PC-EYE BOT / TORO

Final Report

Michael Reiser

IMDL Fall 1999

December 8, 1999

Table of Contents

Abstract

The purpose of this project is to implement an autonomous platform with all of the advantages and the flexibility of a fully functional Intel-based PC. PC-EYE BOT is a robot that will exhibit vision, and use it to make real-time decisions based on the detection of colors. The robot's basic control will be supplied by a 68HC11, and the image capabilities and higher functionalities will be provided by the on-board Laptop. The robot is able to accomplish many advanced features without the costs traditionally associated with these abilities. The robot is equipped with five sensor systems, each driving its own behavior, and a real-time user interface program. This program controls all behavior arbitration and represents all underlying actions in graphical manner to the user. This robot is to serve as a research platform for complex behavior interactions that cannot be accomplished with lesser computational ability.

Executive Summary

PC-EYE BOT represents a significant effort to construct a fairly ambitious autonomous mobile platform. The robot has an extensive sensor suite, consisting of Infrared Emitter/Detector pairs, Bump Switches, a PC Serial Link, a Joystick, and a Color Camera. The main function of the robot is to use these sensors to navigate and exhibit intelligent behavior. The most notable of these sensors is the camera, since it is a high bandwidth device, and using its data in a real-time application is not a trivial issue. Therefore the robot has been designed as a color detector/follower. The implementation of this behavior is not nearly the most complex behavior possible with the platform, but serves as a realizable goal for a project of this type.

The robot is a two-tiered and two-wheeled rolling platform that houses two separate centers of computation. The lower level functions of sensor data collection and motor control are handled by a 68HC11 running a very short icc11- compiled program. The higher level functions of camera operation, joystick polling, decision making, and user interfacing are handled by a Pentium-based laptop. The Most significant accomplishment in this project is the simple interface between the two systems, and the clearly defined tasks required of each system.

This paper details the results of this project. It includes both the hardware and the software descriptions and implementation details, and offers an explanation of the many features of the robot.

Introduction

I wanted my robot to be able to implement vision. I was continuously discouraged from attempting to solve the problem of vision with too little computational horsepower. So reluctantly, I decided that to perform vision successfully, the obvious solution is to supply ample computation ability. As the name implies, the BOT implements both a PC (personal computer) and an EYE (a camera). The jump to a full-blown PC implementation has many beneficial side-effects. Basically anything that can be done with a PC (and done quite easily) can now be done on the robot. This makes expanding the robot nearly trivial, as demonstrated by the ease with which a camera can be added. Also, for control purposes, I added a Joystick interface at the last second. This addition was also very fast to implement. The camera on the robot is connected to the PC, and the PC is used to make decisions about the robot's behavior, based on the captured images. In this instance of the project, the camera is sensitive to the color red and seeks it out. There is a less powerful microcontroller on the robot that controls the motion of the robot as well as the basic sensors.

Integrated System

The PC-EYE BOT contains several systems:

The 68HC11 microcontroller--this board provides the control logic for:

The Drive system – uses servos to direct the motion of the robot.

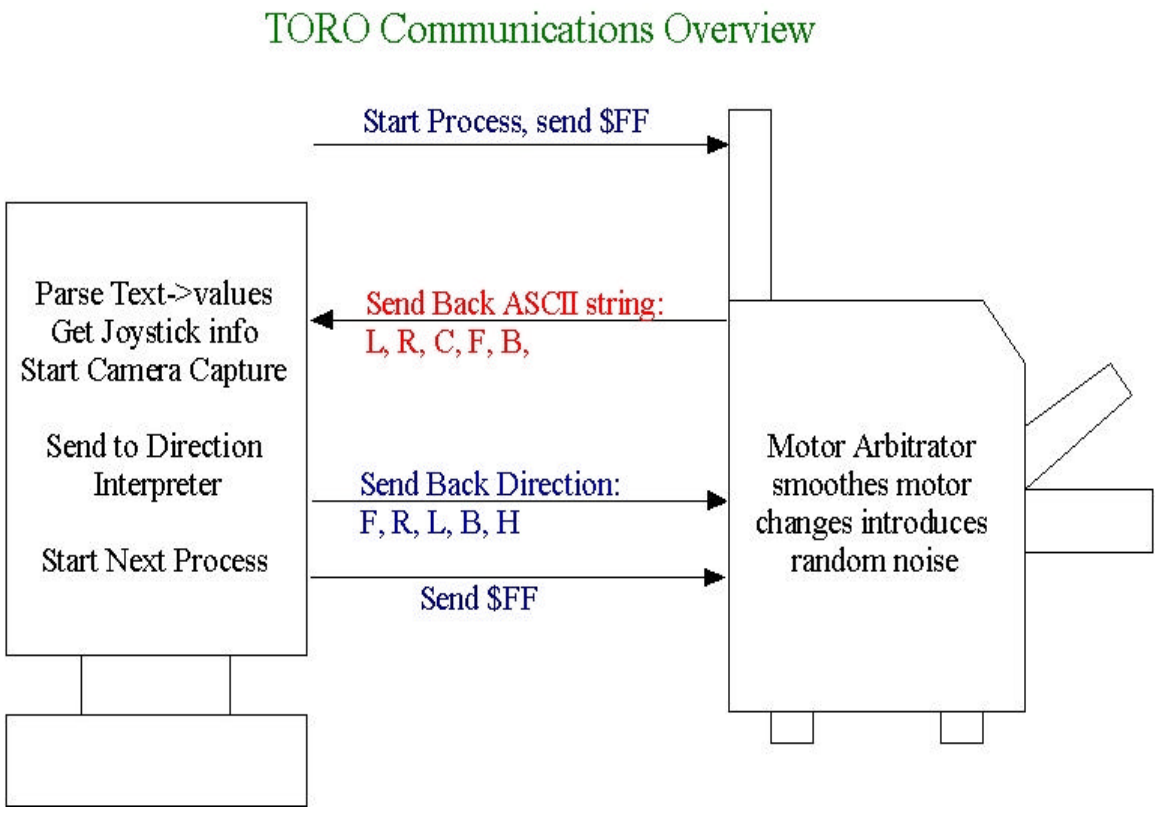
The Low-Level Sensors – uses IR sensors for collision avoidance – detecting the proximity to objets and changing direction. Bump sensors are also used to detect collisions if the IR sensors fail to initiate a change in direction. The vision information from the camera is also used to control collision avoidance.

The PC onboard the robot was initially intended to be a full computer, complete with motherboard and all of the possibilities of a PC's peripheral components. This was to be run off of battery power. Unfortunately due to several compatibility issues, I borrowed a laptop computer and use this in the end result. In my design I use the PC to control:

The Vision System – I use a "web-cam" type camera (color QuickCam 2) and the computer's parallel port interface. I wrote the software that runs on the PC and performs rudimentary image processing on the video data. Initially, I attempted to implement color-following. This behavior was successful, but I did not have time to implement more complex behaviors.

The Communications System – I intended to equip the PC with a wireless Ethernet system so that I can run a TCP/IP network and communicate with my robot. The big advantage of this is that I can remotely have access to nearly all of the PC's capabilities. This goal was dropped in that the laptop provided its own interface, and the wireless Ethernet capability would have been unnecessary.

The most important feature of the communication system is that the PC maintains a constant bi-directional communication with the 68HC11 through a serial connection. The logical separation of functions is then simple. The PC uses the 68HC11's actions and incoming sensor data as well as the data from its own set of sensors to generate decisions (e.g. go left, stop). These decisions are then sent to the 68HC11 via the serial port. Here is the block diagram for this interaction, showing the persistent control loop present on the robot:



Mobile Platform

The platform has been kept very simple. The primary requirement is that there is enough room to hold all of the hardware and sensors as well as provide space for future expansion. The body of PC-EYE BOT is implemented as a tiered rolling platform, slightly elongated (oval–shaped). The size has been chosen as slightly larger than that of the PC motherboard. Although a motherboard was never used, this design was adequate to hold the laptop with only minor modifications. Initially three layers were to be used. The lower layer contains wheels, servos and batteries. Since the weight of the platform was an issue I over-designed and reinforced this portion of the robot. On the final form of the robot, the bottom layer contained “storage room” for: 68’11, communications board, IR emitters, battery pack, servos, wheels, and small bundles of the PC’s connection cables. Above this layer is the control layer – this layer contains the PC laptop, all of the sensors or connections to sensors, and all the cables and connections necessary to support all of the hardware. There was intended to be a layer above this to house the PC additional components – the batteries for the PC, the hard drive, and the camera. This was never needed with the laptop design.

Actuation

This robot autonomously interacts with its environment while continuously collecting video data and making decisions based on that data. But how does the robot get around?

The actuators used on the robot are fully-hacked high torque servos, Tower Hobby brand with a listed torque output of 128 ounce-inches at 6V. Since I am running the servos at a higher voltage (closer to 8 volts) I expect than an even greater maximum torque is generated. I also used two large wheels sold as model airplane wheels. These wheels gave me plenty of trouble. They were big (4.5” diameter) and so were an issue to mount to the servos. Eventually I screwed the servo horn into the wheel, covered the joint in epoxy, and then screwed through the joint to provide a connection mechanism. The screws for the servos were also very hard to find, but eventually I found the correct size screw (2mm) with a hex lock on it which made screwing the wheels in quite easy since I could used an Allen Wrench. I also had dual swivel casters in the rear for support.

Problems with this setup:

Very near to the demo day, when my robot was finally assembled it was obvious that the robot was having a hard time going straight. This had a lot to do with the power level on the batteries, but also with the big slick wheels slipping and the casters misaligning and forcing the platform to favor one side. The wheel slipping I was able to completely fix. I made ”treads” on the wheels with vertical stripes of hot glue, and then o top of this, glued a tacky rubber band around the perimeter of the wheel. I have never seen any robot with wheels that don’t slip at all, and mine did not. The weight being placed directly above the

wheels may have had something to do with it as well. I also removed the second caster and centered the one I was using. These two fixes enabled my robot to maintain a "straight enough" path.

All motor control was done using the icc11 code for motor routines. I use a smoothing function to average desired and previous values of the motor speeds. The algorithm used Is very simple:

- Essentially the 5 directions are interepreted as settin gup a desired spped for each wheel:

  FORWARD: des_speedr = SPEEDR; des_speedl = SPEEDL;

  RIGHT: des_speedr = -SPEEDR; des_speedl = SPEEDL;

  LEFT: des_speedr = SPEEDR; des_speedl = -SPEEDL;

  BACK: des_speedr = -SPEEDR; des_speedl = -SPEEDL;

  STOP: des_speedr = 0; des_speedl = 0;

- The speed is then arrived at by:

  cur_speedr = ((6*cur_speedr)+add_noise(des_speedr))/7;

  cur_speedl = ((6*cur_speedl)+add_noise(des_speedl))/7;

As can be seen, random noise is added at this point. The noise is on the order of about 20% of the total motor speed value, and adds some needed robustness to the motor control. With the noise, even faulty sensor reading will not cause the robot to trap itself

The code for this is listed in Appendix A.

Sensors

PC-EYE BOT contains several types of sensors that allow the machine to intelligently interact with its environment. Since the purpose of the robot is to operate using the video data coming from the camera, there is no physical or tactile interaction with the environment. The most important low level sensors are necessary to provide for obstacle avoidance. The “high level” sensors needed are the robot’s “brain,” the PC, and the robot’s “eyes,” the camera.

The robot contains the following sensor types:

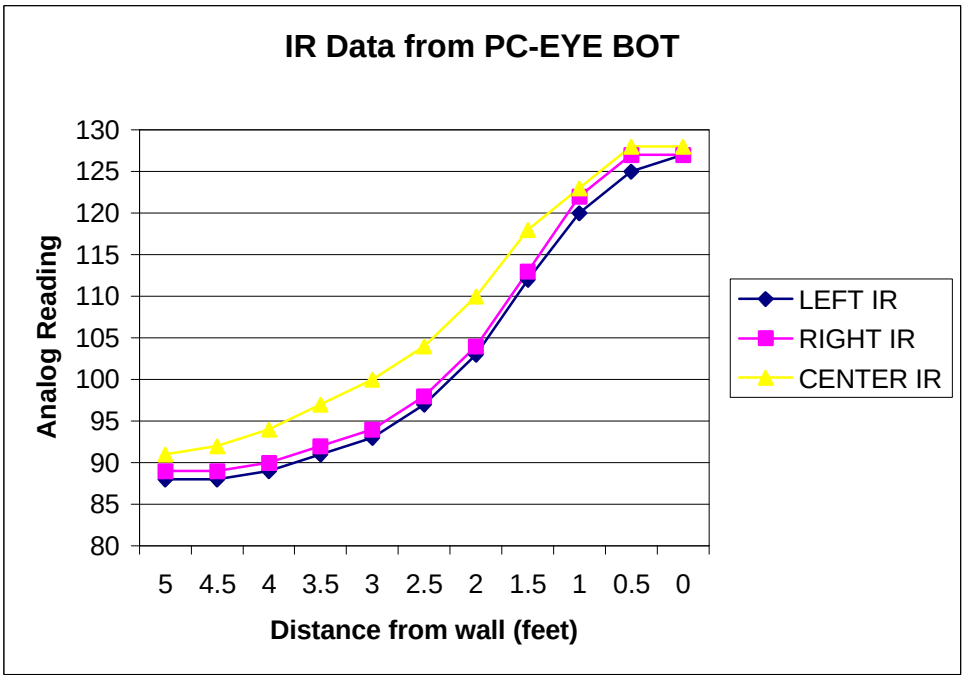
- IR transmitters/receivers
- Bump Switches
- PC system
- Camera connected to the PC
- Joystick for interacting with the PC

Following is an explanation of each system.

IR

PC-EYE BOT contains Three Forward-Facing and one Rear IR Emitter/Receiver pairs. Each emitter is an LED that sends out a modulated 40kHz IR signal which is then detected by the receivers. The receivers and emitters are positioned in such a way that there is no direct line of sight between the pair, so all detected IR must be bouncing off of an object. The IR Receivers used are hacked Sharp sensors. These have been hacked to return an analog voltage that is connected to the A/D port of the 68HC11. The analog

values, when digitized return a value ranging from 88 to 128. A higher number indicates that more IR is being detected by the sensor, and proximity to an object is expected. Here is some sample data from the IR detectors when pointing straight at a wall:



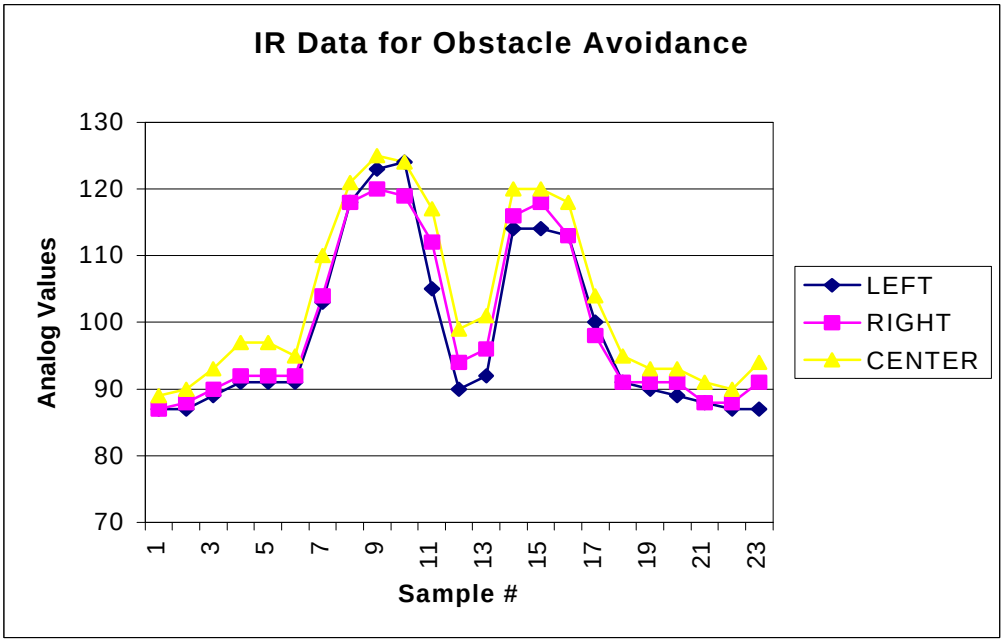
It can be seen that the distance vs. IR values is fairly linear over a range from about 4 feet to 0.5 feet away from the wall.

Obstacle avoidance using IR would then be accomplished by comparing values of IR and moving so as to counteract the highest values. I have found it helpful to set a threshold above which the IR values are “high,” in this case a value of 100 seems appropriate since it indicates a distance of between 2 and 3 feet from the obstacle. For example:

- If all three IR are above the threshold, then reverse direction
- If left or left/center are above the threshold, and left>right, then turn right
- If right or right/center are above the threshold, and right>left, then turn left

The code implementing this algorithm is in Appendix B.

Here is a data set showing that the above behaviors will implement obstacle avoidance:

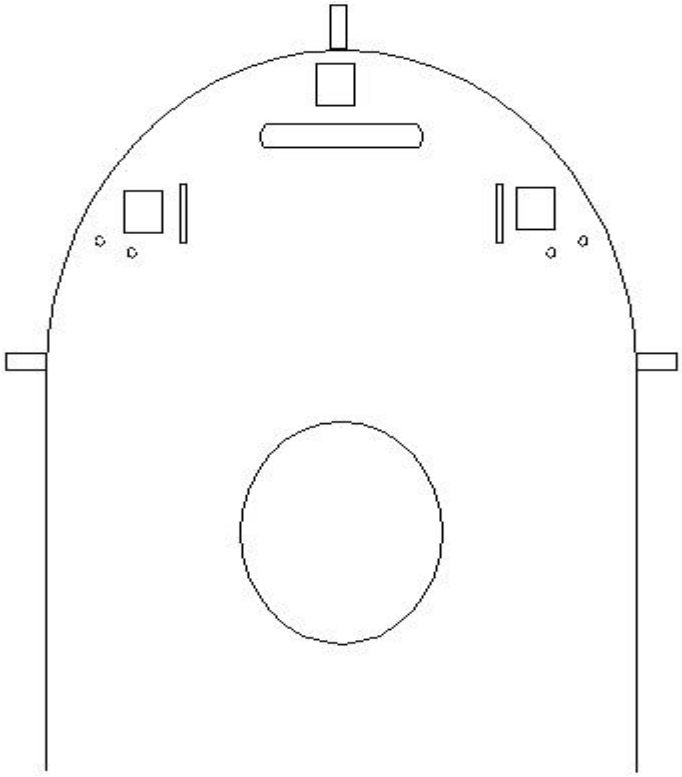


Here the first bump shows an obstacle on the left and the second shown an obstacle right.

BUMP SENSORS

In the event that the IR data should not be properly handled, or some conditions occur that render this data inaccurate (i.e. dark corners) a system is implemented to back-up the IR. Bump sensors are used to detect a direct collision with an object and to assist the robot in getting free of that object. The sensors are just switches that are connected via a resistor divider network to the A/D ports of the 68HC11. By properly selecting the values of the resistors the robot can determine the side at which the collision occurred, and move away from that side. The bump sensors are surrounded by a cantilevered wooden bump ring that has a small range of free movement. The bumpers work by pushing into the switch when contact is made and then using the switches springiness to bounce back.

This bump system is one of the most successful features of the robot, and is more than capable of distributing the collision to one (or more) of the switches. There are six switches on the robot, three in the front and three in the back, spaced out so that a sensor is at each “corner” and at the direct front and direct rear of the platform. Here is an AutoCad sketch of the positioning of the sensors:



In this image the squares represent the IR receivers and the rectangles are the sites for the bump switches.

Here is the data gathered for the values returned by the A/D unit for the bump switches. The font and rear network have been wired in the same way so the values are the same:

Switch(es) Pressed	Lowest A/D Value	Highest A/D Value
Left	41	44
Right	127	129
Center	76	79
Left, Center	100	102
Right, Center	149	153

COMPUTER

The most complex “sensor” used on the robot is the PC. The PC used is not a 486 desktop system as initially planned--but rather a laptop. The specifications for the laptop:

CPU: Pentium 120 MHz

Memory: 24 MB RAM

Hard Drive capacity: ~800 MB

Operating System: Windows 98

In addition all necessary development tools (MS visual Studio and ICC11) have been loaded onto the laptop so that the development of the software can occur on the robot itself. The laptop is placed directly on top of the robot as seen below:

The laptop is connected to the 68HC11 through a serial connection to the communication board that provides for translation from RS232 values to TTL levels.

THE CAMERA

The camera used is a Parallel Port connected Web Camera capable of resolutions between 160×120 pixels to 640×480 pixels. The camera implements automatic gain control and is capable of 4 bit to 24 bit colors. The current camera performance is about 12fps, with poor detail in low light. The only solution to the lighting problem is to provide a light source on the robot. The camera driver uses the Video For Windows Standard and is thus programmable from within a windows environment. The camera setting used during the demo are: 160×120, with 256 colors and automatic lighting and hue settings. The camera data is used to perform image processing. A control called "ezVidCap.ocx" is used to handle the acquisition of the images.

The images are then reduced into a 40×30 image and pixel-by-pixel image processing is done on the smaller image. The processing I use to find red is a combination of thresholding filters. The filter used is: "If (bytRed > 125 And bytGreen < 100 And bytBlue < 125)" then this is very likely to be a red pixel, add to count of red pixels. It was also helpful to filter out dark colors since they were affecting the algorithm, this is done by "If bytRed + bytGreen + bytBlue < 150" then ignore this value since it is very dark. Then the count of red pixels in each of the 40 columns is totaled and flattened into 3 values. These values represent the amount of red in the Right, Center, and Left part of the robot's filed of view. A threshold value of 6 pixels per screen third is used, after the

threshold has been met, the desired direction is simply the screen portion with the most red pixels.

THE JOYSTICK

The joystick used is a Microsoft Sidewinder Pro. This joystick is capable of 3D control and has nearly a dozen buttons on it. I implemented the joystick as a 2D, 4 button joystick, and was able to receive x and y coordinates and button presses from the windows API. Since the range of values for each axis is large—it is a long value from 0 to ~64k—I only used the very extremes of the joystick's position for a value.

Behaviors

As illustrated in the block diagram in the Integrated Systems section, the robot operates in a persistent control loop. The interface between the PC and the 68HC11 is as follows: The PC requests from the 68HC11 its string of sensor values. The OC then uses this information and its own sensors to come up with a direction (do left, right, straight, back, etc…) to send the 68HC11. All of these communications are just ASCII values sent between the two. The directions sent to the 68HC11 are determined based on both IR data and picture data, and the bump sensor data. The PC is running a windows program to handle this operation.

The robot has 4 main sensor systems: camera, joystick, IR and bump sensors. Each system drives its own behavior. For each time period where all sensor data is collected, there is a function that interprets each sensor's data and generates a direction decision for that sensor. Each sensor has a default direction of either Halt or Forward. There is then a direction arbitrator that generates an overall direction decision by maintaining a hierarchy:

Joystick

Camera

Bump

IR

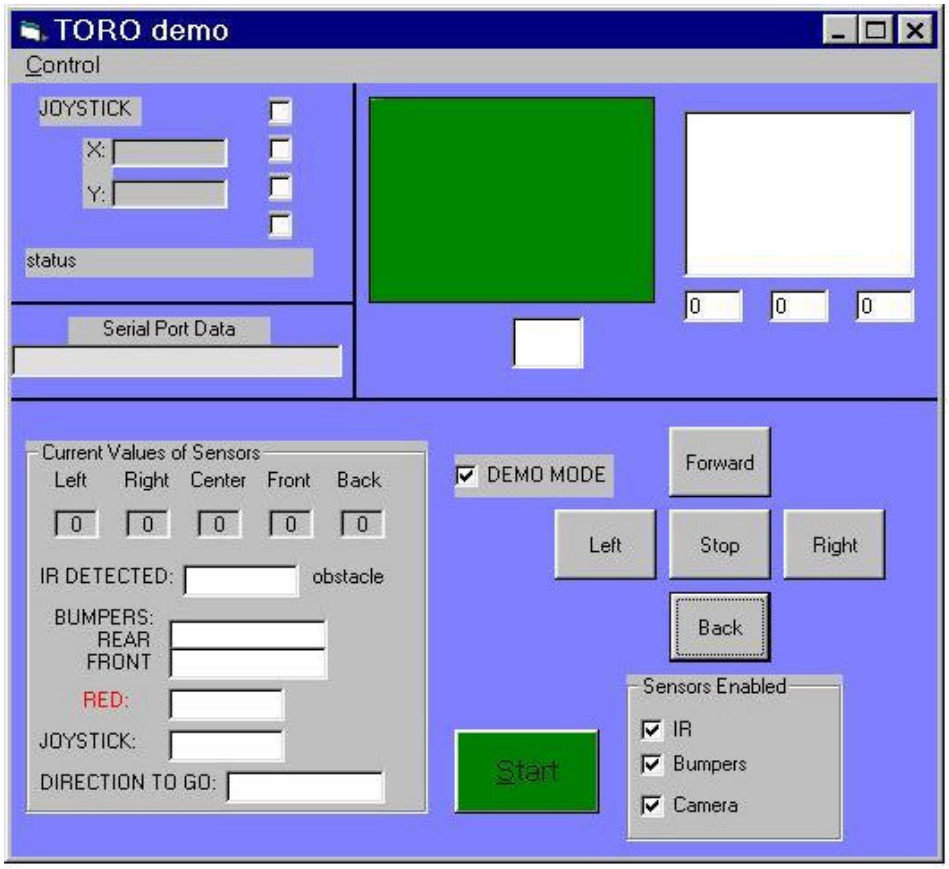
This is not a strict hierarchy in that there are exceptions to this system. For example if Camera is seeing a lot of red and the bump sensor is pressed then the assumption is that the target has been hit and the robot stops. The default direction is used to see if the

sensor requires attention. If the sensor is passing its default direction then it is ignored and the next relevant sensor in the hierarchy is used to determine the direction.

Since this behavior architecture is used it makes it very simple to disable a sensor, in which case it is ignored. Each sensor can be disabled from the main program or via the joystick. Unplugging the joystick will disable it as a sensor. The API provides for detecting that the joystick is unplugged and the program uses this to ignore the joystick. The simplest method of seeing each behavior is disabling the other sensors. For example: to demo the robot, I disabled all sensors but the camera. The robot then remains idle until the camera sensor calls for attention, when it has found red, at which point the sensor's behavior assumes control of the overall system behavior. Because of this enabling and disabling of behaviors, there are actually 4! behaviors.

Experimental Layout and Results

All of the meaningful experimentation was carried out with the computer program application running. Here is a screen shot of the program:



As can be seen this program shows everything that is occurring with all of the sensors so it is very easy to see what is happening and correct it in real-time. The only necessary tests that were conducted were for performance. The overall performance of this application in controlling and & communicating with the robot is found by running the control loop 50 times and timing its duration.

The results:

In Demo Mode: 50 times in 32 seconds – 1.56 Hz

In standard Mode: 50 times in 15 seconds – 3.33 Hz

Conclusion

I think that with the implementation of TORO, I accomplished many things. I was able to perform near real-time image processing for navigation purposes with an unusual set of tools: a $40 camera, a crumbling laptop, and a Visual Basic control program. The biggest limitation of the robot is the performance of the control program. There are many other avenues that can be explored for tightening the code and the overall performance while still keeping the framework currently used. I believe that with no great adjustments the 3.33 Hz rate could be improved to about 5 Hz. By using other methods of image processing I believe this rate could nearly be doubled. Is this important? It is so that true real-time navigation decisions can be made, and the robot's movements would be very smooth. As the robot is currently implemented, he may have a red object in his field of view off to the right, he turns right, and by the time he takes the next snap shot he has passed the red object. Compensating for this effect greatly complicates the program. The other limitation is the narrow field of view of the camera. This also leads to the same pattern of behavior described above.

Perhaps the biggest disappointment I had was not enough time to implement intelligent behaviors. One thing I learned from programming behaviors is that the jump from 1 to 2 sensors does not result in a doubling of code, but actually a quadrupling of code since the interactions between the data must be accounted for. For this reason it is very difficult to have consistent intelligent behavior from so many sensor systems without programming for very complex interactions. Because of this I feel my platform is seriously under-utilized and could be used for much more interesting work.

The changes I made to my wheels and caster right before the demo were very significant in the success of the robot. I was able to implement a virtually slip-free platform. Although it is just a nice thing to have on my robot, it would be wonderful for someone implementing navigation or anything else where a shaft encoder is used.

If I were to redo my project, I would:

- Build a much sturdier platform, and have it shaped to nicely accommodate the hardware I was using
- Keep the platform very modular. I would like to implement a small cube with the processor, batteries, etc. inside and have a common connector on the outside of the cube. This little module could then be easily taken in and out of the robot.
- Do more research first—always look for code someone else ha written before writing my own. It would have saved me considerable time.
- Focus more time on intelligent behavior. I would have been able to do this if I had not encountered so many technical difficulties with my PC platform earlier in the semester.

Documentation

Robotics Reference Books:

Fred Martin, The 6.270 Robot Builder's Guide, MIT Media Lab, Cambridge, MA, 1992

Joseph Jones & Anita Flynn, Mobile Robots: Inspiration to Implementation, A.K. Peters Publishers, Wellesley, MA, 1993

Visual Basic Books:

Mark Pruett, The Black Art of Visual Basic Game Programming, The Waite Group, 1993

Steven Holzner, Visual Basic 6 Black Book, The Coriolis Group, 1998

Video For Windows Programming:

http://ej.bantz.com, http://www.microsoft.com, http://i.am/shrinkwrapvb

Joystick Prgramming:

http://www.microsoft.com

IMDL references:

AutoCAD Tutorial, ICC11 Reference, Programming Behaviors, etc.

Appendix A – ICC11 Code

```
#include <tjpbase.h>
#include <stdio.h>
/*********************** End of Includes ***************************/


/*global variables*/
 int direction;
 int cur_speedr,cur_speedl,des_speedr,des_speedl;

/* Constants: */

#define RBUMPER      analog(7)
#define FBUMPER  analog(5)
#define MID_IR       analog(6)
#define SPEEDR 50
#define SPEEDL 60


#define FORWARD 0x38
#define RIGHT 0x36
#define LEFT 0x34
#define BACK 0x32
#define STOP 0x35
#define SYNCH 0xFF

void motor_arb(void);
void send_serial(void);
int add_noise(int);

/*global variables*/
 int direction;
 int cur_speedr,cur_speedl,des_speedr,des_speedl;

/***************************** Main *********************************/
void main(void)
{

       int command;

  init_analog();
  init_clocktjp();
  init_serial();
  init_motortjp();

       *(unsigned char *)(0x7000) = 0xFF;
       /* Instead of IRE_ON;     turn on IR emitters */

       cur_speedr = 0;
       cur_speedl = 0;


       while(1)
  {

    command = getchar();

    if (command == SYNCH) send_serial();
    else
       {
              direction = command;
```

```
              motor_arb();
       }


       wait(2);

  }/*end while(1)*/

}

void send_serial(void)
{
       printf("L:%d",LEFT_IR);
     printf(",R:%d",RIGHT_IR);
     printf(",C:%d",MID_IR);
     printf(",F:%d",FBUMPER);
     printf(",B:%d,\n",RBUMPER);
}


void motor_arb(void)
{
  switch (direction)/*Set up desired speeds */
      {
              case FORWARD: des_speedr = SPEEDR; des_speedl = SPEEDL; break;
              case RIGHT: des_speedr = -SPEEDR; des_speedl = SPEEDL; break;
              case LEFT: des_speedr = SPEEDR; des_speedl = -SPEEDL; break;
              case BACK: des_speedr = -SPEEDR; des_speedl = -SPEEDL; break;
              case STOP: des_speedr = 0; des_speedl = 0; break;
       }

       cur_speedr = ((6*cur_speedr)+add_noise(des_speedr))/7;
       cur_speedl = ((6*cur_speedl)+add_noise(des_speedl))/7;

       if (direction == STOP) {
              motorp(RIGHT_MOTOR, 0);
       motorp(LEFT_MOTOR, 0);
    }
    else {
       motorp(RIGHT_MOTOR, cur_speedr);
           motorp(LEFT_MOTOR, cur_speedl);
    }
}

int add_noise(int num)
{
 int i;
 unsigned rand;

 rand = TCNT;
 i=(rand % 12);

 if (rand & 0x0001)
 { return num + i;  }
 else
 { return num - i; }

}
```

Appendix B – Visual Basic Code

FORM 1 CODE

```
Option Explicit

Dim retVal As Variant
Public lcv As Integer
'Public ReceivedData As Boolean

Const JoyXLeft = 10000
Const JoyXRight = 50000
Const JoyYForward = 10000
Const JoyYBack = 50000

Private Sub Right_Click()
'To go right send a 6
MSComm1.Output = Chr(54)
End Sub

Private Sub Back_Click()
' To go back send a 2
MSComm1.Output = Chr(50)
End Sub

Private Sub Forward_Click()
' To go forward send an 8
MSComm1.Output = Chr(56)
End Sub

Private Sub Left_Click()
' To go left send a 4
MSComm1.Output = Chr(52)
End Sub
Private Sub Stop_Click()
' To stop send a 5
MSComm1.Output = Chr(53)
End Sub

Private Sub Form_Load()
   ' Get capabilities of joystick1
   'rc = joyGetDevCaps(JOYSTICKID1, caps, Len(caps))

   'Set xaxis = axis(0)
   'Set yaxis = axis(1)

   ' Start the timer
End Sub

Private Sub Form_Unload(Cancel As Integer)
retVal = PlaySound("c:\VBCODE\files\goodbye.wav", 0&, &H20000)
End Sub

Private Sub mnuExit_Click()
Unload Me
```

```
End Sub

Private Sub mnuFormat_Click()
VidWin.ShowDlgVideoFormat
End Sub
Private Sub mnuSource_Click()
VidWin.ShowDlgVideoSource
End Sub


Private Sub MSComm1_OnComm()
    Dim dir As Integer

    Select Case MSComm1.CommEvent
        Case comEvReceive
            SensorText(0).Text = MSComm1.Input
            GetLastIRValues
            GetJSVals

            dir = DIRInterp(IRInterp, BumpFInterp, BumpRInterp,
CamInterp, JoyInterp)

            If (MSComm1.PortOpen = True) And (ShowVid.Value = 0) Then
            MSComm1.Output = Chr(dir)
            End If

            StartNext

            lcv = lcv + 1
            If lcv = 50 Then Beep
    End Select


End Sub


Private Sub sensors_Click(Index As Integer)
If sensors(Index).Value = 1 Then
Select Case Index
    Case 0:
    retVal = PlaySound("c:\VBCODE\files\iron.wav", 0&, &H20000)
    Case 1:
    retVal = PlaySound("c:\VBCODE\files\bumpon.wav", 0&, &H20000)
    Case 2:
    retVal = PlaySound("c:\VBCODE\files\camon.wav", 0&, &H20000)
End Select
Else
Select Case Index
    Case 0:
    retVal = PlaySound("c:\VBCODE\files\iroff.wav", 0&, &H20000)
    Case 1:
    retVal = PlaySound("c:\VBCODE\files\bumpoff.wav", 0&, &H20000)
    Case 2:
    retVal = PlaySound("c:\VBCODE\files\camoff.wav", 0&, &H20000)
End Select
End If
End Sub
```

```
Private Sub ShowVid_Click()
lcv = 0
If (ShowVid.Value = 0) Then
    Picture2.Visible = False
Else
    Picture2.Visible = True
    MSComm1.Output = Chr(53)
End If
End Sub

Public Sub Start_Command_Click()
If MSComm1.PortOpen Then

    MSComm1.Output = Chr(53)
    MSComm1.PortOpen = False
    retVal = PlaySound("c:\VBCODE\files\DEACTIVE.wav", 0&, &H20000)
    Start_Command.Caption = "&Start"
    Start_Command.BackColor = RGB(0, 255, 0)
Else
    MSComm1.InBufferCount = 0
    lcv = 0
    retVal = PlaySound("c:\VBCODE\files\R2.wav", 0&, &H20000)
    ' Turn on the port
    MSComm1.PortOpen = True
    MSComm1.Output = Chr(255)
    Start_Command.Caption = "S&top"
    Start_Command.BackColor = RGB(255, 0, 0)
End If

End Sub
```

MODULE CODE – JOYSTICK.BAS

```
Option Explicit

Public Const MAXPNAMELEN = 32

' The JOYINFOEX user-defined type contains extended information about the joystick
position,
' point-of-view position, and button state.
Type JOYINFOEX
   dwSize As Long                    ' size of structure
   dwFlags As Long                   ' flags to indicate what to return
   dwXpos As Long                    ' x position
   dwYpos As Long                    ' y position
   dwZpos As Long                    ' z position
   dwRpos As Long                    ' rudder/4th axis position
   dwUpos As Long                    ' 5th axis position
   dwVpos As Long                    ' 6th axis position
   dwButtons As Long                 ' button states
   dwButtonNumber As Long            ' current button number pressed
   dwPOV As Long                     ' point of view state
   dwReserved1 As Long               ' reserved for communication between winmm driver
   dwReserved2 As Long               ' reserved for future expansion
End Type

' The JOYCAPS user-defined type contains information about the joystick capabilities
Type JOYCAPS
```

```
   wMid As Integer                     ' Manufacturer identifier of the device driver for
the MIDI output device
                                       ' For a list of identifiers, see the Manufacturer
Indentifier topic in the
                                       ' Multimedia Reference of the Platform SDK.

   wPid As Integer                     ' Product Identifier Product of the MIDI output
device. For a list of
                                       ' product identifiers, see the Product Identifiers
topic in the Multimedia
                                       ' Reference of the Platform SDK.
   szPname As String * MAXPNAMELEN     ' Null-terminated string containing the joystick
product name
   wXmin As Long                       ' Minimum X-coordinate.
   wXmax As Long                       ' Maximum X-coordinate.
   wYmin As Long                       ' Minimum Y-coordinate
   wYmax As Long                       ' Maximum Y-coordinate
   wZmin As Long                       ' Minimum Z-coordinate
   wZmax As Long                       ' Maximum Z-coordinate
   wNumButtons As Long                 ' Number of joystick buttons
   wPeriodMin As Long                  ' Smallest polling frequency supported when
captured by the joySetCapture function.
   wPeriodMax As Long                  ' Largest polling frequency supported when
captured by the joySetCapture function.
   wRmin As Long                       ' Minimum rudder value. The rudder is a fourth
axis of movement.
   wRmax As Long                       ' Maximum rudder value. The rudder is a fourth
axis of movement.
   wUmin As Long                       ' Minimum u-coordinate (fifth axis) values.
   wUmax As Long                       ' Maximum u-coordinate (fifth axis) values.
   wVmin As Long                       ' Minimum v-coordinate (sixth axis) values.
   wVmax As Long                       ' Maximum v-coordinate (sixth axis) values.
   wCaps As Long                       ' Joystick capabilities as defined by the
following flags
                                       '     JOYCAPS_HASZ-     Joystick has z-coordinate
information.
                                       '     JOYCAPS_HASR-     Joystick has rudder
(fourth axis) information.
                                       '     JOYCAPS_HASU-     Joystick has u-coordinate
(fifth axis) information.
                                       '     JOYCAPS_HASV-     Joystick has v-coordinate
(sixth axis) information.
                                       '     JOYCAPS_HASPOV-   Joystick has point-of-view
information.
                                       '     JOYCAPS_POV4DIR-  Joystick point-of-view
supports discrete values (centered, forward, backward, left, and right).
                                       '     JOYCAPS_POVCTS Joystick point-of-view
supports continuous degree bearings.
   wMaxAxes As Long                    ' Maximum number of axes supported by the
joystick.
   wNumAxes As Long                    ' Number of axes currently in use by the joystick.
   wMaxButtons As Long                 ' Maximum number of buttons supported by the
joystick.
   szRegKey As String * MAXPNAMELEN    ' String containing the registry key for the
joystick.
End Type

Declare Function joyGetPosEx Lib "winmm.dll" (ByVal uJoyID As Long, pji As JOYINFOEX) As
Long
' This function queries a joystick for its position and button status. The function
' requires the following parameters;
'     uJoyID-  integer identifying the joystick to be queried. Use the constants
'              JOYSTICKID1 or JOYSTICKID2 for this value.
'     pji-     user-defined type variable that stores extended position information
'              and button status of the joystick. The information returned from
'              this function depends on the flags you specify in dwFlags member of
'              the user-defined type variable.
'
' The function returns the constant JOYERR_NOERROR if successful or one of the
' following error values:
'     MMSYSERR_NODRIVER-      The joystick driver is not present.
```

```
'     MMSYSERR_INVALPARAM-    An invalid parameter was passed.
'     MMSYSERR_BADDEVICEID-   The specified joystick identifier is invalid.
'     JOYERR_UNPLUGGED-       The specified joystick is not connected to the system.

Declare Function joyGetDevCaps Lib "winmm.dll" Alias "joyGetDevCapsA" (ByVal id As Long,
lpCaps As JOYCAPS, ByVal uSize As Long) As Long
' This function queries a joystick to determine its capabilities. The function requires
' the following parameters:
'     uJoyID-  integer identifying the joystick to be queried. Use the contstants
'              JOYSTICKID1 or JOYSTICKID2 for this value.
'     pjc-     user-defined type variable that stores the capabilities of the joystick.
'     cbjc-    Size, in bytes, of the pjc variable. Use the Len function for this value.
' The function returns the constant JOYERR_NOERROR if a joystick is present or one of
' the following error values:
'     MMSYSERR_NODRIVER-   The joystick driver is not present.
'     MMSYSERR_INVALPARAM- An invalid parameter was passed.


Public Const JOYSTICKID1 = 0
Public Const JOYSTICKID2 = 1
Public Const JOY_RETURNBUTTONS = &H80&
Public Const JOY_RETURNCENTERED = &H400&
Public Const JOY_RETURNPOV = &H40&
Public Const JOY_RETURNR = &H8&
Public Const JOY_RETURNU = &H10
Public Const JOY_RETURNV = &H20
Public Const JOY_RETURNX = &H1&
Public Const JOY_RETURNY = &H2&
Public Const JOY_RETURNZ = &H4&
Public Const JOY_RETURNALL = (JOY_RETURNX Or JOY_RETURNY Or JOY_RETURNZ Or JOY_RETURNR Or
JOY_RETURNU Or JOY_RETURNV Or JOY_RETURNPOV Or JOY_RETURNBUTTONS)
Public Const JOYCAPS_HASZ = &H1&
Public Const JOYCAPS_HASR = &H2&
Public Const JOYCAPS_HASU = &H4&
Public Const JOYCAPS_HASV = &H8&
Public Const JOYCAPS_HASPOV = &H10&
Public Const JOYCAPS_POV4DIR = &H20&
Public Const JOYCAPS_POVCTS = &H40&
Public Const JOYERR_BASE = 160
Public Const JOYERR_UNPLUGGED = (JOYERR_BASE + 7)
```

MODULE CODE – PCEYEHELP.BAS

```
Public SensorValues(4) As Integer
Public R, L, C As Integer
Public CurX, CurY As Long
Const AVOID_THRESHOLD = 100
Const CAM_THRESHOLD = 6
Const LBVal = 43
Const RBVal = 128
Const CBVal = 78
Const LCBVal = 101
Const RCBVal = 150

Const Forward = 56
Const Right = 54
Const Left = 52
Const Back = 50
Const HALT = 53
Const intUpperBoundX = 40
Const intUpperBoundY = 30

'for joystick
Dim ji As JOYINFOEX     ' joystick state buffer
```

```
Dim caps As JOYCAPS     ' joystick capabilities
Dim rc As Long          ' return code
Dim i As Long           ' index
Dim mask As Long        ' bitmask
Dim xaxis As Label      ' x-axis control
Dim yaxis As Label      ' y-axis control

Const JoyXLeft = 10000
Const JoyXRight = 50000
Const JoyYForward = 10000
Const JoyYBack = 50000

Declare Function PlaySound Lib "winmm.dll" Alias "PlaySoundA" (ByVal lpszName
As String, ByVal hModule As Long, ByVal dwFlags As Long) As Long


Function DIRInterp(IRval As Integer, FBVal As Integer, RBVal As Integer, CAMval
As Integer, JSVal As Integer) As Integer
    Form1.CAMDIR.BackColor = &HFFFFFF
    Form1.FBUMPDIR.BackColor = &HFFFFFF
    Form1.RBUMPDIR.BackColor = &HFFFFFF
    Form1.IROBS.BackColor = &HFFFFFF
    Form1.JOYDIR.BackColor = &HFFFFFF

If (JSVal <> 56) Then
    DIRInterp = JSVal
    Form1.JOYDIR.BackColor = &H80FF&
ElseIf ((Form1.sensors(2).Value = 1) And (CAMval <> 53) And (L + R + C > 40)
And (FBVal < 56)) Then
    DIRInterp = HALT
    retVal = PlaySound("c:\VBCODE\files\beef.wav", 0&, &H20000)

ElseIf (Form1.sensors(2).Value = 1) And (CAMval <> 53) Then
    If (Form1.MSComm1.PortOpen = True) And (Form1.ShowVid.Value = 0) Then
            Form1.MSComm1.Output = Chr(CAMval)
    End If
    WaitTime (20000)
    DIRInterp = Forward
    Form1.CAMDIR.BackColor = &H80FF&

ElseIf ((Form1.sensors(1).Value = 1) And (FBVal < 56)) Then
    If (Form1.MSComm1.PortOpen = True) And (Form1.ShowVid.Value = 0) Then
            Form1.MSComm1.Output = Chr(FBVal)
    End If
    WaitTime (50000)
    DIRInterp = FBVal
    Form1.FBUMPDIR.BackColor = &H80FF&

ElseIf ((Form1.sensors(1).Value = 1) And (RBVal = 56)) Then
    DIRInterp = RBVal
    Form1.RBUMPDIR.BackColor = &H80FF&
ElseIf (Form1.sensors(0).Value = 1) Then
    DIRInterp = IRval
    Form1.IROBS.BackColor = &H80FF&
Else: DIRInterp = HALT

End If

Form1.Forward_B.BackColor = &H8000000F
Form1.Right_B.BackColor = &H8000000F
Form1.Left_B.BackColor = &H8000000F
Form1.Back_B.BackColor = &H8000000F
Form1.Stop_B.BackColor = &H8000000F
```

```
Select Case DIRInterp
    Case 56:
    Form1.Forward_B.BackColor = RGB(255, 0, 0)
    Form1.Direction.Text = "FORWARD"
    Case 54:
    Form1.Right_B.BackColor = RGB(255, 0, 0)
    Form1.Direction.Text = "RIGHT"
    Case 52:
    Form1.Left_B.BackColor = RGB(255, 0, 0)
    Form1.Direction.Text = "LEFT"
    Case 50:
    Form1.Back_B.BackColor = RGB(255, 0, 0)
    Form1.Direction.Text = "BACK"
    Case 53:
    Form1.Stop_B.BackColor = RGB(255, 0, 0)
    Form1.Direction.Text = "STOP"
    End Select

End Function

Function JoyInterp() As Integer

If (CurY = 0) And (CurX = 0) Then
    Form1.JOYDIR.Text = "NONE"
        JoyInterp = Forward
ElseIf (CurY < JoyYForward) Then
        Form1.JOYDIR.Text = "STOP"
        JoyInterp = HALT
ElseIf (CurY > JoyYBack) Then
        Form1.JOYDIR.Text = "BACK"
        JoyInterp = Back
ElseIf (CurX < JoyXLeft) Then
        Form1.JOYDIR.Text = "LEFT"
        JoyInterp = Left
ElseIf (CurX > JoyXRight) Then
        Form1.JOYDIR.Text = "RIGHT"
        JoyInterp = Right
Else: Form1.JOYDIR.Text = "NONE"
        JoyInterp = Forward
End If
End Function

Function CamInterp() As Integer
    If ((C > CAM_THRESHOLD) And (C >= R) And (C >= L)) Then
        Form1.CAMDIR.Text = "CENTER"
        CamInterp = Forward
    ElseIf ((R > CAM_THRESHOLD) And (R >= C) And (R >= L)) Then
        Form1.CAMDIR.Text = "RIGHT"
        CamInterp = Right
    ElseIf ((L > CAM_THRESHOLD) And (L >= R) And (L >= C)) Then
        Form1.CAMDIR.Text = "LEFT"
        CamInterp = Left
    Else
        Form1.CAMDIR.Text = "NONE"
        CamInterp = HALT
    End If
End Function

Function IRInterp() As Integer
    If ((SensorValues(1) > AVOID_THRESHOLD) And (SensorValues(0) >
AVOID_THRESHOLD) And Abs(SensorValues(1) - SensorValues(0) < 10)) Then
        Form1.IROBS.Text = "FRONT"
```

```
        IRInterp = Back
    ElseIf ((SensorValues(0) > AVOID_THRESHOLD) And (SensorValues(0) >
SensorValues(1))) Then
        Form1.IROBS.Text = "RIGHT"
        IRInterp = Right
    ElseIf ((SensorValues(1) > AVOID_THRESHOLD) And (SensorValues(1) >
SensorValues(0))) Then
        Form1.IROBS.Text = "RIGHT"
        IRInterp = Left
    Else
        Form1.IROBS.Text = "NONE"
        IRInterp = Forward
    End If

End Function

Function BumpFInterp() As Integer
    If (SensorValues(3) > LBVal - 5) And (SensorValues(3) < LBVal + 5) Then
        Form1.FBUMPDIR.Text = "LEFT"
        BumpFInterp = Right
    ElseIf (SensorValues(3) > RBVal - 5) And (SensorValues(3) < RBVal + 5) Then
        Form1.FBUMPDIR.Text = "RIGHT"
        BumpFInterp = Left
    ElseIf (SensorValues(3) > CBVal - 5) And (SensorValues(3) < CBVal + 5) Then
        Form1.FBUMPDIR.Text = "CENTER"
        BumpFInterp = Back
    ElseIf (SensorValues(3) > LCBVal - 5) And (SensorValues(3) < LCBVal + 5)
Then
        Form1.FBUMPDIR.Text = "LEFT, CEN"
        BumpFInterp = Right
    ElseIf (SensorValues(3) > RCBVal - 5) And (SensorValues(3) < RCBVal + 5)
Then
        Form1.FBUMPDIR.Text = "RIGHT, CEN"
        BumpFInterp = Left
    ElseIf (SensorValues(3) > 5) Then
        Form1.FBUMPDIR.Text = "FRONT"
        BumpFInterp = Back
    Else: Form1.FBUMPDIR.Text = "NONE"
        BumpFInterp = Forward
    End If
End Function

Function BumpRInterp() As Integer
    If SensorValues(4) > 5 Then
        Form1.RBUMPDIR.Text = "REAR"
        BumpRInterp = Forward
    Else
        Form1.RBUMPDIR.Text = "NONE"
        BumpRInterp = HALT
    End If

End Function

Sub GetLastIRValues()

    Dim TempVal As String

    SensorValues(0) = FindNum(Form1.SensorText(0).Text, "L:", ",")
    SensorValues(1) = FindNum(Form1.SensorText(0).Text, "R:", ",")
    SensorValues(2) = FindNum(Form1.SensorText(0).Text, "C:", ",")
    SensorValues(3) = FindNum(Form1.SensorText(0).Text, "F:", ",")
    SensorValues(4) = FindNum(Form1.SensorText(0).Text, "B:", ",")
```

```
    Form1.IR_L.Caption = SensorValues(0)
    Form1.IR_R.Caption = SensorValues(1)
    Form1.IR_C.Caption = SensorValues(2)
    Form1.BUMP_F.Caption = SensorValues(3)
    Form1.BUMP_B.Caption = SensorValues(4)

End Sub

Function FindNum(StrInput As String, StrStart As String, StrStop As String) As
Integer
'This function returns the numeric value of a section of
'a string starting after StrStart and ending with StrStop

    Dim StPos As Integer
    Dim EndPos As Integer

    StPos = InStr(StrInput, StrStart)
    EndPos = InStr(StPos, StrInput, StrStop)
    FindNum = Val(Mid(StrInput, StPos + Len(StrStart), (EndPos - StPos)))

End Function

Function GetTextBoxLastLinePos(TxtBox As TextBox) As Integer

GetTextBoxLastLinePos = InStrRev(TxtBox.Text, vbCrLf, Len(TxtBox.Text) - 1) + 1

End Function

Sub StartNext()

If Form1.ShowVid.Value = 0 Then
GetNewPicture
Else: GetNewPictureSlow
End If

If (Form1.MSComm1.PortOpen = True) Then
    Form1.MSComm1.Output = Chr(255)
End If



End Sub

Sub GetNewPicture()
    Dim test, j, x, y As Integer
    Dim Pixels(1, 1) As Long
    Dim RedsArray(intUpperBoundX, 1) As Integer
    Dim NumReds, RVal As Integer

    Dim bytRed, bytGreen, bytBlue, bytAverage As Integer

Form1.VidWin.CapSingleFrame
Form1.VidWin.SaveDIB ("c:\VBCODE\temp.bmp")
Form1.Picture3.PaintPicture LoadPicture("c:\VBCODE\temp.bmp"), 0, 0,
Form1.Picture3.ScaleWidth, Form1.Picture3.ScaleHeight

    For x = 1 To intUpperBoundX
        NumReds = 0
        For y = 8 To intUpperBoundY - 5
            Pixels(1, 1) = Form1.Picture3.Point(x, y)
            bytRed = Pixels(1, 1) And &HFF
            bytGreen = ((Pixels(1, 1) And &HFF00) / &H100) Mod &H100
            bytBlue = ((Pixels(1, 1) And &HFF0000) / &H10000) Mod &H100
```

```
            If bytRed + bytGreen + bytBlue < 150 Then
                Pixels(1, 1) = RGB(256, 256, 256)

            ElseIf (bytRed > 125 And bytGreen < 100 And bytBlue < 125) Then
            'bytGreen + bytBlue Then
                NumReds = NumReds + 1
                Pixels(1, 1) = RGB(256, 0, 0)
            Else
                Pixels(1, 1) = RGB(256, 256, 256)
            End If

         Next y
            RedsArray(x, 1) = NumReds
    Next x
    R = 0
    C = 0
    L = 0

    For j = 1 To 12
        L = L + RedsArray(j, 1)
    Next j

    For j = 13 To 28
        C = C + RedsArray(j, 1)
    Next j

    For j = 29 To 40
        R = R + RedsArray(j, 1)
    Next j

        Form1.pictxt(0).Text = L
        Form1.pictxt(0).Refresh
        Form1.pictxt(1).Text = C
        Form1.pictxt(1).Refresh
        Form1.pictxt(2).Text = R
        Form1.pictxt(2).Refresh

End Sub

Sub GetNewPictureSlow()
    Dim test, j, x, y As Integer
    Dim Pixels(1, 1) As Long
    Dim RedsArray(intUpperBoundX, 1) As Integer
    Dim NumReds, RVal As Integer
    'Public R, RC, LC, L, C As Integer

    Dim bytRed, bytGreen, bytBlue, bytAverage As Integer

Form1.VidWin.CapSingleFrame
Form1.VidWin.SaveDIB ("c:\VBCODE\temp.bmp")
Form1.Picture3.PaintPicture LoadPicture("c:\VBCODE\temp.bmp"), 0, 0,
Form1.Picture3.ScaleWidth, Form1.Picture3.ScaleHeight

    Form1.Picture2.Cls
    For x = 1 To intUpperBoundX
        NumReds = 0
        For y = 8 To intUpperBoundY - 5
            Pixels(1, 1) = Form1.Picture3.Point(x, y)
            bytRed = Pixels(1, 1) And &HFF
            bytGreen = ((Pixels(1, 1) And &HFF00) / &H100) Mod &H100
            bytBlue = ((Pixels(1, 1) And &HFF0000) / &H10000) Mod &H100
```

```
            If bytRed + bytGreen + bytBlue < 150 Then
                Pixels(1, 1) = RGB(256, 256, 256)

            ElseIf (bytRed > 125 And bytGreen < 100 And bytBlue < 125) Then
            'bytGreen + bytBlue Then
                NumReds = NumReds + 1
                Pixels(1, 1) = RGB(256, 0, 0)
            Else
                Pixels(1, 1) = RGB(256, 256, 256)
            End If
            Form1.Picture2.PSet (x * 3, y * 3), Pixels(1, 1)

         Next y
            'Picture2.PSet (x, y), RGB(0, 0, 0)
            RVal = NumReds * 30 + 50
            Form1.Picture2.PSet (x * 3, y * 3 + 1), RGB(RVal, RVal, RVal)
            Form1.Picture2.PSet (x * 3, y * 3 + 2), RGB(RVal, RVal, RVal)
            Form1.Picture2.PSet (x * 3, y * 3 + 3), RGB(RVal, RVal, RVal)
            RedsArray(x, 1) = NumReds
    Next x

    R = 0
    'RC = 0
    C = 0
    'LC = 0
    L = 0


    For j = 1 To 13
        L = L + RedsArray(j, 1)
    Next j

    'For j = 8 To 14
    '    RC = RC + RedsArray(j, 1)
    'Next j

    For j = 14 To 26
        C = C + RedsArray(j, 1)
    Next j

    'For j = 27 To 33
    '    LC = LC + RedsArray(j, 1)
    'Next j

    For j = 27 To 40
        R = R + RedsArray(j, 1)
    Next j

        Form1.pictxt(0).Text = L
        Form1.pictxt(0).Refresh
        Form1.pictxt(1).Text = C
        Form1.pictxt(1).Refresh
        Form1.pictxt(2).Text = R
        Form1.pictxt(2).Refresh

End Sub


Public Sub GetJSVals()
Dim Plugged As Boolean
  rc = joyGetDevCaps(JOYSTICKID1, caps, Len(caps))
  ' Initialize struct
   ji.dwSize = Len(ji)
```

```
   ji.dwFlags = JOY_RETURNALL

   ' Get the current joystick data
   rc = joyGetPosEx(JOYSTICKID1, ji)

   ' Display the status
   If (rc = 0) Then
      Form1.status.Caption = "status: joystick connected"
      Plugged = True
   Else
      If (rc = JOYERR_UNPLUGGED) Then
         Form1.status.Caption = "status: joystick unplugged"
         Plugged = False
      Else
         Form1.status.Caption = "status: joyGetPosEx error, rc = " & rc
      End If
   End If

   ' Display the data on the form
   CurX = ji.dwXpos
   CurY = ji.dwYpos
   Form1.axis(0).Caption = CurX
   Form1.axis(1).Caption = CurY

     mask = 1
   For i = 0 To (caps.wNumButtons - 1)
      If (ji.dwButtons And mask) Then Form1.Button(i).Value = 1 Else
Form1.Button(i).Value = 0
      mask = mask * 2
   Next
   If Form1.Button(1).Value = 1 Then
   Form1.Start_Command_Click
   ElseIf Form1.Button(0).Value = 1 Then
   Form1.sensors(2).Value = Abs(Form1.sensors(2).Value - 1)
   'sensors_Click (2)
   ElseIf Form1.Button(2).Value = 1 Then
   Form1.sensors(0).Value = Abs(Form1.sensors(0).Value - 1)
   'sensors_Click (0)
   ElseIf Form1.Button(3).Value = 1 Then
   Form1.sensors(1).Value = Abs(Form1.sensors(1).Value - 1)
   'sensors_Click (1)
   End If

End Sub

Sub WaitTime(WTime As Long)
Dim x, i As Variant
For i = 1 To WTime
x = (i * i / i) / i * i
Next i
End Sub
```