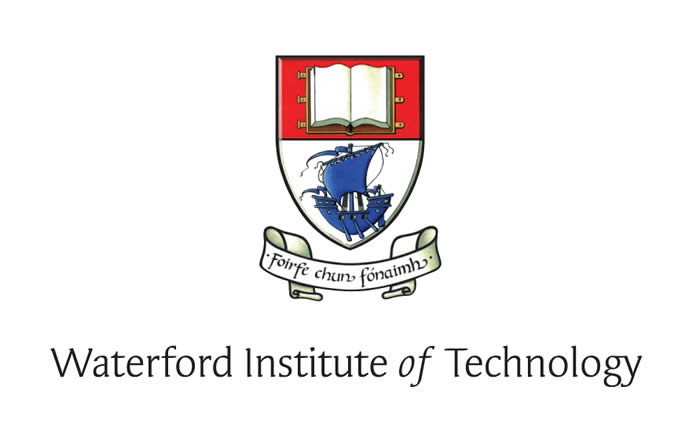
****

**Personal Computer Based Vision System.**

**Student:** TJ Fitzpatrick.

**Student Number:** 20027865.

**Date:** April 24th 2010.

**Lecturers:** Jason Berry, Siobhan Wall, Martin Hayes, John Vance.

**Class:** Embedded Systems Project.

**Course:** Bachelor of Engineering in Electronic Engineering.

**Department:** Department of Technology.

**Waterford Institute of Technology.**

****

**Chapter 1: Introduction.**

**1.1:** Introduction. (Page 5)

**1.2:** Functionality of Technology. (Page 6)

**1.3:** Technology Overview. (Page 7)

**Chapter 2: Hardware Overview.**

**2.1:** Component List. (Page 9)

**2.2:** Can Bus Circuit Schematic. (Page 10)

**2.3:** Hardware Operation. (Page 11)

2.3.1: MCP2510.

2.3.2: PCA82C250T.

**2.4:** The Can Bus. (Page 14)

**Chapter 3: Software Overview.**

**3.1:** The Can Bus Implementation. (Page 17)

3.1.1: Can Bus Flow Diagram.

3.1.2: Can Bus Code Explanation.

**3.2:** A PC Based Vision System. (Page 20)

3.2.1: Gaussian Blur.

3.2.2: Logitech Orbit (pt1).

3.2.3: RGB Filter.

3.2.4: Flood Fill.

3.2.5: Centre of Gravity.

3.2.6: VB Script.

3.2.7: Serial Module.

3.2.8: Logitech Orbit (pt2).

**Chapter 4: Testing.**

**4.1:** Testing. (Page 32)

**4.2:** Possible Problems. (Page 33)

**Chapter 5: Conclusion.**

**5.1:** Conclusion.

**1: Introduction.**

**1.1: Introduction.**

At the start of this year we (The students) were given our project applications for a robot called BENGIEMAN.

BENGIEMAN is a robot consisting of 5 layers which is mounted on “Road wheels” for movement.



**(Figure 1.1.1 is of the BENGIEMAN Structure.)**



**(Figure 1.1.2 is of BENGIEMAN’s “Road Wheels”.)**

Each layer of the robot will consist of a student’s application, and in some instances there could be even two applications.

The project application that I received was a “PC Based Vision System”.

This type of system would be able to distinguish red shaped squares from other coloured squares and base actions on the information created by the squares location on a personal computer screen.

As all applications had a starting point, mine had one too, in the brief where the application was stored I was asked to study and understand other PC based vision systems that were created using software called “Robo-realm”.

The application I was asked to study is found at the following link: http://www.roborealm.com/tutorial/Surveyor\_SRV1b\_Trail/slide100.php

Robo-realm is a program used for PC based vision and image analysis that has compiled many image and vision processing techniques.

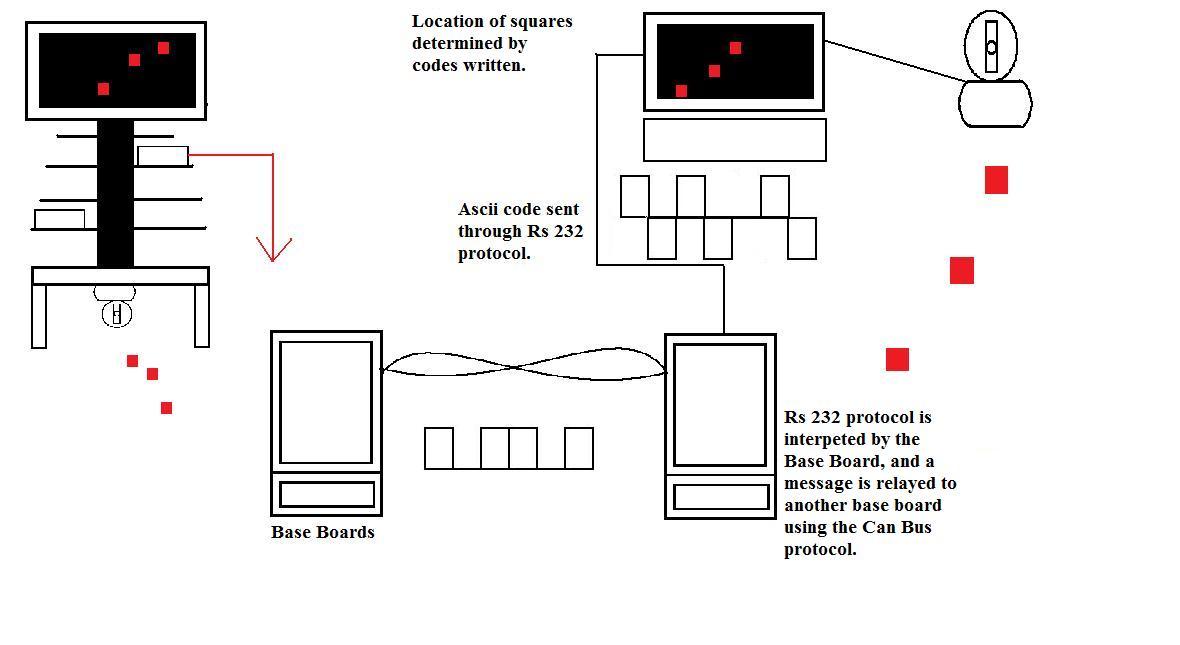
Once Robo-realm identifies a path it will communicate to “A base-board” which is on a layer of BENGIEMAN’s structure using an RS-232 protocol.

The Base-board in turn will then communicate to other boards the location of the path being followed on the screen, through a Can-Bus Protocol.

The Can-Bus is a two wire standard, electronic bus designed to allow microcontrollers and other devices to communicate with one another.

**1.2: Functionality of Technology.**

1.2.1: Functionality Block Diagram.

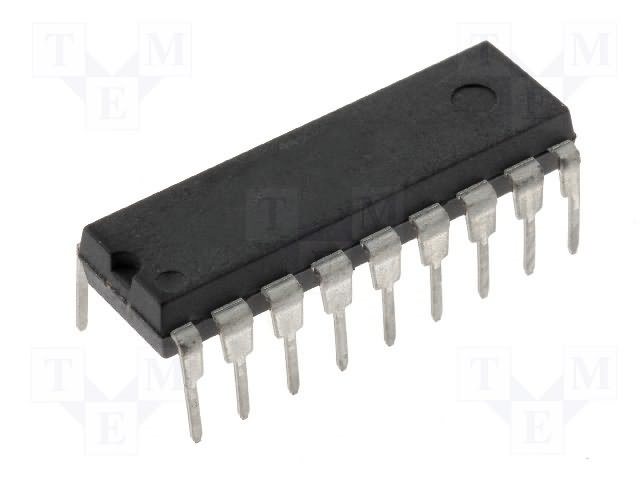
****

**1.3: Technology Overview.**

In order to implement a new type of technology, new micro-chips must be used.

The chips used to implement the Can Bus were the following:

MCP2510.



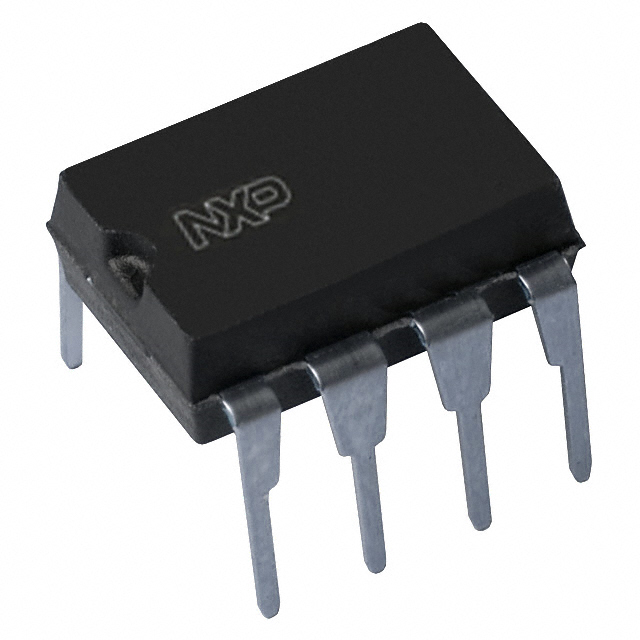
**(http://www.tme.eu/katalog\_pics/d/3/3/d33100d95915bd2d1d269627b51a3dfa/mcp2510-i\_p.jpg)**

**(Figure 1.1.2.1 is of the MCP2510 chip.)**

The MCP2510 is a “Controller Area Network” (CAN) controller that implements a CAN specification of 2.0.

It is able to transmit and receive standard and extended messages to other Micro-controllers also it is capable of acceptance filtering and message management.

PCA82C250T

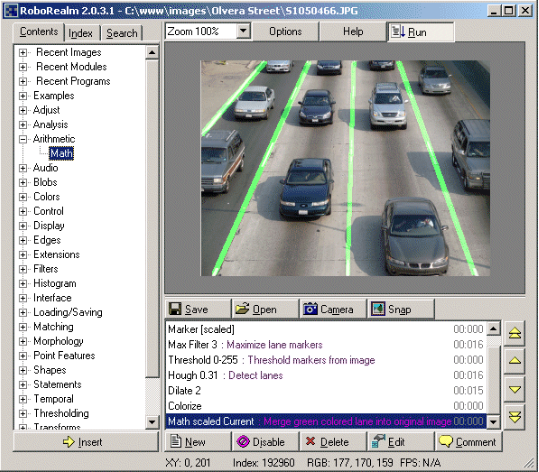


**(http://media.digikey.com/photos/NXP%20Semi%20Photos/568-8-DIP,SOT97-1.jpg)**

**(Figure 1.1.2.2 is of the PCA82C250T Chip.)**

The PCA82C250T is the actual interface between the Can controller itself and the physical bus, it is used in many high speed applications, it also incorporates a current limiting circuit that protects the transmitter output from a short circuit to a positive or negative battery supply voltage.

Robo-Realm.



**(http://www.roborealm.com/images/homepage\_large.gif)**

**(Figure 1.1.2.3 is of a Robo-Realm module being implemented.)**

Robo-realm is a program used in image analysis and vision software.

It incorporates multiple modules that allow for use in PC image analysis, it also allows users to communicate to others via an RS232 port on the machine it’s being run on.

Robo-Realm allows users to create their own systems using complex techniques involved in image processing.

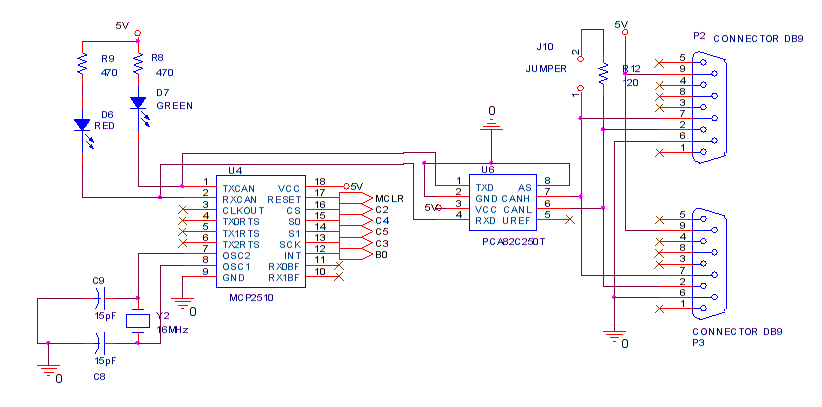
**Chapter 2: Hardware Overview.**

**2.1: Components List.**

| **Component.** | **Value.** | **Quantity.** |
| --- | --- | --- |
| Resistor. | X2 470(ohm) ---- x1 120 (ohm) | x3 |
| Capacitor. | 15pF | x2 |
| Db9 Connector. | ---------- | x2 |
| MCP2510. | ---------- | x2 |
| PCA82C250T | ---------- | x2 |
| Quartz Crystal. | 16Mhz | x1 |
| L.E.D | ---------- | x2 |
| Logitech Sphere Orbit MP Web-Camera. | ---------- | x1 |
| Jumper. | ---------- | x1 |

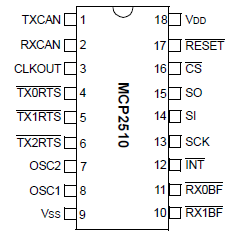
Majority of the hardware used for this base-board was implemented in semester 1, the only part of the base-board not implemented was the CAN Bus itself.

**2.2: Can Circuit Schematic.**



**2.3: Hardware Operation.**

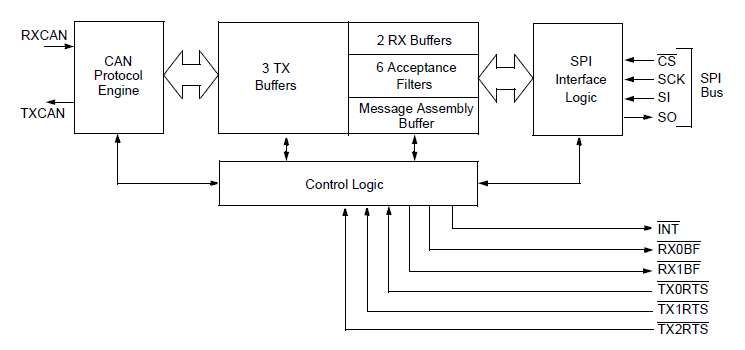
2.3.1: MCP2510.



**(http://www.farnell.com/datasheets/79076.pdf)**

**(Figure 2.3.1.1 is of the MCP2510 can Chip.)**

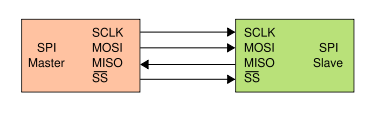
The MCP2510 is a stand alone controller that has been created to bridge applications that incorporate the CAN Bus protocol, the device consists of three blocks.



**(http://www.farnell.com/datasheets/79076.pdf)**

**(Figure 2.3.1.2 is of the internal composition MCP2510 can Chip.)**

On the right of figure 2.3.1.2 there are three inputs and one output, these I/O pins are interfaced to the Pic16F877A using an SPI Bus protocol, the SPI protocol is a synchronous serial data protocol that implements a master/slave circuit this allow for full duplex transmission.

[](http://upload.wikimedia.org/wikipedia/commons/e/ed/SPI_single_slave.svg)

**(http://en.wikipedia.org/wiki/File:SPI\_single\_slave.svg)**

**(Figure 2.3.1.3 is of a master/slave circuit.)**

The CS pin is known as Chip Select, and must be low for the device to be able to communicate to a master controller on the bus line, since there is only one chip during the design of the base board it is recommended, that this pin remains logic 0, this will keep the chip enabled to receive data even when it’s not in use and it will also save power.

SCK pin is used to synchronise the clock to the master’s, this has to be done in order to execute requests by the master to transmit data, and in this case the SCK pin is connected to pin18 (also SCK) of the Pic16f877a, synchronous transmission allows for data to be sent simultaneously.

Both pins SI, and S0 are the Serial Input (SI) and Serial Output (SO) for the SPI bus, it is on these pins that data is transmitted and received, both too and from the Master.

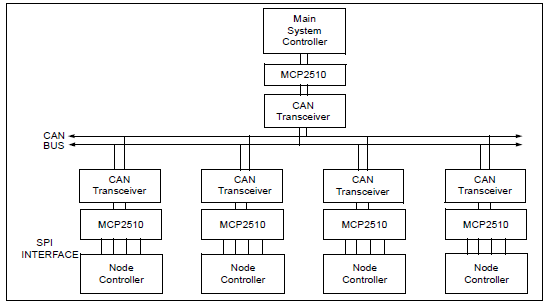
On the bottom right of Figure 2.3.1.2 there are six pins, three inputs(I) and three outputs(O).

The three inputs, TXRTS(0), TXRTS(1), and TXRTS(2) are for communication between the master and slave, they are request lines that the master uses before requesting the slave for data, in this project they are not implemented.

The two outputs RXBF(0), and RXBF(1) are digital buffers, a digital buffer is used in many areas of electronics from high current output drives, to storing information when the chip is unable to process data fast enough, once the current data is processed the MCP2510 will then begin to process data in its buffers, again in this current design the buffers are not being implemented.

The INT pin is an interrupt, an interrupt is a signal that is sent to the processor of the Pic16f877a to allow the MCP2510 time to process information that has to be transmitted or that it has received.

Once data passes through the MCP2510 it will then be sent to the CAN transceiver on the lines TX/RX to be sent onto the physical bus.

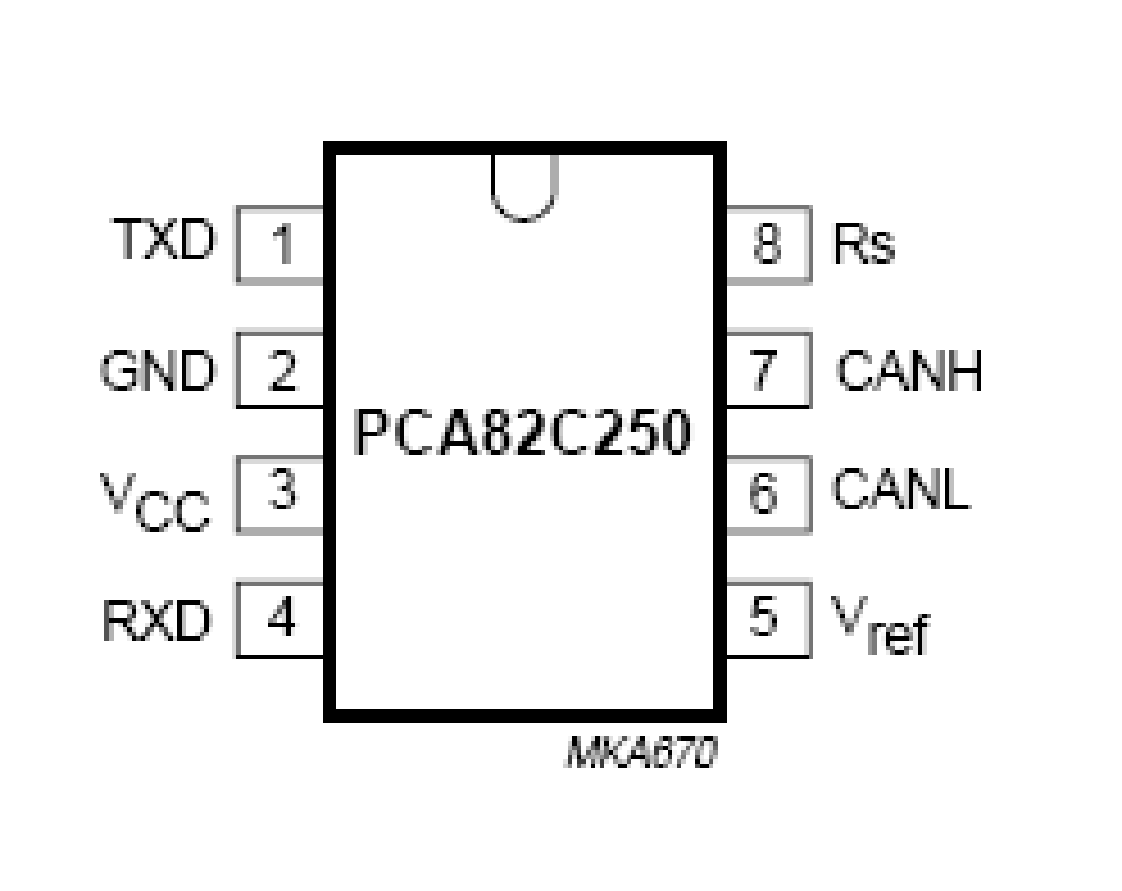


**(http://www.farnell.com/datasheets/79076.pdf)**

**(Figure 2.3.1.4 is of a typical CAN system drop-down configuration.)**

2.3.2: PCA82C250T.

The PCA82C250T is the interface between the can controller and the bus itself.



**(http://www.farnell.com/datasheets/101903.pdf)**

**(Figure 2.3.3.1 is of the can interface chip.)**

There are 8 pins on this chip although in this design only 7 are physically used.

| **Pin.** | **Description.** |
| --- | --- |
| 1 | Data for transmitting input. |
| 2 | Ground |
| 3 | Voltage supply. |
| 4 | Data received output. |
| 5 | Voltage reference. |
| 6 | Low-level CAN Voltage. |
| 7 | High-level CAN Voltage. |
| 8 | Slop resistor input. |

Pins 1, and 4 will be the lines that data travels to and from the MCP2510.

Pin 6 and pin 7 will be the pins that physically connect to the bus, it is through these pins that data will travel to and from other devices.

So now that we do know a small quantity of information about the hardware of the Can Bus, its time to find out, how does this Bus actually function?

**2.4: The CAN Bus.**

The can bus is a two wire bus that consists of a twisted pair of shielded/unshielded wires to connect devices to one another.

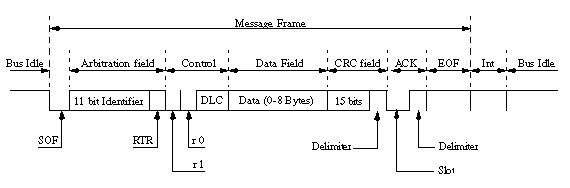


**(http://en.wikipedia.org/wiki/File:25\_pair\_color\_code\_chart.svg)**

**(Figure 2.4.1 is of a twisted pair.)**

It will operate in the most extreme conditions, and has wide-ranging error checking techniques.

The format of a CAN message consists of many different fields, these fields indicate various quantities of information that other devices on the bus may find functional such as “address” and “Data” ect.



**(http://www.mjschofield.com/canworks.htm)**

**(Figure 2.4.2 is of a basic CAN message.)**

The SOF or start of frame field signals to other devices that a message is about to be sent.

The arbitration field is the address of each node/device, this address also determines the priority of the message, a lower address node indicates higher the priority, this field is used to solve disputes on the bus between nodes, as to who gets to send a message first.

The RTR bit is a Remote Transmission Request, this field signals to the receiver that a response has been requested, this response could be a sensor reading.

The Control field consists of six individual bits, two of these six are reserved, r0, and r1 whilst the other four are for a Data Length Check, this type of check indicates the amount of information that there is in the Data Field.

The Data Field carries the data that is transmitted or received from a node, this type of data will be generated from sensor readings, there are 8 Bytes of data that can be sent, which means that 2048 (256 x 8) different types of messages can be sent.

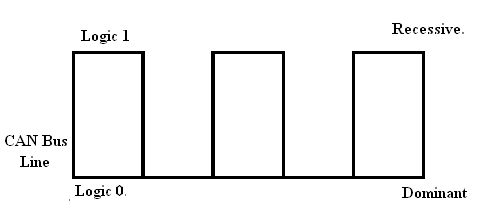
A CRC field is a cyclic redundancy field that is designed to detect changes in pieces of data.

The ACK field consists of two bits, a Slot Bit and a Deliminator Bit, the Slot bit allows for a break in the message field whilst the Deliminator bit basically closes the ACK field.

The INT field or intermission field is a field made up by three bits that close the message frame completely once this filed has finished the bus is predicted to be free an allowed for use.

Now that we know the format of a CAN message we need to see why two nodes that send messages at the same time don’t collide, this is done through the Arbitration field and is known as Non-Destructive Bitwise Arbitration.

On the students version of CAN bus there are addresses ranging from 0 – 140, on any can bus line there are two states dominant and recessive.



**(Figure 2.4.3 is of an example of both dominant and recessive states.)**

Logic 0 is dominant over logic 1, since this occurs the device with the lowest address will have highest priority.

| **Address** | **Bit0** | **Bit1** | **Bit2** | **Bit3** | **Bit4** | **Bit5** | **Bit6** | **Bit7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| And’d | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The above and below table both show bitwise arbitration.

| **Address** | **Bit0** | **Bit1** | **Bit2** | **Bit3** | **Bit4** | **Bit5** | **Bit6** | **Bit7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 20 | 0 | 0 | 0 | 1 | X | X | X | X |
| And’d | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

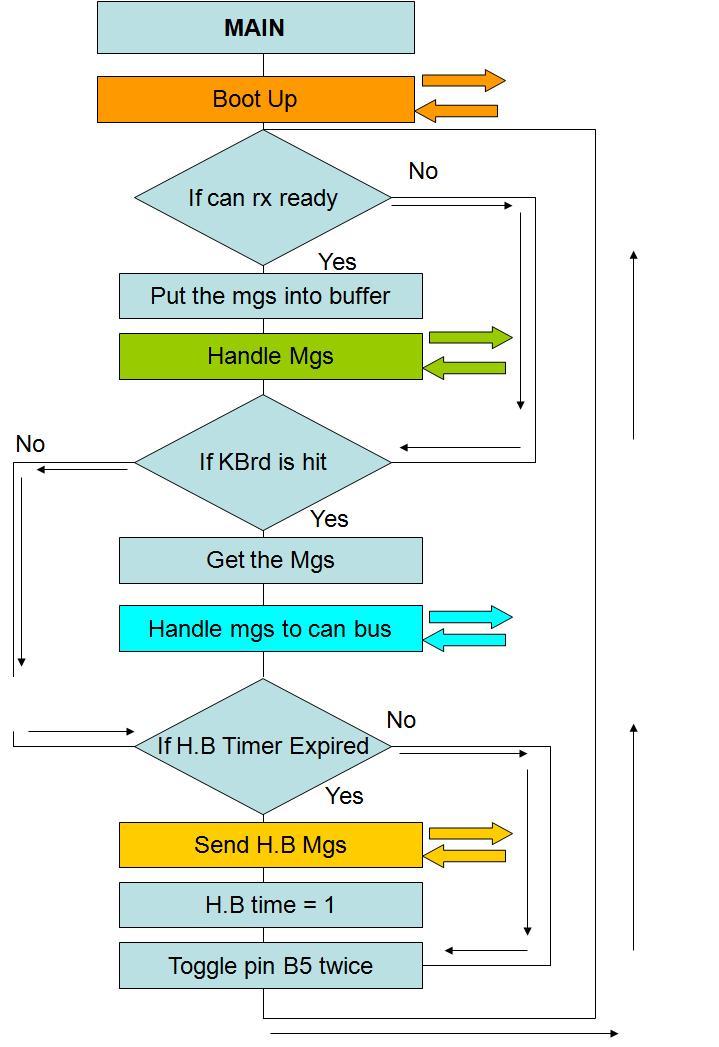
X = don’t care.

Now that we know what Bitwise Arbitration is, we can now understand what the two pins CANL, CANH function as a component of the PCA83C250T.

**Chapter 3: Software Overview.**

**3.1: The Can Bus Implementation.**

3.1.1: The Can Bus Flow Diagram.



The above flow diagram shows the Main Function in the Can Bus Code, it is in this function that the code will initialise.



The above flow diagram shows the Handle message function for the Can Bus code, it is this function that handles incoming messages.



The previous flow diagram shows the heartbeat message function, it is this function that is implemented when the node using it is sending a normal message.

3.1.2: The Can Bus Code Explanation.

In order for the Can Bus hardware to function correctly a protocol must be implemented using C code, this type of code is then send through RS232 to the Pic16F877A.

**Initialisation.**

In order initialise the Pic16F877A we must include “Header Files” these type of files have information stored in them as to the pin locations, the pin functionality and also pre defined variable values.

#include “can\_template.h”

The “.h” are what is know as “Hex” files, these type of files hold binary information with hexadecimal encoding for the MCU, whilst the “.C” files are I/O files used to describe pin locations.

#define CAN\_BASE\_MESSAGE\_ADDRESS\_1 100

“#define” headers are header constants that can be called anywhere throughout the program.

#DEFINE CAN\_HEARTBEAT\_MESSAGE\_ADDRESS 109

The above “#Define” describes the node used for this project’s heartbeat.

The heartbeat message basically functions like a human heartbeat, it allows the master to know that everything is ok at that particular node.

**Interrupt Service Routine**.

An interrupt service routine is a software routine that hardware calls upon, this routine examines an interrupt and invents a way to cope with it.

**Local Functions Declarations**.

Local Function Declarations are declarations of functions being created within this file itself.

**Main.**

The main function is where the code will initially start acting on variables, it is here that all functions being prototyped in this file will be called upon.

The Bus Code itself has been explained in another report previously, as an engineer in training all I had to do was interface the Robo-realm to the base-board through an RS232 port.

**3.2: A P.C.V.S.**

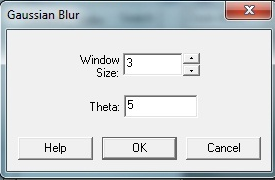
When first assigned this application I had no idea what robo-realm was, and I’ve never heard of it before, so I really did have to learn how to swim, there are tutorials on how to implement this type of design but that was for a different type of robot.

Upon getting used to robo-realm I discovered that it used modules, how I imagine a module is like a function in C code, each individual module had its own unique image processing technique.

3.2.1: Gaussian Blur.

Gaussian Blur is an image processing technique used to smooth an image but preserve its edges.

This type of filter is widely used to strengthen graphics on software, it also reduces image noise and image detail, in order for Gaussian Blur to function correctly we need to pre-define the window size and a value for Theta.



**(Figure 3.2.1.1 is of the properties of the module Gaussian Blur being defined.)**

A higher window size will make an image blurrier, as it effect’s a greater area of the image from the images centre point.



**(Figure 3.2.1.2 is of the image being blurred.)**

3.2.2: Logitech Orbit (pt1).

There are actually two Logitech Orbit modules used in this file, this is because one is controlled by a VB Script program I wrote and the other is user controlled both control the same camera.

The first module will be user controlled.

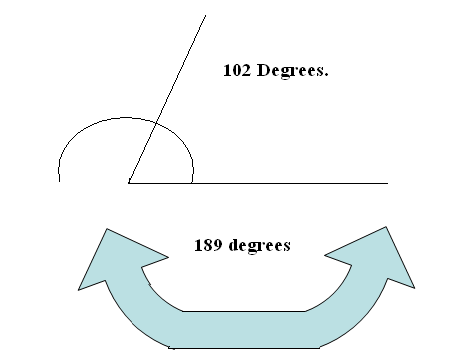
A Logitech Orbit is a web camera that incorporates “pan and tilt”, “pan and tilt” allows us to have a greater view of the objects around us.



**(http://www.byforcet.com/catalog/images/sphe\_G1.jpg?osCsid=97031bc6af0d61ccd23f08ac47fe1224)**

**(Figure 3.2.2.1 is of a Logitech Orbit Sphere MP Camera.)**

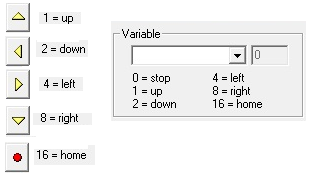
This type of camera allows for a 189 degree view on the Y-axis and a 102 degree view on the X-axis.



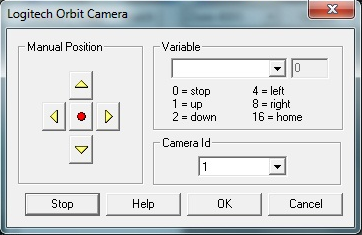
**(Figure 3.2.2.2 is of a Logitech Orbit Sphere MP Camera’s pan and tilt capabilities.)**

The Logitech module itself consists of 5 control switches, a variable drop box and a camera id.

The five control switches are for manual operation, all the user has to do is click the direction they want to look at, whilst the variable box allows the user to introduce a variable in the Vb script module, (Like a global variable in C), the camera ID is just the identification of the camera and is automatically assigned to camera when robo-realm is opened.



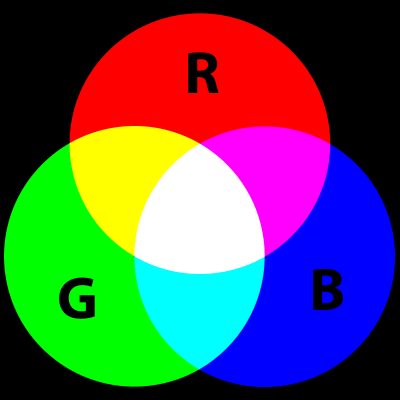
**(Figure 3.2.2.3 is of Logitech Orbit Module variables.)**

****

**(Figure 3.2.2.4 is of a Logitech Orbit Module.)**

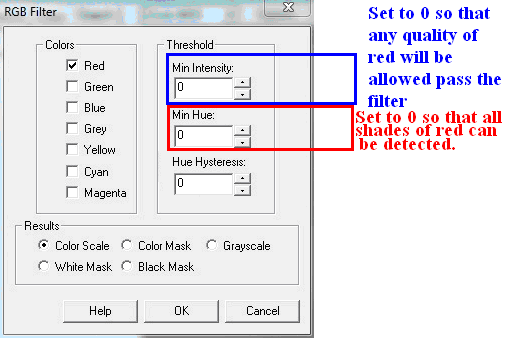
3.2.3: RGB Filter.

The RGB filter is a filter that focus’s its attention towards the original RGB value’s, this means it will focus on the colours selected and reduce all others.



**(Figure 3.2.3.1 is of RGB being combined.)**

**(http://en.wikipedia.org/wiki/File:AdditiveColor.svg)**



**(Figure 3.2.3.2 is of RGB module being implemented.)**

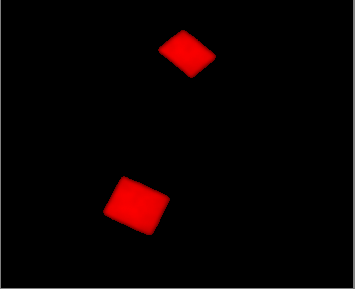
All types of displays have RGB values, weather there plasma display, LED display, Cathode Ray Tube, (CRT) or Liquid Crystal Liquid Display (LCD).

The all incorporate three very small red, green and blue light sources that combine to give an image its colour, these sources are placed together in a triangular format across the screen.

In a computer monitor, or television, each pixel and its colour is represented by a binary value.

What the RGB filter does is that it filters all other values other than red, this means that only red coloured objects will appear on the computer screen.

The above values for hue, and intensity were a mistake, they should be 35, as I found out later in the project that a high intensity yellow can be mistaken as red.

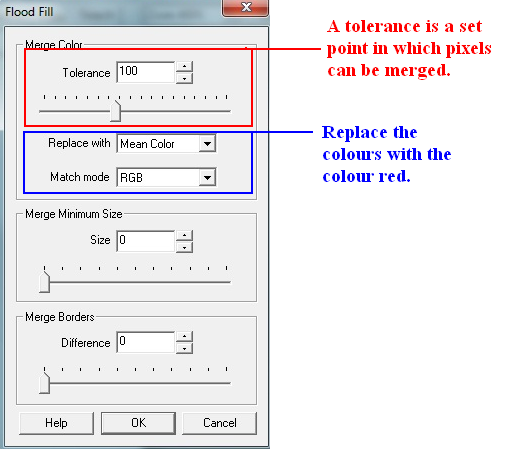
****

**(Figure 3.2.3.3 is of the red squares on screen.)**

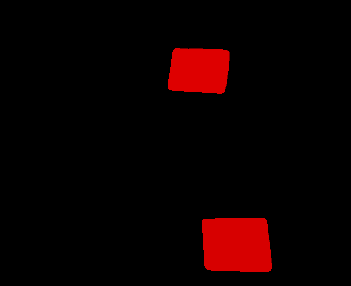
3.2.4: Flood Fill.

Flood fill is a maths algorithm that determines the area linked of a given node in a multi-dimensional display.

The Flood Fill module fills areas of a common colour with a single colour, this is done so that the image can be broken down for further processing.



**(Figure 3.2.4.1 is of the Flood Fill properties being selected.)**



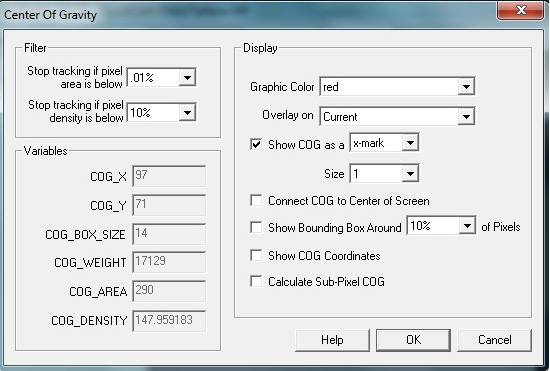
**(Figure 3.2.4.2 is of the Flood Filled Window.)**

3.2.5: Centre of Gravity.

Now that the red objects stand out on the screen we can now track them using the centre of gravity module.

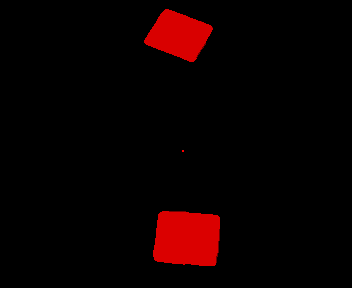
An objects centre of gravity is its average location of weight, to calculate this point varies according to the type of object, but since it’s a square we are tracking the centre of gravity is its height divided by 2.

The COG module does not give us a centre of gravity, but shows us the (x,y) coordinate of the non black pixels centre, since there are only red pixels in the image we can now determine the location of the squares on the screen and if there is any.



**(Figure 3.2.5.1 is of the COG module properties.)**

The location of red pixels has now been determined, the next module VB Script will allow us to act based on that location.

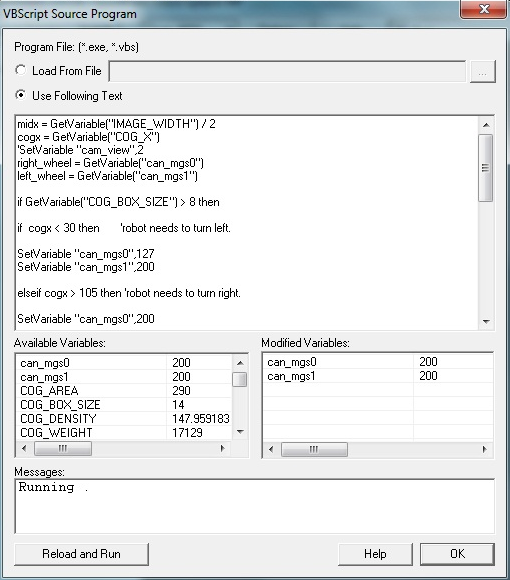


**(Figure 3.2.5.2 is of the COG in the camera window.)**

So now that we know that the average of red pixels is given in an (x,y) location, what happens if there is 2 squares in our screen, what the COG does is it gives the (x,y) location between the two, meaning that as 1 amount of red pixels another will appear,

3.2.6: VB Script.

The VB Script module allows users to implement VB Code into there own unique design, in this case the VB Script is acting on variables inside the screen.



**(Figure 3.2.6.1 is of the some a VB Code being implemented.)**

The code can introduce its own variables as well as make decisions as to what to do with others.

The following is the VB code for the system.

Maxx = GetVariable(“IMAGE\_WIDTH”) ‘these show maximum points on x axis.

Maxy = GetVariable(“IMAGE\_HEIGHT”) ‘these show maximum points on y axis.

Cogx = GetVariable(“COG\_X”)

SetVariable “cam\_view”,2 ‘this variable is for the LOGITECH SPHERE.

SetVariable “middlex”,maxx/2 ‘Creates a new variable called “middlex” with a value of “maxx”/2

SetVariable “middley”,maxy/2 ‘Creates a new variable called “middley” with a value of “midy”/2

ifGetVariable(“COG\_BOX\_SIZE”) > 8 then ‘so the robot wont react to noise.

If cogx <88 then ‘robot needs to turn left.

SetVariable “can\_mgs0”,127 ‘Introduces a new variable called can\_mgs0.

SetVariable “can\_mgs1”,200 ‘Introduces a new variable called can\_mgs1.

Elseif cogx > 230 then ‘robot needs to turn right.

SetVariable “can\_mgs0”,200

SetVariable “can\_mgs1”,127

else ‘if the above conditions are met then go forward.

SetVariable “can\_mgs0”,200

SetVariable “can\_mgs1”,200

end if

else ‘serach

SetTimedVariable “can\_mgs0”, “127”, 2000 ‘this looks left for 2 seconds

SetTimedVariable “can\_mgs1”, “127”, 4000 ‘this looks right for 4 seconds

SetTimedVariable “can\_mgs0”, “127”, 2000 ‘straightens up

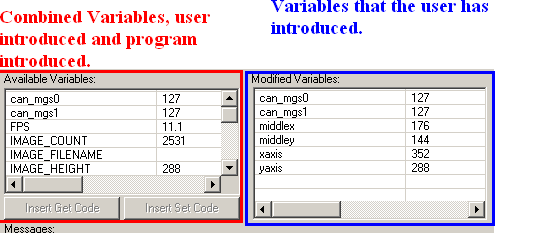
end if

if GetVariable(“COG\_AREA”) = 0 then ‘if there is nothing then stop the robot.

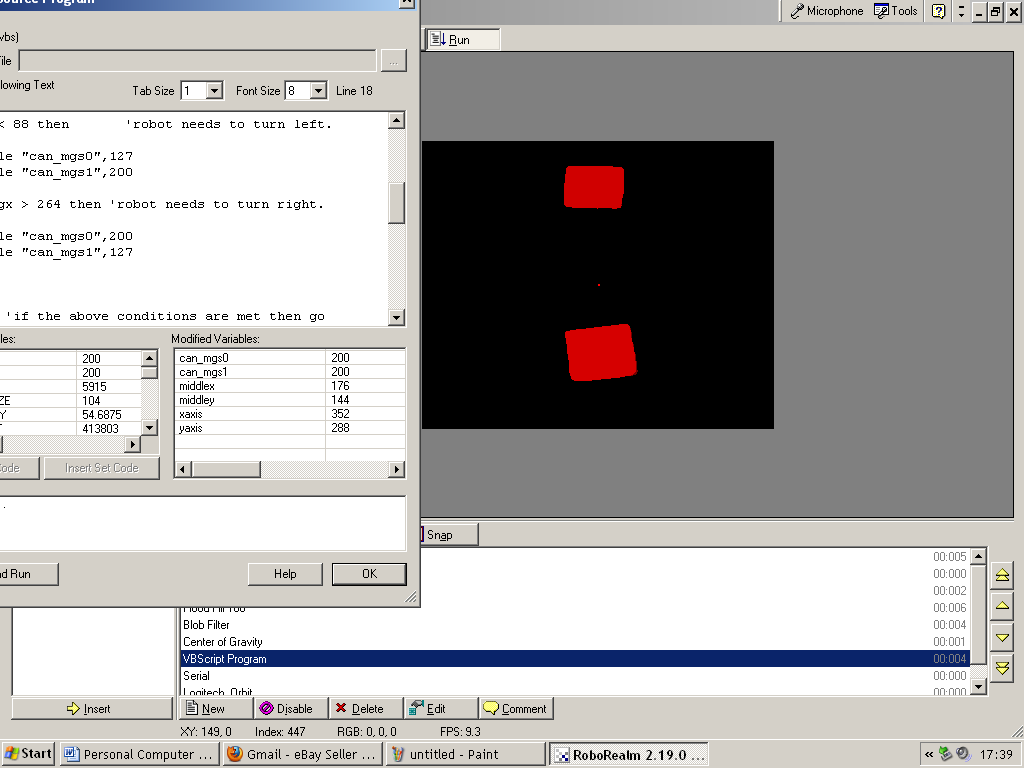
SetVariable “can\_mgs0”,127

SetVariable “can\_mgs1”,127

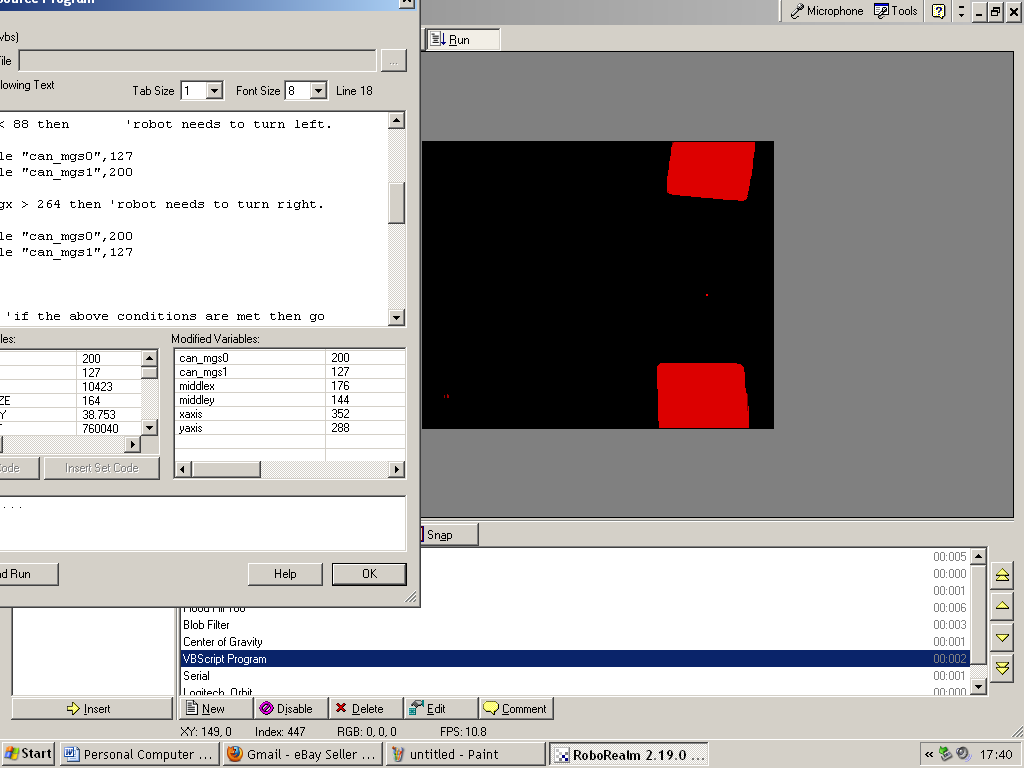
end if



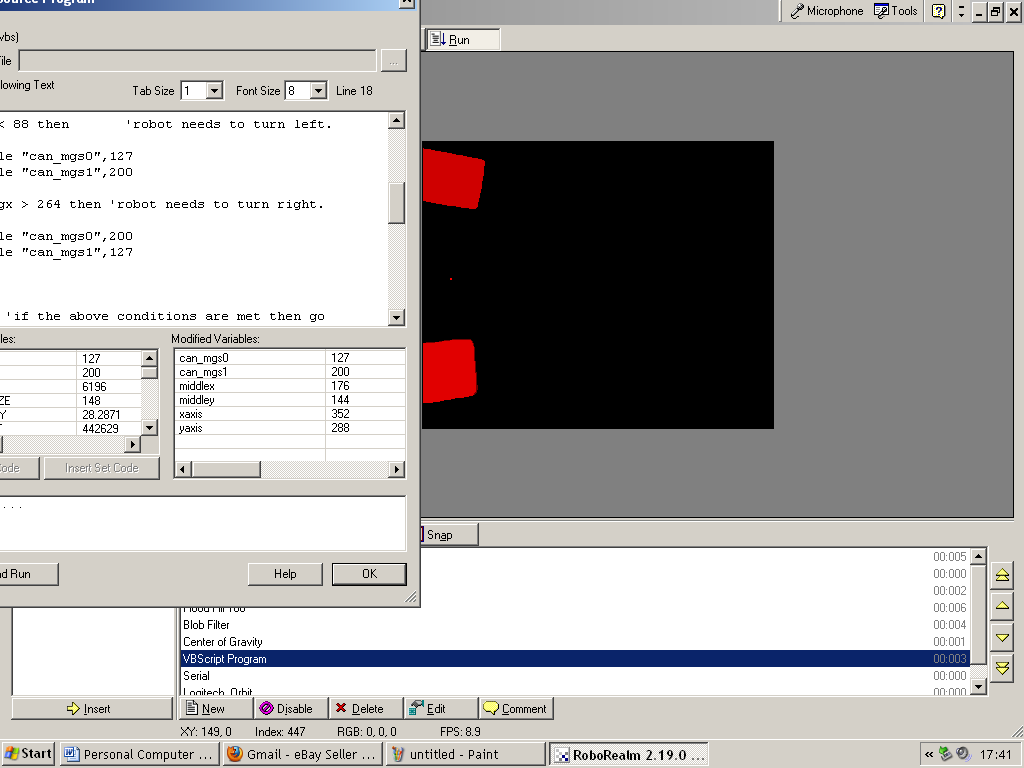
**(Figure 3.2.6.2 is of the some a VB Code being implemented.)**



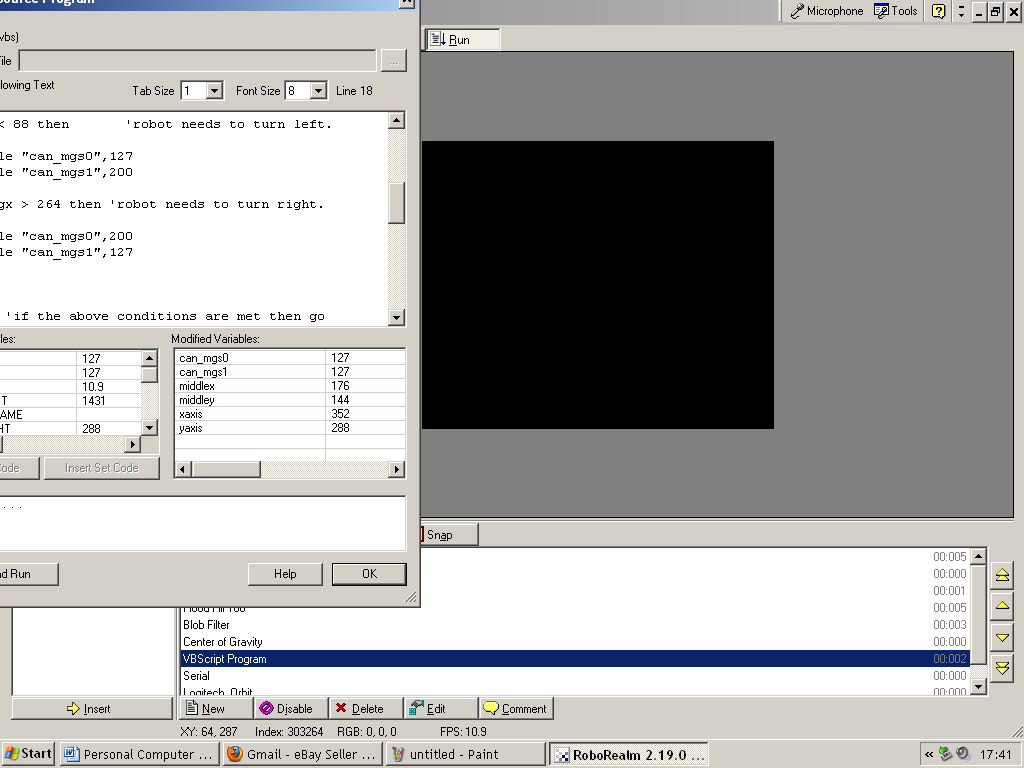
**(Figure 3.2.6.3 is of variables changing due to code written in VB Script.)**



**(Figure 3.2.6.4 is of variables changing due to code written in VB Script.)**



**(Figure 3.2.6.4 is of variables changing due to code written in VB Script.)**

****

**(Figure 3.2.6.5 is of variables changing due to code written in VB Script.)**

3.2.7: Serial Module.

The Serial allows use for the RS232 cable on the computer drive to send signals to the base board, the protocol must be implemented in the serial port for it to communicate with the board.

Variables that have been introduced in the VB Script Module can now be sent to the base board through RS232.

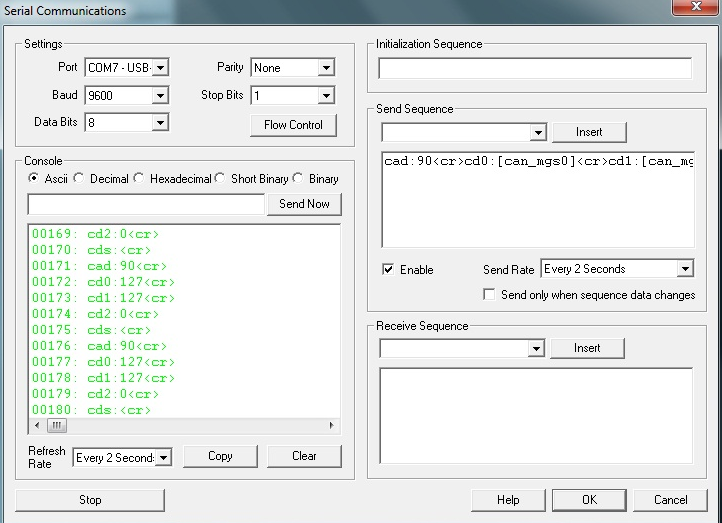
The protocol being implemented is the CAN Bus protocol:

CAD:90<CR>CD1:[CAN\_MGS0]<CR>CD2:[CAN\_MGS1]<CR>CDS:<CR>

90 will be the address that that my board will be communicating to through the can bus.

The variables CAN\_MGS0, and CAN\_MGS1 are variables that have been introduced through VB Script and will change according to the location of the red squares on the screen.

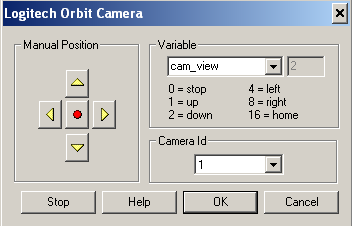
<CR> simply means Carriage Return.



**(Figure 3.2.7.1 is of variables being sent to the base-board via the serial port.)**

3.2.8: Logitech Orbit (pt2).

The Logitech Orbit module was used once again, this is because there is now a variable introduced through VB Script controlling the pan rather than the user, the reason why this module is last is because Roborealm executes its modules serially, rather than concurrent.



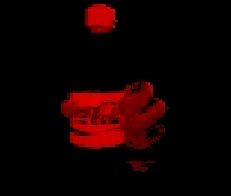
**Chapter 4: Testing.**

**4.1: Testing Modules.**

For testing this application all I needed was my baseboard, as address’s could change, I sent information to myself over the can bus network and displayed it on hyperterminal.

As materials were short I had to implement some of my own.

The below picture shows me testing the RGB filter with a red coca cola label.



**(Figure 4.1.1 is of the RGB Filter Test.)**

When testing certain parts of this project I needed to remember the camera I was using was up-side down, this would mean, that the direction of left is actually right, and right is left, also that the red squares wouldn’t appear on the top of the screen when entering the picture, but they’d appear on the bottom of the screen.



**(Figure 4.1.2 is of the camera looking upside down.)**

At first this confused me, as the robot was turning the opposite way to track the pixels.

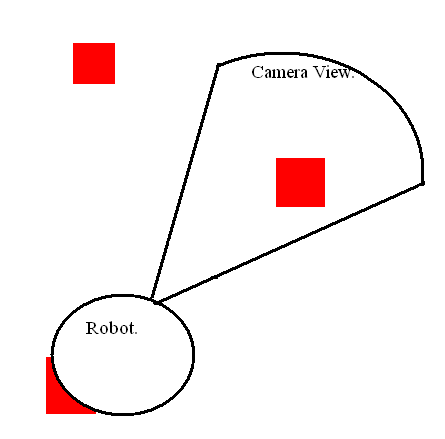
**4.2: Possible Problems.**

One of the biggest problems I can see happening in the future and is that when the path of red squares are spread too far apart that the robot will actually stop dead in its tracks, this is due to a line of code in the VB Script module that eliminates noise:

ifGetVariable(“COG\_BOX\_SIZE”) > 8 then

If the size of the red pixels are less that 8, (Meaning there small from being too far away) the robot will pick them up, but it will be determined that they are just noise rather than a square itself to combat this problem the red squares must be kept close together, but not too close, if they are too close the robot will think that they are connected even though on screen they are not this happens if there are two red squares on either side of the screen parallel to one another, this means that the centre of gravity will be between the two.

Another problem with the red squares is that if they are too far apart the robot might not see them, for example:



**(Figure 4.2.1 is of a possible error.)**

To combat these problems I would suggest keeping the red squares at a distance of about 3 inches apart.

**Chapter 5: Conclusion.**

This project really did test my own ability think and be creative as when I first received my brief I didn’t think I could design it, but as the weeks went by slowly but surly the jigsaw puzzle came together as the work I put into this project began to show, it does have its down falls, (Distance of the squares) but it does what is required by the brief, to improve it I would suggest that maybe a compass could be integrated to the system so that I can tell what way I’m going, also maybe a memory map so that if the robot gets lost or looses the red pixels it can re trace its last movement and then plot a new movement course.