 0x55},
{OV_AWBCTR1, 0x11},
{OV_AWBCTR0, 0x9f},//0x9e for advance AWB
{OV_GGAIN, 0x40},
{OV_BLUE, 0x40},
{OV_RED, 0x40},
{OV_COM8, 0xe7},
{OV_COM10, 0x02},//VSYNC negative
{OV_MTX1, 0x80},
{OV_MTX2, 0x80},
{OV_MTX3, 0x00},
{OV_MTX4, 0x22},
{OV_MTX5, 0x5e},
{OV_MTX6, 0x80},
{OV_MTXS, 0x9e},
{OV_COM16, 0x08},
{OV_EDGE, 0x00},
{OV_REG75, 0x05},
{OV_REG76, 0xe1},
{OV_DNSTH, 0x00},
{OV_REG77, 0x01},
{OV_COM13, 0xc2}, //0xc0,
{OV_REG4B, 0x09},
{OV_SATCTR, 0x60},
{OV_COM16, 0x38},
{OV_CONTRAS, 0x40},
{0x34, 0x11},
{OV_COM11, 0x02},//0x00,//0x02,
{OV_HIST5, 0x89},//0x88,
{0x96, 0x00},
{0x97, 0x30},
{0x98, 0x20},
{0x99, 0x30},
{0x9a, 0x84},
{0x9b, 0x29},
{0x9c, 0x03},
{OV_BD50ST, 0x4c},
{OV_BD60ST, 0x3f},
{0x78, 0x04},
{0x79, 0x01},//Some weird thing with reserved registers.
{0xc8, 0xf0},
{0x79, 0x0f},
{0xc8, 0x00},
{0x79, 0x10},
{0xc8, 0x7e},
{0x79, 0x0a},
{0xc8, 0x80},
{0x79, 0x0b},
{0xc8, 0x01},
{0x79, 0x0c},
{0xc8, 0x0f},
{0x79, 0x0d},
{0xc8, 0x20},
```

```
{0x79, 0x09},
{0xc8, 0x80},
{0x79, 0x02},
{0xc8, 0xc0},
{0x79, 0x03},
{0xc8, 0x40},
{0x79, 0x05},
{0xc8, 0x30},
{0x79, 0x26},
{OV_COM2, 0x03},
{OV_BRIGHT, 0x00},
{OV_CONTRAS, 0x40},
{OV_COM11, 0x42},//0x82,//0xc0,//0xc2,  //night mode

};



//ISR for controlling WEN.
ISR(INT0_vect)
{
  //printf("ISR INT0 Entered\n");
  if (VSYNC_0_Count==1)//start a frame read
  {
    FIFO_WEN_0_SET;
    VSYNC_0_Count++;
  }
  else if (VSYNC_0_Count==2)//end a frame read
  {
    FIFO_WEN_0_CLR;
    VSYNC_0_Count++;
  }
  else if(VSYNC_0_Count == 3)
  {
    FIFO_WEN_0_CLR;
  }
  else
  {
    FIFO_WEN_0_CLR
    VSYNC_0_Count = 0;//wait for a read to be started
  }
}
//ISR for controlling WEN.
ISR(INT1_vect)
{
  //printf("ISR INT1 Entered\n");
  if (VSYNC_1_Count==1)//start a frame read
  {
    FIFO_WEN_1_SET;
    VSYNC_1_Count++;
  }
  else if (VSYNC_1_Count==2)//end a frame read
  {
    FIFO_WEN_1_CLR;
    VSYNC_1_Count++;
  }
  else if(VSYNC_1_Count == 3)
  {
    FIFO_WEN_1_CLR;
```

```
  }
  else
  {
    FIFO_WEN_1_CLR
    VSYNC_1_Count = 0;//wait for a read to be started
  }
}

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

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

  return TWI_statusReg.lastTransOK;
}

//Read Register Method
unsigned char rdOV7670Reg(unsigned char regID, unsigned char *regDat)
{
  /*  I2C Traffic Generated:
   *  S | OV_ADDR + W | A | RegID | A | P |
   *  S | OV_ADDR + R | A | Data  |~A | P |
   */
  //I2C Interface
  unsigned char messageBuf[TWI_BUFFER_SIZE]; //Initialise a buffer
  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.
  messageBuf[1] = regID;             // The first byte is used for Address
    Pointer.
  TWI_Start_Transceiver_With_Data( messageBuf, 2 );

  // Request/collect the data from the Slave
  messageBuf[0] = (OV7670_ADDR<<TWI_ADR_BITS) | (TRUE<<TWI_READ_BIT); // The
    first byte must always consist of General Call code or the TWI slave
    address.
  TWI_Start_Transceiver_With_Data( messageBuf, 2 );

  // Get the received data from the transceiver buffer
  TWI_Get_Data_From_Transceiver( messageBuf, 2 );
  *regDat = messageBuf[1];
  return TWI_statusReg.lastTransOK;
}


unsigned char OV7670_init()
```

```
{
  uint8_t i = 0;
  if(0==wrOV7670Reg(OV_COM7, 0x80)) //Reset Camera
  {
    return 1;
  }
  _delay_ms(10);
  for(i=0; i<SETTINGS_LENGTH; i++)
  {
    if( 0==wrOV7670Reg(default_settings[i][0], default_settings[i][1] ))
    {
      return 1;
    }
    _delay_ms(1);
  }

  return 0;
}

void FIFO_init( void )
{
  //disable both outputs
  FIFO_nOE_0_SET;
  FIFO_nOE_1_SET;
  //Reset Buffer 0
  FIFO_WRST_0_CLR;
  FIFO_RCLK_0_CLR;
  //FIFO_nOE_0_CLR;
  FIFO_nRRST_0_SET;
  FIFO_WEN_0_CLR;
  _delay_us(10);
  FIFO_RCLK_0_SET;
  _delay_us(10);
  FIFO_RCLK_0_CLR;
  FIFO_nRRST_0_CLR;
  _delay_us(10);
  FIFO_RCLK_0_SET;
  _delay_us(10);
  FIFO_RCLK_0_CLR;
  FIFO_nRRST_0_SET;
  _delay_us(10);
  FIFO_WRST_0_SET;

  //Reset Buffer 1
  FIFO_WRST_1_CLR;
  FIFO_RCLK_1_CLR;
  //FIFO_nOE_1_CLR;
  FIFO_nRRST_1_SET;
  FIFO_WEN_1_CLR;
  _delay_us(10);
  FIFO_RCLK_1_SET;
  _delay_us(10);
  FIFO_RCLK_1_CLR;
  FIFO_nRRST_1_CLR;
  _delay_us(10);
  FIFO_RCLK_1_SET;
  _delay_us(10);
  FIFO_RCLK_1_CLR;
  FIFO_nRRST_1_SET;
```

```
    _delay_us(10);
    FIFO_WRST_1_SET;

  }

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

    DDRA = 0;
    if(ID == 1)
    {
      FIFO_RCLK_1_SET;
      data = PINA;
      FIFO_RCLK_1_CLR;
      data <<= 8;
      FIFO_RCLK_1_SET;
      data |= PINA;
      FIFO_RCLK_1_CLR;
    }
    else
    {
      FIFO_RCLK_0_SET;
      data = PINA;
      FIFO_RCLK_0_CLR;
      data <<= 8;
      FIFO_RCLK_0_SET;
      data |= PINA;
      FIFO_RCLK_0_CLR;
    }
    return(data);
  }


  //Resets both pointers
  void FIFO_Reset(uint8_t CameraID)
  {
    FIFO_nOE_0_SET;
    FIFO_nOE_1_SET;
    if(CameraID == 0)
    {
      FIFO_WRST_0_CLR;
      FIFO_nRRST_0_CLR;
      FIFO_RCLK_0_SET;
      FIFO_RCLK_0_CLR;
      FIFO_nRRST_0_SET;
      FIFO_WRST_0_SET;
    }
    else
    {
      FIFO_WRST_1_CLR;
      FIFO_nRRST_1_CLR;
      FIFO_RCLK_1_SET;
      FIFO_RCLK_1_CLR;
      FIFO_nRRST_1_SET;
      FIFO_WRST_1_SET;
    }
```

```
}
void LoadImagesToBuffer()
{
  VSYNC_0_Count = 0;
  VSYNC_1_Count = 0;
  FIFO_Reset(0);
  FIFO_Reset(1);
  VSYNC_0_Count = 1;
  VSYNC_1_Count = 1;

}
uint8_t GetImageIfAvailiable(FIL *File, uint8_t CameraID)
{

  if( ((CameraID == 0) && (VSYNC_0_Count == 3)) ||
    ((CameraID == 1) && (VSYNC_1_Count == 3)) )
  {

    //Write Bitmap Headers
    WriteBMPHeader(File);
    WriteDIBHeader(File);
    if (CameraID == 0)
    {
      //Enable output of Camera 0
      FIFO_nOE_0_CLR;
      //Reset Read Pointer
      FIFO_nRRST_0_CLR;
      FIFO_RCLK_0_SET;
      FIFO_RCLK_0_CLR;
      FIFO_nRRST_0_SET;
    }
    else
    {
      //Enable output of Camera 0
      FIFO_nOE_1_CLR;
      //Reset Read Pointer
      FIFO_nRRST_1_CLR;
      FIFO_RCLK_1_SET;
      FIFO_RCLK_1_CLR;
      FIFO_nRRST_1_SET;
    }
    int i, j;
    uint32_t pointer;
    uint16_t Temp;
    uint32_t p;
    FRESULT fr;
    //for(j = HEIGHT; j>0; j--)
    for(j = 0; j < HEIGHT; j++)
    {
      pointer = 0;
      for(i = 0; i < WIDTH; i++)
      {
        Temp = FIFO_TO_AVR(CameraID);
        //USART0_Senduint16(Temp);

        Buff[pointer++] = (uint8_t)(Temp >> 8);
        Buff[pointer++] = (uint8_t)Temp;
      }
```

```
      pointer = (uint32_t)j * (uint32_t)WIDTH * 2 + BMPHEADERSIZE +
    DIBHEADERSIZE;
      f_lseek(File, pointer);
      fr = f_write(File, Buff, WIDTH * 2, &p);
      if(fr != FR_OK)
      {
        //printf("Write Fail.\n");
        VSYNC_0_Count = 0;
        VSYNC_1_Count = 0;
        FIFO_Reset(CameraID);
        FIFO_nOE_0_SET;
        FIFO_nOE_1_SET;
        return 1;
      }
    }
    FIFO_Reset(CameraID);
    //fr = f_close(File);
    FIFO_nOE_0_SET;
    FIFO_nOE_1_SET;
    return 0;
  }
  else
  {
    return 2;
  }
}
```

C.1.1.8 PCA9542A.h

../Code/DualOV7670/PCA9542A.h

```
/*
 * PCA9542A.h
 *
 * Created: 13/11/2012 23:24:48
 *  Author: hslovett
 */


#ifndef PCA9542A_H_
#define PCA9542A_H_
#include "Config.h"

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

#define NO_SELECT 0x00
#define CH0     0x04
#define CH1     0x05

unsigned char PCA9542A_Init();
unsigned char PCA9542A_SetChannel(uint8_t Channel);

#endif /* PCA9542A_H_ */
```

C.1.1.9 PCA9542A.c

../Code/DualOV7670/PCA9542A.c

```
/*
 * PCA9542A.c
 *
 * Created: 13/11/2012 23:24:40
 *  Author: hslovett
 */
#include "PCA9542A.h"


unsigned char PCA9542A_Init()
{
  unsigned char messageBuf[TWI_BUFFER_SIZE];
  messageBuf[0] = (PCA9542A_ADDR  <<TWI_ADR_BITS) | (FALSE<<TWI_READ_BIT); // The first byte must always consit of General Call code or the TWI slave address.
  messageBuf[1] = NO_SELECT;             // The first byte is used for commands.
  // The second byte is used for the data.
  TWI_Start_Transceiver_With_Data( messageBuf, 2 );

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

  return TWI_statusReg.lastTransOK;
}

unsigned char PCA9542A_SetChannel( uint8_t Channel )
{
  unsigned char messageBuf[TWI_BUFFER_SIZE];
  messageBuf[0] = (PCA9542A_ADDR  <<TWI_ADR_BITS) | (FALSE<<TWI_READ_BIT); // The first byte must always consit of General Call code or the TWI slave address.
  messageBuf[1] = Channel;             // The first byte is used for commands.
                        // The second byte is used for the data.
  TWI_Start_Transceiver_With_Data( messageBuf, 2 );

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

  return TWI_statusReg.lastTransOK;
}

unsigned char PCA9542A_ReadChannel()
{
  unsigned char messageBuf[TWI_BUFFER_SIZE];
  messageBuf[0] = (PCA9542A_ADDR  <<TWI_ADR_BITS) | (TRUE<<TWI_READ_BIT); // The first byte must always consit of General Call code or the TWI slave address.

  TWI_Start_Transceiver_With_Data( messageBuf, 1 );

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

```
  // Get the received data from the transceiver buffer
  TWI_Get_Data_From_Transceiver( messageBuf, 2 );
  return TWI_statusReg.lastTransOK;
}
```

C.1.1.10 TWI_Master.h

../Code/DualOV7670/TWI_Master.h

```
/*****************************************************************************
*
* Atmel Corporation
*
* File              : TWI_Master.h
* Compiler          : IAR EWAAVR 2.28a/3.10c
* Revision          : Revision: 1.13
* Date              : Date: 24. mai 2004 11:31:22
* Updated by        : Author: ltwa
*
* Support mail      : avr@atmel.com
*
* Supported devices : All devices with a TWI module can be used.
*                     The example is written for the ATmega16
*
* AppNote           : AVR315 - TWI Master Implementation
*
* Description       : Header file for TWI_Master.c
*                     Include this file in the application.
*
****************************************************************************/

/*  Modified by Henry Lovett (hl13g10@ecs.soton.ac.uk) to allow SCL frequency
    to be specified and TWBR calculated
 *  Also allows AVR internal pull up resistors to be used.
 */
#ifndef _TWI_MASTER_H
#define _TWI_MASTER_H
#include <avr/io.h>
#include <avr/interrupt.h>
#include "Config.h"
/****************************************************************************
  TWI Status/Control register definitions
****************************************************************************/

#define INTERNAL_PULLUPS 0

#define TWI_BUFFER_SIZE 4   // Set this to the largest message size that will
    be sent including address byte.

#define SCL_Freq    100000    //SCL Frequency in Hertz
#define TWI_TWBR    (char)(F_CPU / 2 / SCL_Freq - 8) //Equation to calculate
    TWBR Based on SCL Frequency and Clock Frequency

//#define TWI_TWBR            0x0C //400KHz       // TWI Bit rate Register
    setting.
```

```
//#define TWI_TWBR        0x34 //100KHz                                    //
     Se Application note for detailed
                                        // information on setting this value.
// Not used defines!
//#define TWI_TWPS          0x00        // This driver presumes prescaler = 00

/****************************************************************************
  Global definitions
****************************************************************************/

union TWI_statusReg                       // Status byte holding flags.
{
    unsigned char all;
    struct
    {
        unsigned char lastTransOK:1;
        unsigned char unusedBits:7;
    };
};

extern union TWI_statusReg TWI_statusReg;

/****************************************************************************
  Function definitions
****************************************************************************/
void TWI_Master_Initialise( void );
unsigned char TWI_Transceiver_Busy( void );
unsigned char TWI_Get_State_Info( void );
void TWI_Start_Transceiver_With_Data( unsigned char * , unsigned char );
void TWI_Start_Transceiver( void );
unsigned char TWI_Get_Data_From_Transceiver( unsigned char *, unsigned char );

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

#define TRUE          1
#define FALSE         0

/****************************************************************************
  TWI State codes
****************************************************************************/
// General TWI Master staus codes
#define TWI_START                  0x08  // START has been transmitted
#define TWI_REP_START              0x10  // Repeated START has been
    transmitted
#define TWI_ARB_LOST               0x38  // Arbitration lost

// TWI Master Transmitter staus codes
#define TWI_MTX_ADR_ACK            0x18  // SLA+W has been tramsmitted and ACK
     received
#define TWI_MTX_ADR_NACK           0x20  // SLA+W has been tramsmitted and
    NACK received
#define TWI_MTX_DATA_ACK           0x28  // Data byte has been tramsmitted and
     ACK received
```

```
#define TWI_MTX_DATA_NACK          0x30  // Data byte has been tramsmitted and NACK received

// TWI Master Receiver staus codes
#define TWI_MRX_ADR_ACK            0x40  // SLA+R has been tramsmitted and ACK received
#define TWI_MRX_ADR_NACK           0x48  // SLA+R has been tramsmitted and NACK received
#define TWI_MRX_DATA_ACK           0x50  // Data byte has been received and ACK tramsmitted
#define TWI_MRX_DATA_NACK          0x58  // Data byte has been received and NACK tramsmitted

// TWI Slave Transmitter staus codes
#define TWI_STX_ADR_ACK            0xA8  // Own SLA+R has been received; ACK has been returned
#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
#define TWI_STX_DATA_ACK           0xB8  // Data byte in TWDR has been transmitted; ACK has been received
#define TWI_STX_DATA_NACK          0xC0  // Data byte in TWDR has been transmitted; NOT ACK has been received
#define TWI_STX_DATA_ACK_LAST_BYTE 0xC8  // Last data byte in TWDR has been transmitted (TWEA = 0 ); ACK has been received

// TWI Slave Receiver staus codes
#define TWI_SRX_ADR_ACK            0x60  // Own SLA+W has been received ACK has been returned
#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
#define TWI_SRX_GEN_ACK            0x70  // General call address has been received; ACK has been returned
#define TWI_SRX_GEN_ACK_M_ARB_LOST 0x78  // Arbitration lost in SLA+R/W as Master; General call address has been received; ACK has been returned
#define TWI_SRX_ADR_DATA_ACK       0x80  // Previously addressed with own SLA+W; data has been received; ACK has been returned
#define TWI_SRX_ADR_DATA_NACK      0x88  // Previously addressed with own SLA+W; data has been received; NOT ACK has been returned
#define TWI_SRX_GEN_DATA_ACK       0x90  // Previously addressed with general call; data has been received; ACK has been returned
#define TWI_SRX_GEN_DATA_NACK      0x98  // Previously addressed with general call; data has been received; NOT ACK has been returned
#define TWI_SRX_STOP_RESTART       0xA0  // A STOP condition or repeated START condition has been received while still addressed as Slave

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

#endif
```

C.1.1.11 TWI_Master.c

../Code/DualOV7670/TWI_Master.c

```c
/*****************************************************************************
*
* Atmel Corporation
*
* File              : TWI_Master.c
* Compiler          : IAR EWAAVR 2.28a/3.10c
* Revision          : Revision: 1.13
* Date              : Date: 24. mai 2004 11:31:20
* Updated by        : Author: ltwa
*
* Support mail      : avr@atmel.com
*
* Supported devices : All devices with a TWI module can be used.
*                     The example is written for the ATmega16
*
* AppNote           : AVR315 - TWI Master Implementation
*
* Description       : This is a sample driver for the TWI hardware modules.
*                     It is interrupt driveren. All functionality is controlled through
*                     passing information to and from functions. Se main.c for samples
*                     of how to use the driver.
*
*
****************************************************************************/


#include "TWI_Master.h"

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

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

/****************************************************************************
Call this function to set up the TWI master to its initial standby state.
Remember to enable interrupts from the main application after initializing the TWI.
****************************************************************************/
void TWI_Master_Initialise(void)
{
  #if INTERNAL_PULLUPS == 1//enable built in pullups for I2C Lines
    DDRC = 0x00;
    PORTC = (1 << PC0) | (1 << PC1);
  #else
    #pragma message("External I2C Pull Ups Required.")
  #endif
  TWBR = TWI_TWBR;                                  // Set bit rate register (Baudrate). Defined in header file.
  // TWSR = TWI_TWPS;                                  // Not used. Driver presumes prescaler to be 00.
  TWDR = 0xFF;                                      // Default content = SDA released.
```

```
  TWCR =  (1<<TWEN)|                              // Enable TWI-interface and release TWI pins.
         (0<<TWIE)|(0<<TWINT)|                    // Disable Interupt.
         (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|         // No Signal requests.
         (0<<TWWC);                               //
}

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

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

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

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

  TWI_msgSize = msgSize;                        // Number of data to transmit.
  TWI_buf[0]  = msg[0];                         // Store slave address with R/W setting.
  if (!( msg[0] & (TRUE<<TWI_READ_BIT) ))       // If it is a write operation, then also copy data.
  {
    for ( temp = 1; temp < msgSize; temp++ )
      TWI_buf[ temp ] = msg[ temp ];
  }
  TWI_statusReg.all = 0;
```

```
  TWI_state         = TWI_NO_STATE ;
  TWCR = (1<<TWEN)|                              // TWI Interface enabled.
         (1<<TWIE)|(1<<TWINT)|                   // Enable TWI Interupt and clear the flag.
         (0<<TWEA)|(1<<TWSTA)|(0<<TWSTO)|        // Initiate a START condition.
         (0<<TWWC);                              //
}

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

/****************************************************************************
Call this function to read out the requested data from the TWI transceiver buffer. I.e. first call
TWI_Start_Transceiver to send a request for data to the slave. Then Run this function to collect the
data when they have arrived. Include a pointer to where to place the data and the number of bytes
requested (including the address field) in the function call. The function will hold execution (loop)
until the TWI_ISR has completed with the previous operation, before reading out the data and returning.
If there was an error in the previous transmission the function will return the TWI error code.
****************************************************************************/
unsigned char TWI_Get_Data_From_Transceiver( unsigned char *msg, unsigned char msgSize )
{
  unsigned char i;

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

  if( TWI_statusReg.lastTransOK )                // Last transmission competed successfully.
  {
    for ( i=0; i<msgSize; i++ )                  // Copy data from Transceiver buffer.
    {
      msg[ i ] = TWI_buf[ i ];
    }
  }
```

```
  return( TWI_statusReg.lastTransOK );
}

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

ISR(TWI_vect)
{
  static unsigned char TWI_bufPtr;

  switch (TWSR)
  {
    case TWI_START:             // START has been transmitted
    case TWI_REP_START:         // Repeated START has been transmitted
      TWI_bufPtr = 0;                                     // Set buffer pointer to the TWI Address location
    case TWI_MTX_ADR_ACK:       // SLA+W has been tramsmitted and ACK received
    case TWI_MTX_DATA_ACK:      // Data byte has been tramsmitted and ACK received
      if (TWI_bufPtr < TWI_msgSize)
      {
        TWDR = TWI_buf[TWI_bufPtr++];
        TWCR = (1<<TWEN)|                                 // TWI Interface enabled
               (1<<TWIE)|(1<<TWINT)|                      // Enable TWI Interupt and clear the flag to send byte
               (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|           //
               (0<<TWWC);                                 //
      }else                     // Send STOP after last byte
      {
        TWI_statusReg.lastTransOK = TRUE;                 // Set status bits to completed successfully.
        TWCR = (1<<TWEN)|                                 // TWI Interface enabled
               (0<<TWIE)|(1<<TWINT)|                      // Disable TWI Interrupt and clear the flag
               (0<<TWEA)|(0<<TWSTA)|(1<<TWSTO)|           // Initiate a STOP condition.
               (0<<TWWC);                                 //
      }
      break;
    case TWI_MRX_DATA_ACK:      // Data byte has been received and ACK tramsmitted
      TWI_buf[TWI_bufPtr++] = TWDR;
    case TWI_MRX_ADR_ACK:       // SLA+R has been tramsmitted and ACK received
      if (TWI_bufPtr < (TWI_msgSize-1) )                  // Detect the last byte to NACK it.
      {
        TWCR = (1<<TWEN)|                                 // TWI Interface enabled
               (1<<TWIE)|(1<<TWINT)|                      // Enable TWI Interupt and clear the flag to read next byte
```

```
                (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|           // Send ACK after reception
                (0<<TWWC);                                  //
        }else                   // Send NACK after next reception
        {
          TWCR = (1<<TWEN)|                                 // TWI Interface enabled
                (1<<TWIE)|(1<<TWINT)|                       // Enable TWI Interupt and clear the flag to read next byte
                (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|           // Send NACK after reception
                (0<<TWWC);                                  //
        }
        break;
      case TWI_MRX_DATA_NACK:     // Data byte has been received and NACK tramsmitted
        TWI_buf[TWI_bufPtr] = TWDR;
        TWI_statusReg.lastTransOK = TRUE;                   // Set status bits to completed successfully.
        TWCR = (1<<TWEN)|                                   // TWI Interface enabled
              (0<<TWIE)|(1<<TWINT)|                         // Disable TWI Interrupt and clear the flag
              (0<<TWEA)|(0<<TWSTA)|(1<<TWSTO)|             // Initiate a STOP condition.
              (0<<TWWC);                                    //
        break;
      case TWI_ARB_LOST:          // Arbitration lost
        TWCR = (1<<TWEN)|                                   // TWI Interface enabled
              (1<<TWIE)|(1<<TWINT)|                         // Enable TWI Interupt and clear the flag
              (0<<TWEA)|(1<<TWSTA)|(0<<TWSTO)|             // Initiate a (RE)START condition.
              (0<<TWWC);                                    //
        break;
      case TWI_MTX_ADR_NACK:      // SLA+W has been tramsmitted and NACK received
      case TWI_MRX_ADR_NACK:      // SLA+R has been tramsmitted and NACK received
      case TWI_MTX_DATA_NACK:     // Data byte has been tramsmitted and NACK received
//    case TWI_NO_STATE              // No relevant state information available; TWINT =  0
      case TWI_BUS_ERROR:         // Bus error due to an illegal START or STOP condition
      default:
        TWI_state = TWSR;                                   // Store TWSR and automatically sets clears noErrors bit.
                                                            // Reset TWI Interface
        TWCR = (1<<TWEN)|                                   // Enable TWI-interface and release TWI pins
              (0<<TWIE)|(0<<TWINT)|                         // Disable Interupt
              (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|             // No Signal requests
              (0<<TWWC);                                    //
    }
}
```

C.1.1.12 Usart.h

../Code/DualOV7670/Usart.h

```
/*
 * Usart.h
 *
 * Created: 25/10/2012 22:25:14
 *  Author: hslovett
 */


#ifndef USART_H_
#define USART_H_

#include "Config.h"
#include <stdio.h>
#include <avr/io.h>
#define USART0_BITRATE 57600
#define UBBR F_CPU/16/USART0_BITRATE-1

void USART0_Init ();
void Usart_SendChar(char data);
unsigned char Usart_Receive( void );
int Usart_printf(char var, FILE *stream);
void Usart_get_line (char *buff, int len);
void USART0_Senduint16 (uint16_t Data);
// void USART0_SendChar( unsigned char data );
// unsigned char USART0_Receive( void );
// void USART0_SendString(char str[]);

#endif /* USART_H_ */
```

C.1.1.13 Usart.c

../Code/DualOV7670/Usart.c

```
/*
 * Usart.c
 *
 * Created: 25/10/2012 22:25:04
 *  Author: hl13g10@ecs.soton.ac.uk
 */


#include "Usart.h"


void USART0_Init()
{
  uint16_t ubrr = UBBR;
  //Set baud rate
  UBRR0H = (unsigned char)(ubrr >>8);
  UBRR0L = (unsigned char)ubrr ;
  //Enable receiver and transmitter
```

```
  UCSR0B = (1<<RXEN0)|(1<<TXEN0);

  UCSR0C = 0x06; //set asynchronous, no parity, one stop bit, 8 bit transfer.

  //UCSR0B |= (1 << RXCIE0) | (1 << TXCIE0); //set RX and TX interrupt on
}
void Usart_SendChar(char data)
{
    // Wait for empty transmit buffer
    while ( !(UCSR0A & (1 << UDRE0)) );
    // Start transmission
    UDR0 = data;
}
unsigned char Usart_Receive( void )
{
  /* Wait for data to be received */
  while ( !(UCSR0A & (1<<RXC0)) )
  ;
  /* Get and return received data from buffer */
  //Usart_SendChar(UDR0);
  return UDR0;
}

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

void Usart_get_line (char *buff, int len)
{
  cli();
  char c;
  int i = 0;


  for (;;) {
    c = Usart_Receive();
    if (c == '\r') break;
    if ((c == '\b') && i) {
      i--;
      Usart_SendChar(c);
      continue;
    }
    if (c >= ' ' && i < len - 1) {  /* Visible chars */
      buff[i++] = c;
      Usart_SendChar(c);
    }
  }
  buff[i] = 0;
  Usart_SendChar('\n');
  sei();
```

```
}
void USART0_Senduint16 (uint16_t Data)
{
  Usart_SendChar(Data >> 8);
  Usart_SendChar(Data & 0xFF);
}
```

C.1.2 Dual Camera User Interface

C.1.2.1 DualCamera_UI.c

../Code/DualCamera_UI/DualCamera_UI.c

```
/*
 * DualCamera_UI.c
 *
 * Created: 12/11/2012 08:32:27
 *  Author: hslovett
 */


#include <avr/io.h>
#include <avr/interrupt.h>
#include "TWI_slave.h"

#define ButtonMask  0x0F

#define TWI_CMD_MASTER_WRITE 0x10
#define TWI_CMD_MASTER_READ  0x20

// When there has been an error, this function is run and takes care of it
unsigned char TWI_Act_On_Failure_In_Last_Transmission ( unsigned char
    TWIerrorMsg );

int main(void)
{

  char ButtonStatus = 0xFF;
  unsigned char TWI_slaveAddress;
  unsigned char messageBuff[TWI_BUFFER_SIZE];
  DDRD = 0xFF;// Port D is the LED output
  DDRC = 0x00; //PortC is the switch input
  //PORTC = 0xFF;
  TWI_slaveAddress = 0x15;
  TWI_Slave_Initialise( (unsigned char)((TWI_slaveAddress<<TWI_ADR_BITS) | (
    TRUE<<TWI_GEN_BIT) ));
  sei();
  TWI_Start_Transceiver();
    while(1)
    {

    ButtonStatus = (ButtonStatus & PINC) & ButtonMask;
    //PORTD = ButtonStatus;
        // Check if the TWI Transceiver has completed an operation.
        if ( ! TWI_Transceiver_Busy() )
```

```
        {
          // Check if the last operation was successful
          if ( TWI_statusReg.lastTransOK )
          {
            // Check if the last operation was a reception
            if ( TWI_statusReg.RxDataInBuf )
            {
              TWI_Get_Data_From_Transceiver(messageBuff, 2);
              // Check if the last operation was a reception as General Call
              if ( TWI_statusReg.genAddressCall )
              {
                // Put data received out to PORTB as an example.
                PORTB = messageBuff[0];
              }
              else // Ends up here if the last operation was a reception as Slave Address Match
              {
                // Example of how to interpret a command and respond.

                // TWI_CMD_MASTER_WRITE stores the data to PORTB
                if (messageBuff[0] == TWI_CMD_MASTER_WRITE)
                {
                  PORTD = messageBuff[1];
                }
                // TWI_CMD_MASTER_READ prepares the data from PINB in the transceiver buffer for the TWI master to fetch.
                if (messageBuff[0] == TWI_CMD_MASTER_READ)
                {
                  messageBuff[0] = ButtonStatus;
                  TWI_Start_Transceiver_With_Data( messageBuff, 1 );
              ButtonStatus = ButtonMask; //clear all logged button presses
                }
              }
            }
            else // Ends up here if the last operation was a transmission
            {
              //__no_operation(); // Put own code here.
            }
            // Check if the TWI Transceiver has already been started.
            // If not then restart it to prepare it for new receptions.
            if ( ! TWI_Transceiver_Busy() )
            {
              TWI_Start_Transceiver();
            }
          }
          else // Ends up here if the last operation completed unsuccessfully
          {
            //TWI_Act_On_Failure_In_Last_Transmission( TWI_Get_State_Info() );
          }
    }
    }
}

unsigned char TWI_Act_On_Failure_In_Last_Transmission ( unsigned char TWIerrorMsg )
{
  // A failure has occurred, use TWIerrorMsg to determine the nature of the failure
  // and take appropriate actions.
```

```
  // Se header file for a list of possible failures messages.

  // This very simple example puts the error code on PORTB and restarts the
    transceiver with
  // all the same data in the transmission buffers.
  //PORTB = TWIerrorMsg;
  TWI_Start_Transceiver();

  return TWIerrorMsg;
}
```

C.1.2.2 TWI_slave.h

../Code/DualCamera_UI/TWI_slave.h

```
/*****************************************************************************
*
* Atmel Corporation
*
* File              : TWI_Slave.h
* Compiler          : IAR EWAAVR 2.28a/3.10c
* Revision          : Revision: 2475
* Date              : Date: 2007-09-20 12:00:43 +0200 (to, 20 sep 2007)
* Updated by        : Author: mlarsson

* Support mail      : avr@atmel.com
*
* Supported devices : All devices with a TWI module can be used.
*                     The example is written for the ATmega16
*
* AppNote           : AVR311 - TWI Slave Implementation
*
* Description       : Header file for TWI_slave.c
*                     Include this file in the application.
*
****************************************************************************/
/*! \page MISRA
 *
 * General disabling of MISRA rules:
 * * (MISRA C rule 1) compiler is configured to allow extensions
 * * (MISRA C rule 111) bit fields shall only be defined to be of type
    unsigned int or signed int
 * * (MISRA C rule 37) bitwise operations shall not be performed on signed
    integer types
 * As it does not work well with 8bit architecture and/or IAR

 * Other disabled MISRA rules
 * * (MISRA C rule 109) use of union - overlapping storage shall not be used
 * * (MISRA C rule 61) every non-empty case clause in a switch statement shall
     be terminated with a break statement
*/

/****************************************************************************
  TWI Status/Control register definitions
****************************************************************************/
```

```
#define TWI_BUFFER_SIZE 4      // Reserves memory for the drivers transceiver buffer.
                               // Set this to the largest message size that will be sent including address byte.

/****************************************************************************
  Global definitions
****************************************************************************/

union TWI_statusReg_t                       // Status byte holding flags.
{
    unsigned char all;
    struct
    {
        unsigned char lastTransOK:1;
        unsigned char RxDataInBuf:1;
        unsigned char genAddressCall:1;                        // TRUE = General call, FALSE = TWI Address;
        unsigned char unusedBits:5;
    };
};

extern union TWI_statusReg_t TWI_statusReg;

//static unsigned char dont_sleep = 0;

/****************************************************************************
  Function definitions
****************************************************************************/
void TWI_Slave_Initialise( unsigned char );
unsigned char TWI_Transceiver_Busy( void );
unsigned char TWI_Get_State_Info( void );
void TWI_Start_Transceiver_With_Data( unsigned char * , unsigned char );
void TWI_Start_Transceiver( void );
unsigned char TWI_Get_Data_From_Transceiver( unsigned char *, unsigned char );

ISR( TWI_vect );

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

#define TRUE          1
#define FALSE         0

/****************************************************************************
  TWI State codes
****************************************************************************/
// General TWI Master staus codes
#define TWI_START                  0x08  // START has been transmitted
#define TWI_REP_START              0x10  // Repeated START has been transmitted
#define TWI_ARB_LOST               0x38  // Arbitration lost
```

```c
// TWI Master Transmitter staus codes
#define TWI_MTX_ADR_ACK            0x18  // SLA+W has been tramsmitted and ACK received
#define TWI_MTX_ADR_NACK           0x20  // SLA+W has been tramsmitted and NACK received
#define TWI_MTX_DATA_ACK           0x28  // Data byte has been tramsmitted and ACK received
#define TWI_MTX_DATA_NACK          0x30  // Data byte has been tramsmitted and NACK received

// TWI Master Receiver staus codes
#define TWI_MRX_ADR_ACK            0x40  // SLA+R has been tramsmitted and ACK received
#define TWI_MRX_ADR_NACK           0x48  // SLA+R has been tramsmitted and NACK received
#define TWI_MRX_DATA_ACK           0x50  // Data byte has been received and ACK tramsmitted
#define TWI_MRX_DATA_NACK          0x58  // Data byte has been received and NACK tramsmitted

// TWI Slave Transmitter staus codes
#define TWI_STX_ADR_ACK            0xA8  // Own SLA+R has been received; ACK has been returned
#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
#define TWI_STX_DATA_ACK           0xB8  // Data byte in TWDR has been transmitted; ACK has been received
#define TWI_STX_DATA_NACK          0xC0  // Data byte in TWDR has been transmitted; NOT ACK has been received
#define TWI_STX_DATA_ACK_LAST_BYTE 0xC8  // Last data byte in TWDR has been transmitted (TWEA = 0 ); ACK has been received

// TWI Slave Receiver staus codes
#define TWI_SRX_ADR_ACK            0x60  // Own SLA+W has been received ACK has been returned
#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
#define TWI_SRX_GEN_ACK            0x70  // General call address has been received; ACK has been returned
#define TWI_SRX_GEN_ACK_M_ARB_LOST 0x78  // Arbitration lost in SLA+R/W as Master; General call address has been received; ACK has been returned
#define TWI_SRX_ADR_DATA_ACK       0x80  // Previously addressed with own SLA+W; data has been received; ACK has been returned
#define TWI_SRX_ADR_DATA_NACK      0x88  // Previously addressed with own SLA+W; data has been received; NOT ACK has been returned
#define TWI_SRX_GEN_DATA_ACK       0x90  // Previously addressed with general call; data has been received; ACK has been returned
#define TWI_SRX_GEN_DATA_NACK      0x98  // Previously addressed with general call; data has been received; NOT ACK has been returned
#define TWI_SRX_STOP_RESTART       0xA0  // A STOP condition or repeated START condition has been received while still addressed as Slave

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

C.1.2.3 TWI_slave.c

../Code/DualCamera_UI/TWI_slave.c

```c
/*****************************************************************************
*
* Atmel Corporation
*
* File              : TWI_Slave.c
* Compiler          : IAR EWAAVR 2.28a/3.10c
* Revision          : Revision: 2475
* Date              : Date: 2007-09-20 12:00:43 +0200 (to, 20 sep 2007)
* Updated by        : Author: mlarsson
*
* Support mail      : avr@atmel.com
*
* Supported devices : All devices with a TWI module can be used.
*                     The example is written for the ATmega16
*
* AppNote           : AVR311 - TWI Slave Implementation
*
* Description       : This is sample driver to AVRs TWI module.
*                     It is interupt driveren. All functionality is controlled through
*                     passing information to and from functions. Se main.c for samples
*                     of how to use the driver.
*
****************************************************************************/
/*! \page MISRA
 *
 * General disabling of MISRA rules:
 * * (MISRA C rule 1) compiler is configured to allow extensions
 * * (MISRA C rule 111) bit fields shall only be defined to be of type unsigned int or signed int
 * * (MISRA C rule 37) bitwise operations shall not be performed on signed integer types
 * As it does not work well with 8bit architecture and/or IAR

 * Other disabled MISRA rules
 * * (MISRA C rule 109) use of union - overlapping storage shall not be used
 * * (MISRA C rule 61) every non-empty case clause in a switch statement shall be terminated with a break statement
*/

#include <avr/io.h>
#include <avr/interrupt.h>
#include "TWI_slave.h"

static unsigned char TWI_buf[TWI_BUFFER_SIZE];     // Transceiver buffer. Set the size in the header file
static unsigned char TWI_msgSize  = 0;             // Number of bytes to be transmitted.
static unsigned char TWI_state    = TWI_NO_STATE;  // State byte. Default set to TWI_NO_STATE.

// This is true when the TWI is in the middle of a transfer
// and set to false when all bytes have been transmitted/received
```

```
// Also used to determine how deep we can sleep.
static unsigned char TWI_busy = 0;

union TWI_statusReg_t TWI_statusReg = {0};           // TWI_statusReg is defined in TWI_Slave.h

/****************************************************************************
Call this function to set up the TWI slave to its initial standby state.
Remember to enable interrupts from the main application after initializing the TWI.
Pass both the slave address and the requrements for triggering on a general call in the
same byte. Use e.g. this notation when calling this function:
TWI_Slave_Initialise( (TWI_slaveAddress<<TWI_ADR_BITS) | (TRUE<<TWI_GEN_BIT) );
The TWI module is configured to NACK on any requests. Use a TWI_Start_Transceiver function to
start the TWI.
****************************************************************************/
void TWI_Slave_Initialise( unsigned char TWI_ownAddress )
{
  TWAR = TWI_ownAddress;                            // Set own TWI slave address. Accept TWI General Calls.
  TWCR = (1<<TWEN)|                                 // Enable TWI-interface and release TWI pins.
         (0<<TWIE)|(0<<TWINT)|                      // Disable TWI Interupt.
         (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|           // Do not ACK on any requests, yet.
         (0<<TWWC);                                 //
  TWI_busy = 0;
}

/****************************************************************************
Call this function to test if the TWI_ISR is busy transmitting.
****************************************************************************/
unsigned char TWI_Transceiver_Busy( void )
{
  return TWI_busy;
}

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

/****************************************************************************
Call this function to send a prepared message, or start the Transceiver for reception. Include
a pointer to the data to be sent if a SLA+W is received. The data will be copied to the TWI buffer.
```

```
Also include how many bytes that should be sent. Note that unlike the similar Master function, the
Address byte is not included in the message buffers.
The function will hold execution (loop) until the TWI_ISR has completed with the previous operation,
then initialize the next operation and return.
****************************************************************************/
void TWI_Start_Transceiver_With_Data( unsigned char *msg, unsigned char msgSize )
{
  unsigned char temp;

  while ( TWI_Transceiver_Busy() ) {}             // Wait until TWI is ready for next transmission.

  TWI_msgSize = msgSize;                        // Number of data to transmit.
  for ( temp = 0; temp < msgSize; temp++ )      // Copy data that may be transmitted if the TWI Master requests data.
  {
    TWI_buf[ temp ] = msg[ temp ];
  }
  TWI_statusReg.all = 0;
  TWI_state         = TWI_NO_STATE ;
  TWCR = (1<<TWEN)|                             // TWI Interface enabled.
         (1<<TWIE)|(1<<TWINT)|                  // Enable TWI Interupt and clear the flag.
         (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|       // Prepare to ACK next time the Slave is addressed.
         (0<<TWWC);                             //
  TWI_busy = 1;
}

/****************************************************************************
Call this function to start the Transceiver without specifing new transmission data. Useful for restarting
a transmission, or just starting the transceiver for reception. The driver will reuse the data previously put
in the transceiver buffers. The function will hold execution (loop) until the TWI_ISR has completed with the
previous operation, then initialize the next operation and return.
****************************************************************************/
void TWI_Start_Transceiver( void )
{
  while ( TWI_Transceiver_Busy() ) {}             // Wait until TWI is ready for next transmission.
  TWI_statusReg.all = 0;
  TWI_state         = TWI_NO_STATE ;
  TWCR = (1<<TWEN)|                             // TWI Interface enabled.
         (1<<TWIE)|(1<<TWINT)|                  // Enable TWI Interupt and clear the flag.
         (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|       // Prepare to ACK next time the Slave is addressed.
         (0<<TWWC);                             //
  TWI_busy = 0;
}
/****************************************************************************
Call this function to read out the received data from the TWI transceiver buffer. I.e. first call
```

```
TWI_Start_Transceiver to get the TWI Transceiver to fetch data. Then Run this function to collect the
data when they have arrived. Include a pointer to where to place the data and the number of bytes
to fetch in the function call. The function will hold execution (loop) until the TWI_ISR has completed
with the previous operation, before reading out the data and returning.
If there was an error in the previous transmission the function will return the TWI State code.
****************************************************************************/
unsigned char TWI_Get_Data_From_Transceiver( unsigned char *msg, unsigned char msgSize )
{
  unsigned char i;

  while ( TWI_Transceiver_Busy() ) {}             // Wait until TWI is ready for next transmission.

  if( TWI_statusReg.lastTransOK )               // Last transmission completed successfully.
  {
    for ( i=0; i<msgSize; i++ )                 // Copy data from Transceiver buffer.
    {
      msg[ i ] = TWI_buf[ i ];
    }
    TWI_statusReg.RxDataInBuf = FALSE;          // Slave Receive data has been read from buffer.
  }
  return( TWI_statusReg.lastTransOK );
}


// ********** Interrupt Handlers ********** //
/****************************************************************************
This function is the Interrupt Service Routine (ISR), and called when the TWI interrupt is triggered;
that is whenever a TWI event has occurred. This function should not be called directly from the main
application.
****************************************************************************/
ISR(TWI_vect)
{
  static unsigned char TWI_bufPtr;

  switch (TWSR)
  {
    case TWI_STX_ADR_ACK:            // Own SLA+R has been received; ACK has been returned
//    case TWI_STX_ADR_ACK_M_ARB_LOST: // Arbitration lost in SLA+R/W as Master; own SLA+R has been received; ACK has been returned
      TWI_bufPtr   = 0;                                 // Set buffer pointer to first data location
    case TWI_STX_DATA_ACK:           // Data byte in TWDR has been transmitted; ACK has been received
      TWDR = TWI_buf[TWI_bufPtr++];
      TWCR = (1<<TWEN)|                                 // TWI Interface enabled
```

```
                (1<<TWIE)|(1<<TWINT)|                              // Enable TWI Interupt and clear the flag to send byte
                (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|                   //
                (0<<TWWC);                                         //
        TWI_busy = 1;
        break;
      case TWI_STX_DATA_NACK:          // Data byte in TWDR has been transmitted; NACK has been received.
                                       // I.e. this could be the end of the transmission.
        if (TWI_bufPtr == TWI_msgSize) // Have we transceived all expected data?
        {
          TWI_statusReg.lastTransOK = TRUE;                        // Set status bits to completed successfully.
        }
        else                          // Master has sent a NACK before all data where sent.
        {
          TWI_state = TWSR;                                        // Store TWI State as errormessage.
        }

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

        TWI_busy = 0;   // Transmit is finished, we are not busy anymore
        break;
      case TWI_SRX_GEN_ACK:            // General call address has been received; ACK has been returned
//    case TWI_SRX_GEN_ACK_M_ARB_LOST: // Arbitration lost in SLA+R/W as Master; General call address has been received; ACK has been returned
        TWI_statusReg.genAddressCall = TRUE;
      case TWI_SRX_ADR_ACK:            // Own SLA+W has been received ACK has been returned
//    case TWI_SRX_ADR_ACK_M_ARB_LOST: // Arbitration lost in SLA+R/W as Master; own SLA+W has been received; ACK has been returned
                                                                   // Dont need to clear TWI_S_statusRegister.generalAddressCall due to that it is the default state.
        TWI_statusReg.RxDataInBuf = TRUE;
        TWI_bufPtr   = 0;                                          // Set buffer pointer to first data location

                                                                   // Reset the TWI Interupt to wait for a new event.
        TWCR = (1<<TWEN)|                                          // TWI Interface enabled
                (1<<TWIE)|(1<<TWINT)|                              // Enable TWI Interupt and clear the flag to send byte
                (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|                   // Expect ACK on this transmission
                (0<<TWWC);
        TWI_busy = 1;
```

```
        break;
      case TWI_SRX_ADR_DATA_ACK:       // Previously addressed with own SLA+W; data has been received; ACK has been returned
      case TWI_SRX_GEN_DATA_ACK:       // Previously addressed with general call; data has been received; ACK has been returned
        TWI_buf[TWI_bufPtr++]     = TWDR;
        TWI_statusReg.lastTransOK = TRUE;                 // Set flag transmission successfull.
                                                          // Reset the TWI Interupt to wait for a new event.
        TWCR = (1<<TWEN)|                                 // TWI Interface enabled
               (1<<TWIE)|(1<<TWINT)|                      // Enable TWI Interupt and clear the flag to send byte
               (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|           // Send ACK after next reception
               (0<<TWWC);                                 //
        TWI_busy = 1;
        break;
      case TWI_SRX_STOP_RESTART:       // A STOP condition or repeated START condition has been received while still addressed as Slave
                                                          // Enter not addressed mode and listen to address match
        TWCR = (1<<TWEN)|                                 // Enable TWI-interface and release TWI pins
               (1<<TWIE)|(1<<TWINT)|                      // Enable interrupt and clear the flag
               (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|           // Wait for new address match
               (0<<TWWC);                                 //

        TWI_busy = 0;  // We are waiting for a new address match, so we are not busy

        break;
      case TWI_SRX_ADR_DATA_NACK:      // Previously addressed with own SLA+W; data has been received; NOT ACK has been returned
      case TWI_SRX_GEN_DATA_NACK:      // Previously addressed with general call; data has been received; NOT ACK has been returned
      case TWI_STX_DATA_ACK_LAST_BYTE: // Last data byte in TWDR has been transmitted (TWEA = 0 ); ACK has been received
//    case TWI_NO_STATE                // No relevant state information available; TWINT = 0
      case TWI_BUS_ERROR:         // Bus error due to an illegal START or STOP condition
        TWI_state = TWSR;                //Store TWI State as errormessage, operation also clears noErrors bit
        TWCR =   (1<<TWSTO)|(1<<TWINT);  //Recover from TWI_BUS_ERROR, this will release the SDA and SCL pins thus enabling other devices to use the bus
        break;
      default:
        TWI_state = TWSR;                                 // Store TWI State as errormessage, operation also clears the Success bit.
        TWCR = (1<<TWEN)|                                 // Enable TWI-interface and release TWI pins
               (1<<TWIE)|(1<<TWINT)|                      // Keep interrupt enabled and clear the flag
```

```
              (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|           // Acknowledge on any
    new requests.
              (0<<TWWC);                                  //

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

C.2 MATLAB Code for Image Algorithm Prototyping

C.2.0.4 loadimages.m

../MATLAB/loadimages.m

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

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

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

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

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

C.2.0.5 GetSubImage.m

../MATLAB/GetSubImage.m

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

XLow = StartCoordinates(1)-(BoxSize(1)/2);
YLow = StartCoordinates(2)-(BoxSize(2)/2);
if(XLow<1)
    XLow = 1;
end

if(YLow < 1)
    YLow = 1;
end
```

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

%SubImage = zeros(BoxSize);
for i = XLow:XHigh
    for j = YLow:YHigh
        if LZ == 3
            for z = 1:3
                SubImage(i-XLow+1,j-YLow+1,z) = Image(i,j,z);
            end
        elseif LZ == 1
            SubImage(i-XLow+1,j-YLow+1) = Image(i,j);
        else
            error('Number of Dimensions "%d" are not supported', LZ);
        end
    end
end

end
```

C.2.0.6 SADAll.m

../MATLAB/SADAll.m

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

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

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

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

%Display
figure;
subplot(2,2,1);
imshow(left);
title('Left Image');
```

```
subplot(2,2,2);
imshow(right);
title('Right Image');

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

figure;
surf(Results);
shading flat;
%end
```

C.2.0.7 SSDAll.m

../MATLAB/SSDAll.m

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

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

for i = 25:(I-25)
    for j = 25:(J-25)
        LeftSub = GetSubImage(left, BoxSize, [i, j]);
        Diff = LeftSub - RightSub;
        Diff = Diff.^2;
        Results(i,j) = sum(Diff(:));
    end
end

%Display
figure;
subplot(2,2,1);
imshow(left);
title('Left Image');

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

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

figure;
```

```
surf(Results);
shading flat;
%end
```

C.2.0.8 NCC.m

../MATLAB/NCC.m

```

loadimages;
show;
BoxSize = [50,50];
MaxConfMatches = 20;
%SubCoord = [145, 300];
figure(1);
%[rightSub, rect_Sub] = imcrop(right);
figure(2);
imshow(right);
rSubCoord = ginput(1);
rSubCoord = [190,190]; [rSubCoord(2), rSubCoord(1)];
rSubCoord = round(rSubCoord);
close;
tic;
rightSub = GetSubImage(right, BoxSize, rSubCoord);
%imshow(rightSub);
rightSubGray = rgb2gray(rightSub);
leftGray = rgb2gray(left);
rightGray = rgb2gray(right);
cL = normxcorr2(rightSubGray(:,:), leftGray(:,:));
figure(2);
% subplot(1,2,1);
surf(cL), shading flat;
title('Normalised Cross Correlation of Right Sub and Left Image');
toc;
% cR = normxcorr2(rightSubGray(:,:), rightGray(:,:));
% subplot(1,2,2);
% surf(cR), shading flat;
% title('Normalised Cross Correlation of Right Sub and Right Image');

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

%Find coordintes of best match.
[Y,X] = size(cL);
maxValue = 0;
LeftMatchCoord = [0,0];
NumConfidentMatches = 0;

for i = 1:X
   for j = 1:Y
       Val = cL(j,i);
       if Val > 0.9
           NumConfidentMatches = NumConfidentMatches + 1;
```

```
        end
        if Val > maxValue
            maxValue = Val;
            LeftMatchCoord = [j-(BoxSize(1) / 2 ), i-(BoxSize(2) / 2 )];
        end
    end
end

Result = [maxValue, LeftMatchCoord];
figure(1);
if NumConfidentMatches >= 1 && NumConfidentMatches < MaxConfMatches
    left(LeftMatchCoord(1)-(BoxSize(1)/2):LeftMatchCoord(1)+(BoxSize(1)/2), LeftMatchCoord(2)-(BoxSize(2))/2)=255;
    left(LeftMatchCoord(1)-(BoxSize(1)/2):LeftMatchCoord(1)+(BoxSize(1)/2), LeftMatchCoord(2)+(BoxSize(2))/2)=255;
    left(LeftMatchCoord(1)-(BoxSize(1)/2),LeftMatchCoord(2)-(BoxSize(2)/2): LeftMatchCoord(2)+(BoxSize(2))/2)=255;
    left(LeftMatchCoord(1)+(BoxSize(1)/2),LeftMatchCoord(2)-(BoxSize(2)/2): LeftMatchCoord(2)+(BoxSize(2)/2))=255;

    right(rSubCoord(1)-(BoxSize(1)/2):rSubCoord(1)+(BoxSize(1)/2),rSubCoord(2)-(BoxSize(2)/2))=255;
    right(rSubCoord(1)-(BoxSize(1)/2):rSubCoord(1)+(BoxSize(1)/2),rSubCoord(2)+(BoxSize(2)/2))=255;
    right(rSubCoord(1)-(BoxSize(1)/2),rSubCoord(2)-(BoxSize(2)/2):rSubCoord(2)+(BoxSize(2)/2))=255;
    right(rSubCoord(1)+(BoxSize(1)/2),rSubCoord(2)-(BoxSize(2)/2):rSubCoord(2)+(BoxSize(2)/2))=255;

    subplot(1,2,1);
    imshow(left);
    subplot(1,2,2);
    imshow(right);
%      LeftMatchCoord
%      rSubCoord
%      NumConfidentMatches
    Distance = Range(rSubCoord(2), LeftMatchCoord(2));
    sprintf('Distance to Object = %d metres', Distance)
elseif NumConfidentMatches >= MaxConfMatches
    title(sprintf('Too many matches found : %d', NumConfidentMatches));
else
    title(sprintf('No Reliable Match Found'));

end
```

References

Atmel Corporation. *AVR311: TWI Slave*, 2007.

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.

Patrick Goebel. Robot cartography: Ros + slam, 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.

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

Microsoft. Xbox 360 kinect, 2012.

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

Mark Nixon. Computer vision demonstration website: Corner detection, 2005.

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.

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/`.