UNIVERSITY OF SOUTHAMPTON

FACULTY OF PHYSICAL AND APPLIED SCIENCES

Electronics and Computer Science

Two Dimensional Stereoscopic Mapping Robot

by

Henry S. Lovett

Technical Report

December 3, 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^2C Inter-Integrated Circuit
TWI Two Wire Interface
SCCB Serial Camera Control Bus
kB KiloBytes

Chapter 1

Introduction

The Introduction to my Report ...

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

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

2.2 Image Algorithms

2.2.1 Comparison Algorithms

Before

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 (**?**

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.

3.3 Motor Control

do something of this SOON

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

3.4 PCB Development

Also do something of this SOON

Chapter 4

Investigation into Vision Algorithms

4.1 Comparison

4.2 Range Finding

Chapter 5

Conclusions

It works.

Appendix A

Circuit Diagrams

A.1 OV7670 Breakout Board Schematic

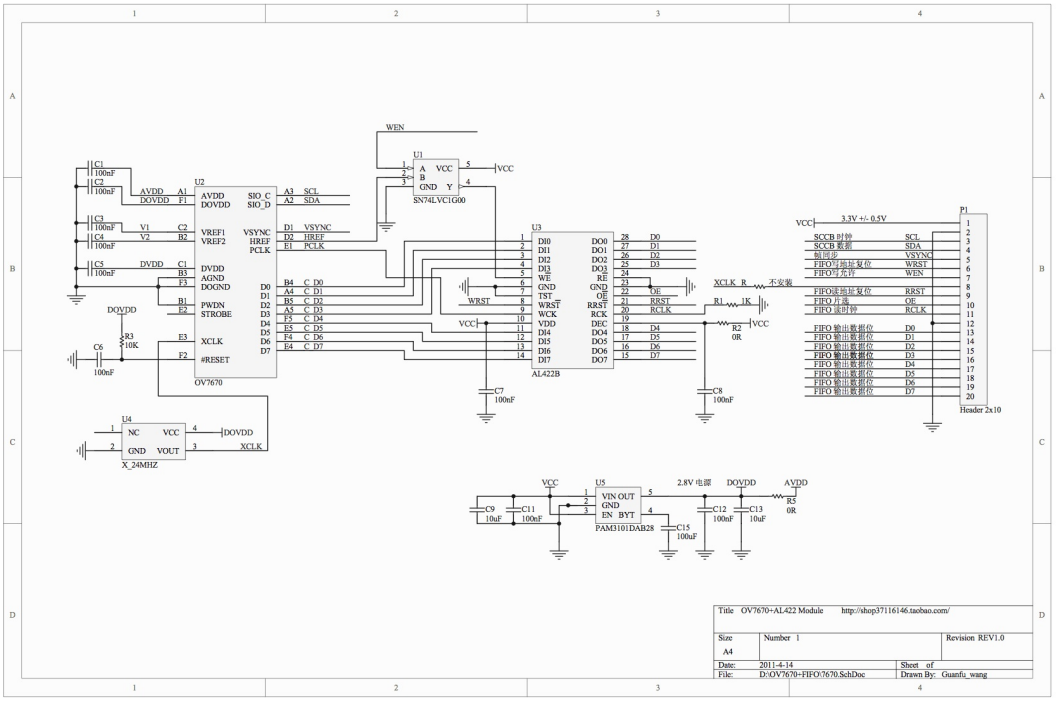

Figure A.1: The circuit diagram for the OV7670 breakout board

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

References

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

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

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

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