Intelligent Machines Design Laboratory - EEL 5666C: Final Formal Written Report

The Baseballinator

Prepared by: Eric Bennett

April 18, 2011

Instructors:

Dr. Arroyo & Dr. Schwartz

TAs:

Sean Frucht

Ryan Stevens

Tim Martin

Josh Weaver

Devin Hughes

Opening

Abstract

The Baseballinator is a small, autonomous, game-playing robot. This report will outline the design and operation of the robot. The Baseballinator locates a ball, travels toward it, picks the ball up, and bats the ball with a small baseball bat. The robot was successful in completing these tasks.

Executive Summary

To accomplish the task of finding a ball, The Baseballinator uses an IP Webcam to send video to a computer. The computer analyzes the video using OpenCV software. The computer sends out the ball direction to the robot via XBee modules. The robot homes in on the ball by adjusting its motor commands. Once the ball is close enough, the robot lowers its arm and attempts to pick up the ball. After this is done, a strong servo pulls the bat against a strong elastic force. The arm lines the ball up for batting and the servo releases the bat. The ball is then hit and the cycle repeats itself.

Introduction

To accomplish the tasks associated with finding a ball, picking it up, and batting it, while displaying obstacle avoidance behavior, many systems had to be integrated to work together.

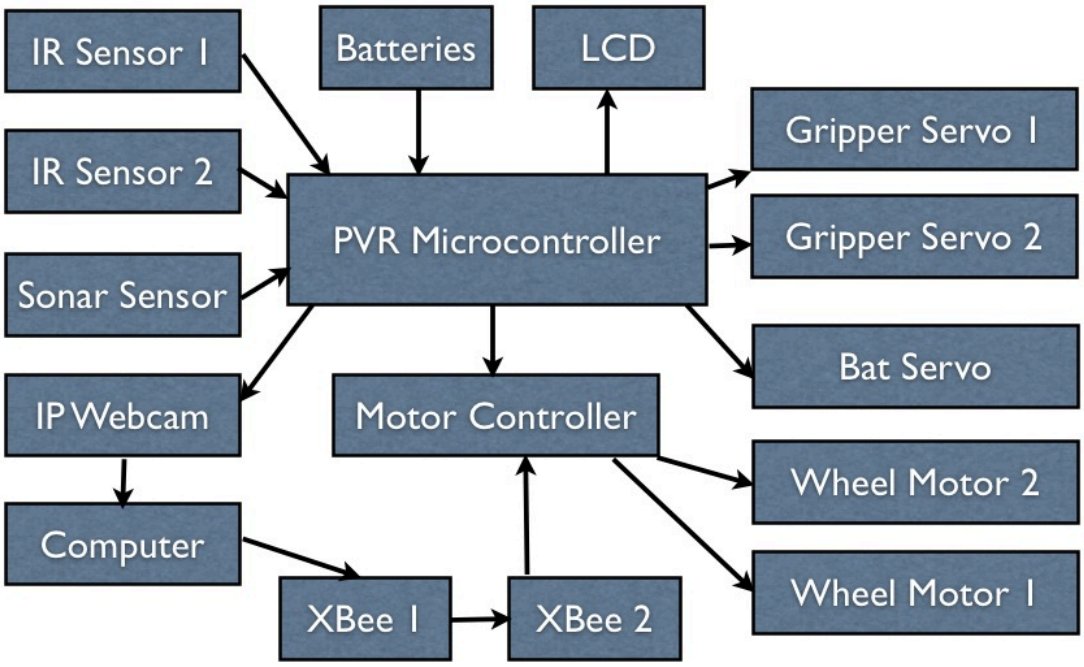
This report will detail The Baseballinator's design. This report starts with a high level description of the entire integrated system. It will move on to the specifics about the mobile platform. After this, there are sections describing the methods of actuation and sensing including circuit diagrams, justification for design decisions, and part characteristics.

Background and Similar Work: Autonomous toys are an attractive product. They allow kids to play in an interactive way with their toy while exposing them to the capabilities of technology for producing intelligent behavior. The Baseballinator allows kids to play the role of outfielder while the robot plays the role of batter and ball fetcher. There have been robots built to do similar things. For instance, Frank Barnes has created the Headless Batter which is a human sized construction of motors and arms. His device detects a high speed pitch of a real baseball and bats it. However, the Headless Batter cannot move anywhere, and has very high power requirements. Another robot built by researchers at Tokyo University have developed a robot that can bat slow pitches and direct the ball to land in a basket with high accuracy. This robot is also immobile. The Baseballinator is different in a few key aspects. Firstly, The Baseballinator is mobile. Therefore, it can be placed near the playing area and it will find its way to where it needs to go. Secondly, The Baseballinator can fetch the ball. Thirdly, this is meant to be a playmate for kids. And lastly, The Baseballinator has lower power requirements and is completely wireless - running off of the onboard batteries.

Main Body

Integrated System

The total integrated system consists of sensors, actuators, batteries, the LCD, and the XBees - all centered around the microcontroller as seen in the diagram below.



The six batteries used to power the PVR microcontroller were 1.2V AA batteries (connected in series). The LCD, each servo, each motor, each sensor, the webcam and XBee 2 were all powered from the microcontroller outputs.

The IR and sonar sensors were used for obstacle avoidance. The sonar sensor was also used to detect the proximity of the ball. The webcam was powered by the board, and sent its video to the computer. The computer analyzes the video and sends a character through XBee 1 to XBee 2 informing the robot to either move left or right. The LCD was used to display what part of the algorithm the robot was in for help in debugging.

Mobile Platform

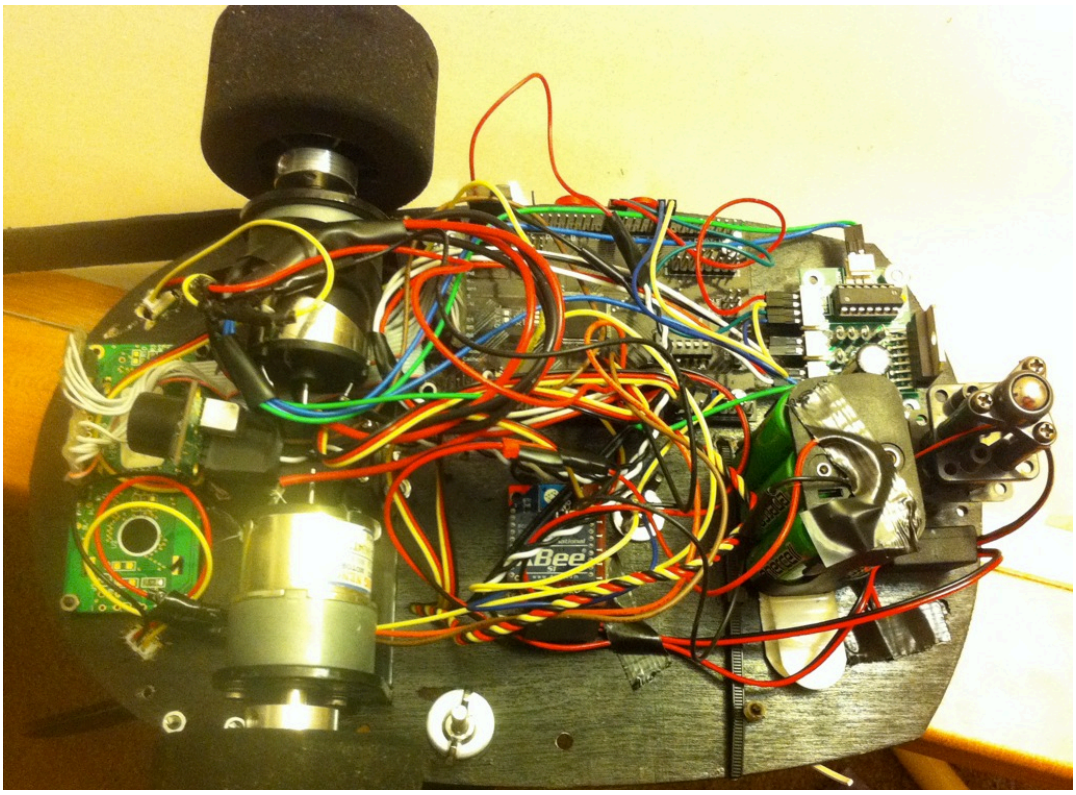
The final mobile platform with all components looked like the picture below. The platform is made of one single piece of thin wood. The gripper is made of the same wood. All of the wood was spray painted black. The LCD, bat support, camera support, and servo support are all

screwed into place. The gripper servos were jammed into place to allow quick changes in positioning during prototyping (to save the time required for screwing and unscrewing them). The robot has 2 wheels with motors screwed into place near the front of the robot. There is a ball caster on the rear end of the robot with its height adapted for use with the robot.

The bat is angled at 45 degrees. When it comes time to hit the ball out of the gripper, the gripper arm also moves to 45 degrees as well.

The wood platform was first designed in AutoDesk Inventor and later cut with the T-Tech machine in the IMDL room. The gripper and many mounting holes were cut later as needed.
Most of the electronic hardware was fit on the underside of the platform as seen in the picture below. The LCD, sonar, and motors can be seen on the left. The microcontroller and XBee module can be seen in the middle of the picture. The battery pack, motor driver, and caster can

be seen on the right. The IR sensors are on the top side of the figure on either side of the LCD and can be seen in the previous picture.



Actuation

Many actuators were used for motion required for the robot.

Two DC motors (controlled with a motor driver) were used to rotate the wheels. The DC motors were part of the EZR1 kit from Solutions Cubed. The motor driver was the Roboblock Dual DC Motor Controller featuring the L298H Bridge motor driver.

For the gripper arm, 2 servos were needed. The servo used to move the arm from the ground to batting position was a 180 degree servo. The second servo was a 90 degree servo needed to pull a string to open the gripper arm. Both of these servos were HiTEC's HS-311 servo.

For gripping, once the ball is found and is in close proximity in front of the robot, the arm sequence is as follows. First, the arm is lowered to an angle above ground. Then the string servo pulls the gripper's string to open the grippers arm. Then the gripper lowers itself to ground level while the robot makes a turn to the right (in order to ensure the ball is picked up in the right position on the left of the gripper). The gripper closes and quickly raises the ball above the camera. The bat servo pulls the bat back and the gripper moves to 45 degrees. The bat servo pulls the bat even further after this point and once it goes "too far" the bat is released and bats the ball. The servos then go to default positions.

The HS-805BB servo was used to pull the bat back. This servo was chosen for its high torque enabling it to pull the bat back against a stronger elastic force. This servo was also 180 degrees enabled.

Although not an actuator, for the sake of completeness, the LCD that was used was the Basic 16x2 Character LCD.

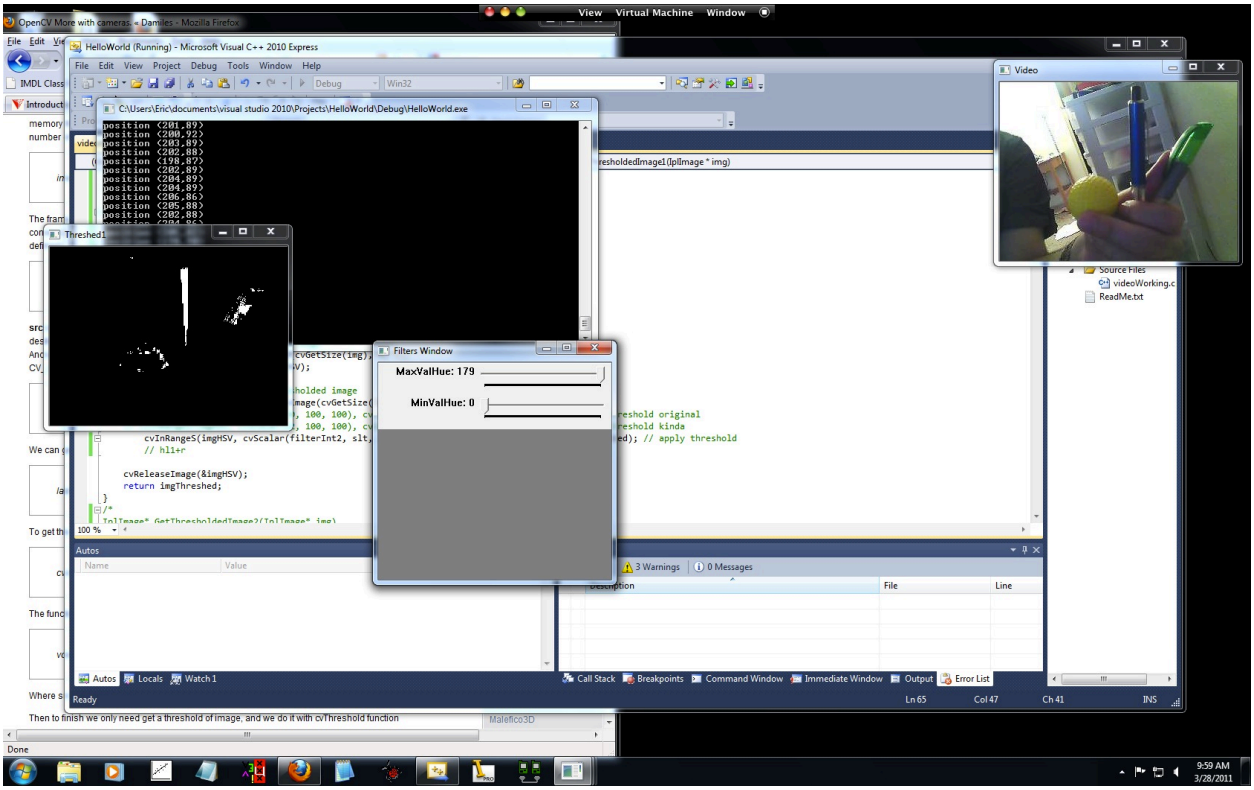
Sensors

Many sensors were needed in order for the robot to successfully implement the desired tasks.

The Cisco Wireless IP Webcam (by Linksys) was used for video feed. The Ad-Hoc mode allowed the camera to send its video to the computer directly, but the signal was far too weak. Because of this, the video was first sent to a router, then to the computer.

The computer was a 2.4 GHz Intel Core i5 MacBook Pro running Windows 7. Video was processed in Visual Express 2010 (C++) and utilized OpenCV software for video analysis. The software was made to convert the RGB frames from the video to HSV. After this, the frame would be thresholded in the saturation and value fields. The Hue value was also thresholded, but was able to be adjusted by the user in real-time as the video is being processed to allow for quick testing to see what values work best to new environments. For demo

day in NEB, orange seemed to work best. The Hue minimum and maximum threshold values were set to 2 and 14 (orange) respectively for the demo.



The C++ code then uses moments to detect the position of the thresholded binary image (which would contain only the orange ball in the receptive field). The x coordinate (in terms of pixels) was used to determine whether the robot should turn left or right to adjust itself to move towards the ball. If no ball was detected, the robot was meant to just stay in obstacle avoidance mode. To implement this, 4 characters were needed to be sent from the computer to the robot. The first indicates standby mode. The second and third indicate motion to the left and right, respectively. The fourth indicated movement forward. The XBee modules were used to send and receive these characters.

The image above shows the software detecting color. Orange is not shown, but is what ended up working best.

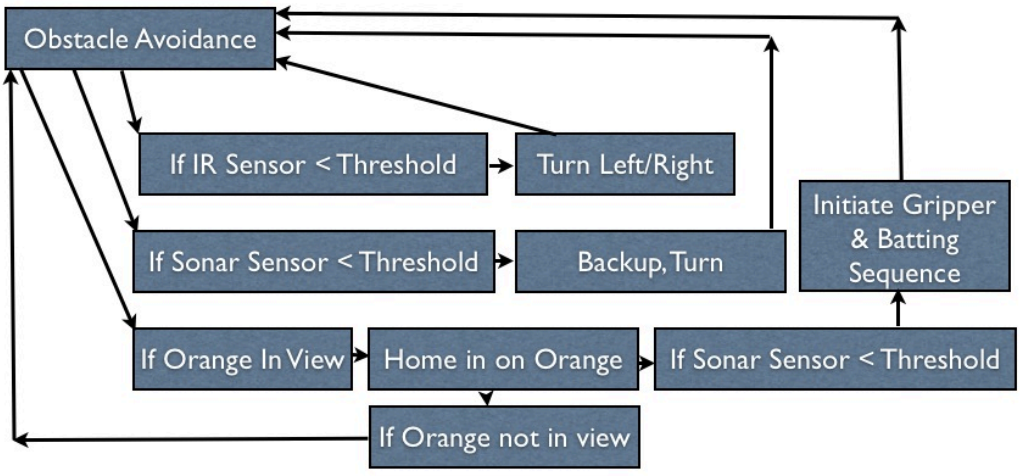
Information is sent from the computer to the robot via RF XBee Modules. These devices were chosen due to their simplicity in setup and functionality. Two XBee 1mW Chip Antenna were used in conjunction with a USB dongle and Explorer Regulated breakout board.

Two Sharp GP2D120 IR Range Sensors were used to assist in obstacle avoidance. These sensors had a range of 4 to 30 centimeters. Precise calibration and characterization of these was not necessary since all that was needed from these was to detect if an object was within 10 cm. If the threshold output voltage was reached, the microcontroller simply sent a command to turn away from that direction for a small amount of time.

The Ultrasonic Range Finer - XL-Maxsonar EZ3 was used for obstacle avoidance and for detecting the proximity of the ball to be picked up. The analog output voltage was used to transmit the proximity data. This particular sonar sensor has a smaller beam width than similar sonar sensors. This was chosen since the robot needed to know the precise location of the ball to be picked up.

Behaviors

The robot is required to perform a variety of tasks. The flow diagram below shows these tasks and the way they are chosen to be executed.



In the above diagram, obstacle avoidance consists of driving in a straight line while monitoring the IR and sonar sensors. Once an orange ball is detected in the field of view, the robot adjusts its motors to center the orange object to the center of the camera's view. The robot would grip the ball and bat it once the sonar detected the ball was within a certain distance and the cycle would repeat itself.

Experimental Layouts & Results

The robot was successful in obstacle avoidance and the task of homing in on an orange ball, picking it up and batting it. A few minor details made it less efficient. For instance, there existed a delay in the video feed. Because of this delay, the robot would oscillate left and right at a frequency equal to the inverse of the delay while homing in on the ball. This is because once the camera has centered itself on the orange ball, the robot keeps on turning because it still sees the need to rotate because the computer is still processing old frames in which the ball is still not centered. This caused the ball to fall out of sight initially. A quick fix was to slow the motors down keeping the amplitude of the oscillations low.

Another drawback was that the main batting servo was not strong enough to pull the bat back when battery power was not at its maximum (when the batteries had just been charged). To remedy this, the gripper servo was made to push the gripper into the batting servo's arm to assist in pushing the bat backwards. The elastics were also loosened resulting in the ball not being batted as far as could be on fresh batteries. The batteries could only supply enough power for a few minutes - after that, everything would still work fine, but the batting servo would not be able to be pulled all the way back against the elastics by itself. When it could, the ball could be batted to eye-level height.

For ball recognition, the code depended on the color of the ball and not the shape. This meant that the orange ball had to be the only orange object in the field of view - or at least represent 80% of the pixels in the camera's field of view. The robot would sometimes try to home in on some far away object, such as an orange juice container (like in the demo). The camera was angled downwards to minimize cases where students would be wearing an orange shirt. In the end, this was not much of a problem.

Closing

Conclusion

The Baseballinator was a successful robot prototype. Obstacle avoidance, as well as finding, gripping, and batting balls were performed well. A few drawbacks such as delay, and the way balls were recognized made the robot's sequence of actions less robust, however, under demo conditions, the robot executed its tasks reasonably.

Future work would include building a robot that can bat a real baseball ball pitched to it, throw/ pitch a baseball, and to also catch a ball thrown to it in uncontrolled conditions seen in real baseball.

Documentation

Please see the website (https://sites.google.com/site/imdlautomatedrobotproject/) for datasheets and resources used.

Appendices

Send me an email if you have any questions about this project. ericbennett@ufl.edu

Code for PVR (C code):

```
#include <avr/io.h>
#include "avr_compiler.h"
#include "usart_driver.h"
#include "PVR.h"
#include <stdio.h>
#include "stdlib.h"

#define USART USARTF0
#define USART_BAUD 11500
#define SERIAL_UBBRVAL(baud) ((((F_CPU / 16) + (baud / 2)) / (baud)) - 1)

#define SAMPLE_SIZE 32
USART_data_t USART_data;


// Global Variables!
int sonarThresh = 70;
int countThresh = 2;
int forward =  0b11000011;
int backward = 0b11000000;
int rotateL =  0b11000010;
int rotateR =  0b11000001;
int crawlR =   0b10000001;
int crawlL =   0b01000010;
int stop = 0b00000000;

        // Servo Assignments//
        // C0 is ARM
        // C1 is CAMERA
        // C2 is STRING
        // C3 is bat mover

int initialdelay = 2000;
int servodelay = 25;
int servodelaydown = 6;
int servodelayopen = 5;
int servodelaybat = 10;
```

```
int flag = 1;
int batservoinitial = 0;
int battingangle = 45;
int zeroangle = 0;


////////////// FUNCTIONS ////////////

// Note: should have made a servodelay(int delay) function
void arm_ground_to_batting()
{
	int k = 0 ;

	for (k=-90; k<40; k++)
	{
		ServoC0(k);
		delay_ms(servodelaydown);
	}

}

void arm_ground_to_zero()
{
	int k = 0 ;

	for (k=-90; k<zeroangle; k++)
	{
		ServoC0(k);
		delay_ms(servodelaydown);
	}

}

void arm_zero_to_batting()
{
	int k = 0 ;

	for (k=zeroangle; k<battingangle; k++)
	{
		ServoC0(k);
		delay_ms(servodelaydown);
	}

}

void arm_zero_to_ground()
{
	int k = 0 ;
	for (k=zeroangle; k>-89; k=k-1)
	{
		ServoC0(k);
		delay_ms(servodelay);
	}
}

void arm_batting_to_almost_ground()
{
	int k = 0 ;
```

```
	for (k=24; k>-60; k=k-1)
	{
		ServoC0(k);
		delay_ms(servodelaydown);
	}
}

void arm_batting_to_zero()
{
	int k = 0 ;
	for (k=24; k>zeroangle; k=k-1)
	{
		ServoC0(k);
		delay_ms(servodelaydown);
	}
}

void arm_almost_ground_to_ground()
{
	int k = 0 ;
	for (k=-60; k>-100; k=k-1)
	{
		ServoC0(k);
		delay_ms(servodelaydown);
	}
}

void arm_zero_to_almost_ground()
{
	int k = 0 ;
	for (k=zeroangle; k>-60; k=k-1)
	{
	ServoC0(k);
	delay_ms(servodelaydown);
	}
}

void bat_zero_to_minus()
{
	int k = 0 ;
	for (k=0; k>-100; k=k-1)
	{
		ServoC3(k);
		delay_ms(servodelaybat);
	}
}

void bat_minus_to_plus()
{
	int k = 0 ;
	for (k=-100; k<100; k++)
	{
		ServoC3(k);
		delay_ms(servodelaybat);
	}
}

void bat_minus_to_zero()
```

```
{
	int k = 0 ;
	for (k=-100; k<0; k++)
	{
		ServoC3(k);
		delay_ms(servodelaybat);
	}
}

void bat_zero_to_half_plus()
{
	int k = 0 ;
	for (k=0; k<100; k++)
	{
		ServoC3(k);
		delay_ms(servodelaybat);
	}
}

void bat_zero_to_plus()
{
	int k = 0 ;
	for (k=0; k<100; k++)
	{
		ServoC3(k);
		delay_ms(servodelaybat);
	}
}

void bat_half_plus_to_plus()
{
	int k = 0 ;
	for (k=80; k<100; k++)
	{
		ServoC3(k);
		delay_ms(servodelaybat);
	}
}

void bat_plus_to_zero()
{
	int k = 0 ;
	for (k=100; k>0; k=k-1)
	{
		ServoC3(k);
		delay_ms(servodelaybat);
	}
}




void arm_open()
{
	int k = 0 ;
	for (k=100; k>-100; k=k-1)
	{
		ServoC2(k);
```

```
                delay_ms(servodelayopen);
        }
}


void arm_close()
{
        PORTJ_OUT = forward;
        delay_ms(200);
        PORTJ_OUT = crawlR;
        delay_ms(200);
        int k = 0 ;
        for (k=-100; k<100; k++)
        {
                ServoC2(k);
                delay_ms(servodelayopen);
        }
        PORTJ_OUT = stop;
}


void backup()
{
                PORTJ_OUT = backward;
                delay_ms(400);
}

void rotateLeft()
{
                PORTJ_OUT = rotateL;
                delay_ms(700);
}


 // Compares sonar reading to threshold. Reacts if triggered more than 'count'
int checksonar(int sonar,int count)
{
        if (sonar < sonarThresh)
                count=count+1;

        lcdGoto(1,1);
        lcdInt(count);

        if (count > countThresh)
        {
                backup();
                rotateLeft();
                count=0;
        }

        return(count);
}

void pickupball2()    ///This is the algorithm to pick up a ball with the arm and bat it
{
                arm_zero_to_almost_ground();
                arm_open(); // maybe open arm while going down.
                ServoC3(30);
```

```
                arm_almost_ground_to_ground();
                arm_close();
                ServoC3(-10);

                arm_ground_to_zero();
                delay_ms(400);
                ServoC0(-100); //To help Batter pull back

                ServoC3(-45);
                arm_zero_to_batting();
                delay_ms(1000);

        //      PORTJ_OUT = rotateR;
        //      delay_ms(1500);
        //      PORTJ_OUT = stop;

                ServoC3(-100);
                delay_ms(400);

                arm_batting_to_zero();
                ServoC3(0);
}


/////// Serial Code Stuff (Thanks Steven + Internet)

void usart_initialize(void)
{
        //pin 3 output
        PORTF.DIRSET = PIN3_bm;
        //pin2 input
        PORTF.DIRCLR = PIN2_bm;
//      USART_InterruptDriver_Initialize(&USART_data, &USART, USART_DREINTLVL(3));
        //usartc0, 8 data bits, no parity, 1 stop bit
        USART_Format_Set(&USART, USART_CHSIZE_8BIT_gc, USART_PMODE_DISABLED_gc, false);
        //ENABLE INTERRUPT
//      USART_RxdInterruptLevel_Set(USART_data.usart, USART_RXCINTLVL(3));
        //set baud rate
        USART_Baudrate_Set(&USART, 17, 0);

        //ENABLE RX AND TX
        USART_Rx_Enable(&USART);
        USART_Tx_Enable(&USART);
        //Enabel PMIC Interrupt level low
        //PMIC.CTRL |= PCMIC_LOLVLEX_bm;

        //enable global interrupts
        //sei();
}

inline void usart_tx_byte(char DataByte)
{
        int txrxVal = 0;

        while(1)
        {
                txrxVal = USARTF0_STATUS;
```

```
                txrxVal &= 0x20;;
                if(txrxVal == 0x20)
                {
                        USARTF0_DATA = DataByte;
                        break;
                }
        }
}

inline void usart_tx_string(char *StringPtr)
{
        int i = 0;
        while(StringPtr[i] != 0) //while not null terminator
        {
                usart_tx_byte(StringPtr[i]);
                i++;
        }
}


inline char usart_rx_byte(void)
{
                int txrxVal =  USARTF0_STATUS;
                txrxVal &= 0x80;
                char data;
        if(txrxVal == 0x80)
                {
                        data = USARTF0_DATA;
                }
                else
                {
                        data = 0;
                }
                return data;
}


//////////////////////////////// !!! MAIN LOOP MAIN LOOP !!!  ////////////////////////
int main(void)
{
        xmegaInit();        //setup XMega
        delayInit();        //setup delay functions
        ServoCInit();       //setup PORTC Servos
        ServoDInit();       //setup PORTD Servos
        ADCAInit();                 //setup PORTA analong readings
        lcdInit();                  //setup LCD on PORTK
        lcdString("PV Robotics");   //display "PV Robotics" on top line (Line 0) of LCD
        lcdGoto(1,0);               //move LCD cursor to the second line (Line 1) of LCD
        lcdString("Board Demo");    //display "Board Demo" on second line
        PORTQ_DIR |= 0xFF;          //set Q0 (LED) as output
        PORTJ_DIR |= 0xFF;          //set port F as output

// variables for main loop
        int i = -100;
        int value = 0;
        int sonar = 0;
        int count = 0;
        int ob_avoid = 0;
```

```
// Set initial conditions
   PORTJ_OUT = stop;
            ServoC0(0);
            //ServoC0(45);
            ServoC2(100);
            ServoC3(batservoinitial);
            delay_ms(initialdelay);
            usart_initialize(); //

            delay_ms(3000);
//          pickupball(); // for testing

char avalue;
//int avalue = 0;
int intvalue;
int posx = 0;

int addthis = 4;
int kk=22;

//while(1) // for testing
//{
//lcdData(0x01);    //Clear LCD

//lcdInt(kk);
//ServoC0(kk);
//kk=kk+1;
//delay_ms(3000);
//}

//// Make sure batting arm and grabber don't interfere with eachother
ServoC0(-45);
ServoC3(45);
delay_ms(500);
ServoC3(10);
delay_ms(200);
ServoC0(0);

int ir = 0;
int irr = 0;

// while(1) // test the IR sensors
{
            lcdData(0x01);
//          lcdString("IR: ");
            ir = ADCA1();
            irr = ADCA2();
            ir = (ir-300)/1;
            irr = (irr-300)/1;

            lcdInt(ir);
            lcdString("    ");
            lcdInt(irr);

            delay_ms(300);

}
```

```
// pickupball2(); // for testing

while(1)
{
// uncomment and loop to test xbee communication and character display
 // lcdInt(i);
//          i++;

          lcdData(0x01);                        //Clear LCD
//          lcdInt(i);
//          delay_ms(500);
          avalue = usart_rx_byte();
//          lcdChar(avalue);
//          delay_ms(500);
//          if (avalue == 'g' || avalue == 'G')
//          {         lcdString(avalue);          // testing the output - displaying it to the LCD
//          }
//          delay_ms(500);
///         lcdInt(avalue);
//          delay_ms(500);
//          intvalue = atoi(avalue); // atoi seems to mess things up.  results in no output to LCD
          intvalue = avalue - '0';
//          intvalue = intvalue + addthis;
//          lcdData(0x01);                        //Clear LCD
          lcdInt(intvalue);
//          delay_ms(500);


          sonar = ADCA0();                      //Read A/D value from PortA,
          sonar = (sonar-300)/1;          //normalize its reading (like in obstacleavoid())

// if (sonar > 40)
          {
                    if (intvalue == 9) // If ball is on the right of field of camera view, then...
                              {         PORTJ_OUT = crawlR;
                                        sonar = ADCA0();            //Read A/D value from PortA, Pin0 (between 0-4096)
                                        sonar = (sonar-300)/1;      //Sonar's smallest value was -84 at closest distance

                                                  if (sonar < 25) // If ball is near, then...
                                                  {
                                                            PORTJ_OUT = forward;
                                                            delay_ms(120);              // move forward a little bit first
                                                            PORTJ_OUT = stop;
                                                            pickupball2();
                                                  }

                              }

                    if (intvalue == 1)
                              {
                                        PORTJ_OUT = crawlL;

                                        sonar = ADCA0();                    //Read A/D value from PortA, Pin0 (between
0-4096)
                                        sonar = (sonar-300)/1;
                                                  if (sonar < 25)
```

```
                                        {
                                                PORTJ_OUT = forward;
                                                delay_ms(120);                  // move forward a little bit first
                                                PORTJ_OUT = stop;

                                                pickupball2();
                                        }
                        }

                ///// " if ball is not in view, then do obstacle avoidance
                if (intvalue == 2)
                        {
                                //// LOOK FOR BALL    /    OBSTACLE AVOIDANCE
                                PORTJ_OUT = forward;      //initial movement is forward

                                sonar = ADCA0();  //Read A/D value from PortA, Pin0 (between 0-4096)
                                sonar = (sonar-300)/1;
                                lcdData(0x01);                          //Clear LCD
                                lcdString("Sonar: ");
                                lcdInt(sonar);

                                count = checksonar(sonar,count); // goes backwards and ends with rotateL

                                delay_ms(15);

                // IR sensor code.  go right if somethings on the left, go left if somethings on the right
                                        ir = ADCA1();
                                        ir = (ir-300)/1;
                                        irr = ADCA2();
                                        irr = (irr-300)/1;
                                        if (ir > 2000)                                          ///// must test value!!!
                                                PORTJ_OUT = crawlR;

                                        if (irr > 2000)
                                                PORTJ_OUT = crawlL;

                                        delay_ms(500);
                                }
        }
}
}
```

Code for Computer (C++ Code in Visual Studio Express 2010):

(See the online links to see how to set up the libraries and dependencies, etc...)

```
/**

//// restart computer / visual studio
//// CONNECT TO NETWORK
//// PLUG IN XBEE USB
//// TURN ON ROBOT
//// RUN CPP CODE WHEN BLUE LIGHT ON CAMERA STOPS BLINKING

// If not working, go to cam website, user password might be admin admin. fixed ip address.  same wep and
// key 1 passwordd, same ssid (2WIRE323)

NOTES on IP WEBCAM: TopLeft is (0,0), BottomRight is (318,239)ish.  Biggest Xvalue is 318 on the far right
Delay Exists when displaying video.  Don't display video for real thing!
```

```
// Do not use pre-compiled headers!  right click on videoWorking

 * Display video from webcam, filter image...
 * The OpenCV color tracking portion of this code was taken heavily from a tutorial online at aishack.com
 * Other online sources were used as well... see website for details
 */
#include "stdafx.h"
#include <stdio.h>
#include "cv.h"
#include "highgui.h"
#include "Serial.h"

using namespace cv;
using namespace std;

// HSV - HUE (color), SATURATION (0 grey, 255 color), VALUE (0 black, 255 white)
int sut=255; // original is 255
int slt =150; // original is 100
int vut = 255; // original is 255
int     vlt=100; // original is 100

int hl1 = 25;
int hl2 = 10;
int hl3 = 15;
int hl4 = 20;
int hl5 = 25;
int hl6 = 30;
int hl7 = 35;
int r = 10;
// The above values were for testing initially
// Became obsolete with the trackbar function

// FOR YELLOW
// SO FAR hue from 25 to 35 works best ... sat, 150 to 255, val 100 to 255  (WINDOW 5)
//

// FOR GREEN
// H(60,100), S(50,255), V(50,255)

int g_switch_value = 0;
int filterInt = 0;
int filterInt2 = 0;
int lastfilterInt = -1;

void switch_callback( int position ){
        filterInt = position;
}
void switch_callback2( int position ){
        filterInt2 = position;
}

//// Thresholding Function
IplImage* GetThresholdedImage1(IplImage* img)
{
        // Convert the image into an HSV image
    IplImage* imgHSV = cvCreateImage(cvGetSize(img), 8, 3);
    cvCvtColor(img, imgHSV, CV_BGR2HSV);

        // Create new image to hold thresholded image
        IplImage* imgThreshed = cvCreateImage(cvGetSize(img), 8, 1);
        // cvInRangeS(imgHSV, cvScalar(22, 100, 100), cvScalar(32, 255, 255), imgThreshed); // apply
//threshold absolute
//      cvInRangeS(imgHSV, cvScalar(filterInt2, slt, vlt), cvScalar(filterInt, sut, vut), imgThreshed); //
//apply threshold with trackbar
        cvInRangeS(imgHSV, cvScalar(2, slt, vlt), cvScalar(14, sut, vut), imgThreshed); // apply threshold
//ORANGE

        cvReleaseImage(&imgHSV);
    return imgThreshed;
}
```

```
                int dir = 0;

int main( int argc, char **argv )
{
        fprintf( stderr, "Starting..." );

        ////////////////////// Serial stuff
        CSerial serial;

    // Attempt to open the serial port (COM1)

    serial.Open(_T("COM3"));
        fprintf( stderr, "COM3 Opened..." );
    // Setup the serial port (9600,N81) using hardware handshaking

    serial.Setup(CSerial::EBaud115200,CSerial::EData8,CSerial::EParNone,CSerial::EStop1); // BAUD RATE
//CHANGE HERE
    serial.SetupHandshaking(CSerial::EHandshakeOff);
        fprintf( stderr, "serial has been set up..." );
    // The serial port is now ready and we can send/receive data. If
    // the following call blocks, then the other side doesn't support
    // hardware handshaking.
    // serial.Write("Hello world");

        const char* name = "Filters Window";
    CvCapture *capture = 0;
    IplImage  *frame = 0;
    int       key = 0;

        // Loop to test serial
/*      int test = 4;
        while(1)
        {
                serial.Write(test);
        }
*/

        /* initialize camera */
    capture = cvCreateFileCapture("http://192.168.1.67/img/video.mjpeg");
                //capture = cvCaptureFromCAM("http://192.168.1.67/img/video.mjpeg"); // didn't work

    /* always check if camera was opened/initialized */
    if ( !capture ) {
        fprintf( stderr, "Cannot!!!!!!!!!!!!!! open/initialize webcam!\n" );
        return 1;
    }
 fprintf( stderr, "Camera opened." );
    /* create a window for the video and threshed video*/
    cvNamedWindow( "Video", CV_WINDOW_AUTOSIZE );
    cvNamedWindow( "Threshed1", CV_WINDOW_AUTOSIZE );

        // This image holds the "scribble" data...
    // the tracked positions of the ball
    IplImage* imgScribble = NULL;
        // Anything to do with "scribble" is now vestigial

        cvNamedWindow( name, 1 ); // Create Trackbar Window

        // Create trackbar, use callbacks
        cvCreateTrackbar( "MaxValHue", name, &g_switch_value, 179, switch_callback );
        cvCreateTrackbar( "MinValHue", name, &g_switch_value, 179, switch_callback2 );

    while( key != 'q' ) {
        /* get a frame */
        frame = cvQueryFrame( capture );
        if( !frame ) break; // End if frame not found

                // cvSmooth( frame, frame, CV_BLUR, 4, 4 );             // Smooth image if needed (apply
//blurring filter (moving average))
```

```
                //  cvErode(frame, frame, 0, 2);                        // Erode filter - not necessary

                // If this is the first frame, we need to initialize it
        if(imgScribble == NULL)
        {
            imgScribble = cvCreateImage(cvGetSize(frame), 8, 3);
        }

                // Holds the yellow thresholded image (yellow = white, rest = black)
        IplImage* imgYellowThresh1 = GetThresholdedImage1(frame);


                //// Calc position assuming its the only thing that is that color!

                // Calculate the moments to estimate the position of the ball
        CvMoments *moments = (CvMoments*)malloc(sizeof(CvMoments));
        cvMoments(imgYellowThresh1, moments, 1);

        // The actual moment values
        double moment10 = cvGetSpatialMoment(moments, 1, 0);
        double moment01 = cvGetSpatialMoment(moments, 0, 1);
        double area = cvGetCentralMoment(moments, 0, 0);

                // Holding the last and current ball positions
        static int posX = 0;
        static int posY = 0;

 //       int lastX = posX;
 //       int lastY = posY;

        posX = moment10/area;
        posY = moment01/area;


                if (posX > 160)
                        dir = 9; // turn right

                if ((posX < 160) & (posX > 0))
                        dir = 1; // turn left

                if (posX < 1)
                        dir = 2; // don't turn left or right, just obstacle avoid

                serial.Write(dir); // Send command through XBees to robot

                // Print out positions for debugging purposes
        printf("position (%d,%d)\n", posX, posY);
/*
                //// Vestigial line drawing for ball tracking
                // We want to draw a line only if its a valid position
        if(lastX>0 && lastY>0 && posX>0 && posY>0)
        {
            // Draw a yellow line from the previous point to the current point
       //     cvLine(imgScribble, cvPoint(posX, posY), cvPoint(lastX, lastY), cvScalar(0,255,255), 5);
        }

                // Add the scribbling image and the frame...
                // cvAdd(frame, imgScribble, frame);
*/
        cvShowImage("Threshed1", imgYellowThresh1);
        cvShowImage("Video", frame);

                 // Wait for a keypress
        int c = cvWaitKey(10);
        if(c!=-1)
        {
            // If pressed, break out of the loop
            break;
        }
                // Release the thresholded image+moments... we need no memory leaks.. please
        cvReleaseImage(&imgYellowThresh1);
```

```
        delete moments;
    }

        // We're done using the camera. Other applications can now use it
    cvReleaseCapture(&capture);

        serial.Close();
    return 0;
}
```