

Ballbot

Scott Pokorny

EML 5666 Intelligent Machines Design Lab

Instructors:

Dr. A. Antonio Arroyo and Dr. Eric M. Schwartz

TAs:

Tim Martin, Ryan Stevens, and Josh Weaver

Table of Contents

Abstract	4
Executive Summary.....	4
Introduction	4
Integrated System.....	5
Microcontroller Board	5
Power	5
Movement.....	5
Obstacle Avoidance.....	6
Detection.....	6
Debugging	6
Mobile Platform	6
Motors.....	7
Wheels	8
Actuation.....	8
Sensors.....	9
Camera	9
Sonar	9
Bump Switches.....	9
Behaviors.....	10
Robot Code	10
Calibration Mode	10
Collection Mode.....	10
Camera Code.....	13
Main	13
Finding Targets.....	14
Finding the Robot.....	15
Determining Instructions	16
Experimental Layout and Results.....	19
Initial Board Testing	19
Initial Sonar Testing.....	19
Initial System Testing	19

Obstacle Avoidance Initial Testing.....	19
Obstacle Avoidance Beta Testing.....	19
Dead Zone Testing	19
Camera Testing	20
XBEE Testing.....	20
Orientation Testing	20
Integrated Orientation Testing	20
Collection Testing.....	21
Conclusion.....	21
Documentation	21
Appendices.....	21
Robot Code	21
Camera Code.....	25
Operation Code.....	25
HSV Color Code	30
XBEE Serial Port Test	33

Abstract

The Ballbot will autonomously move about the tennis court in order to pick up tennis balls for the player. This allows more time to be dedicated to playing the game and less to retrieving balls.

Executive Summary

The Ballbot will be an autonomous robot that collects tennis balls. The robot will consist of a mobile platform and stationary camera. The mobile platform will hold the microcontroller, the collector, an LCD, sonar, and bump switches. The camera will be by itself observing the court and communicating with the mobile platform.

The robot will collect tennis balls by detecting them with a stationary camera that will direct a robot toward the ball. The camera will send data to a laptop which will make calculations and send directions to the robot using XBEE terminals.

In order to prevent the robot from crashing into objects, including players, obstacle avoidance will be programmed into all behaviors. In order to detect obstacles, a sonar module has been installed. In order to prevent major collisions, bump sensors consisting of basic push switches have been installed.

To move around, the mobile platform consists of two drive wheels and two casters. Each caster is attached to one of the robot arms in order to keep the robot balanced. Each wheel has a motor to allow the robot to make very tight turns.

An LCD is currently used to display the values of the sonar and bump switches. These values are updated every time the robot's current behavior starts a new iteration. The LCD is also used to show whether the robot is in calibration mode or collection mode.

To collect tennis balls, an actuator consisting of four belts moves balls from the ground into the back of the robot. Each belt is powered by a motor, which are powered by two 9V battery packs. Two belts have been modified to have Velcro attached to them in order to assist in picking up tennis balls.

Introduction

When playing tennis, it can seem that a good chunk of play time is wasted walking around the court picking up the tennis balls. In this day and age, it only seems logical that the balls should at least meet you halfway and all be easily accessible in one location. Ballbot will be made to help meet the player halfway by autonomously gathering tennis balls and storing them in an easy to access container.

Integrated System

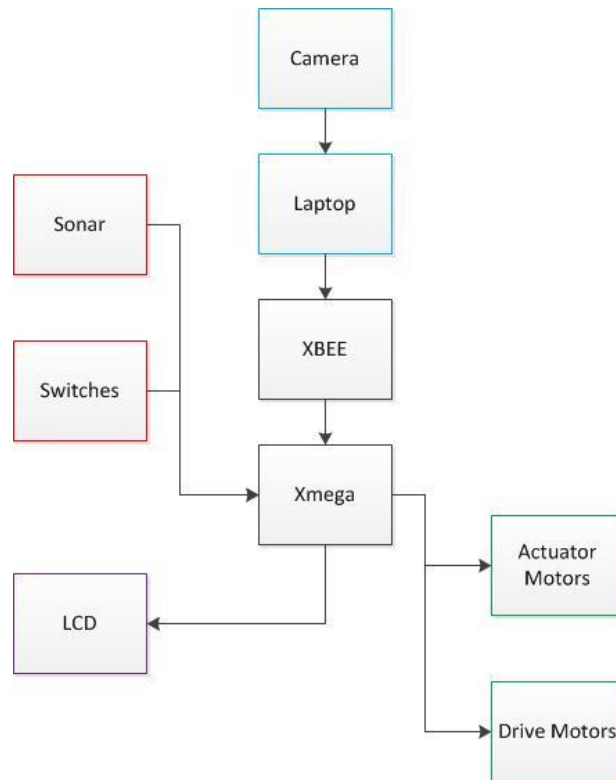


Figure 1: Current system flow chart

The Ballbot integrated system will consist of switches, a camera, sonar, motors, and an LCD. Currently there are four systems, as represented by the colored borders, which send and receive information from the Xmega microcontroller board.

Microcontroller Board

The microcontroller board used in this robot is the Epiphany DIY board. It is built by Tim Martin and can be purchased through the company Out of the Box. The main processor in the board is an ATxmega64A1 chip. Additionally, it can support up to four motors, twenty four servos, and eight analog to digital converter ports.

Power

The robot is currently powered by one 12V battery pack. The ground terminal of the battery pack is attached to a switch that controls the power to the entire board. Two 9V battery packs are used to power the actuator motors.

Movement

The robot is currently driven by two drive wheels. Two caster wheels have been added to the front of the robot in order to balance the robot and keep everything except from the ramp from dragging. The microcontroller board will power the motors and adjust their speeds to allow for turning to allow for the robot to line up with a detected tennis ball.

Obstacle Avoidance

The current obstacle avoidance sensor system consists of one sonar device and two bump switches. The sonar is installed in the front of the robot in order to detect any objects in front of the robot. The bump switches are placed in the dead zones of the sonar to act as fail safes. If the sensor detects that an object is too close to the robot, the robot will turn until the sonar declares that it is safe to continue. The current value that will set off the sonar is 400 mV, or about 23.5 inches. The sonar is angled slightly upward so that it does not detect objects that are less than six inches in height at the moment. This may be reduced to five depending on further testing. The sonar must not go below 3 inches though, due to the fact that tennis balls must be able to approach the robot without starting up the obstacle avoidance code. When a bump switch is pressed, this means that the robot is right up on an object. Thus, the robot will back up and turn away from the side that has been bumped. In order to keep the robot from turning from one obstacle to another, the robot will not move forward until all the sensors have been cleared.

Detection

An IP camera and laptop will be used to detect tennis balls and the robot. The camera will attach to the laptop, which will use visual studio and the OpenCV library to find the robot and tennis balls. When both the robot and a tennis ball are found, the laptop will decide the direction that the robot needs to travel and send it instructions via XBEE. When the distance between the tennis ball and the robot becomes small, the laptop will send an additional instruction to the robot for starting the collection motors. These motors will stay on until the instruction is not sent. Once the instruction is not sent by the XBEE, the microcontroller will begin a timer which will turn off the motors when it finishes.

Debugging

The LCD is used to report what behaviors the system is currently in. This has helped in programming the values for obstacle avoidance and finding the dead zones of the sonar. The LCD displays the current outputs of each switch and the sonar. This is used to ensure that the sonar is performing correctly before moving into the actual operation. To exit this behavior, the user would press the left bump switch when facing the front of the robot.

Mobile Platform

The current mobile platform has a rectangular design. It is driven by two drive wheels located near the center of the robot. This allows for a small turning radius when both wheels are used. The motors for the wheels are completely housed inside the robot and will be located underneath the ramp. On the top of the ramp is the actuator for tennis ball collection. Above that is housing for the microcontroller board, followed by the top panel of the robot. The LCD is attached to the rear lid of the robot to allow for easy viewing.

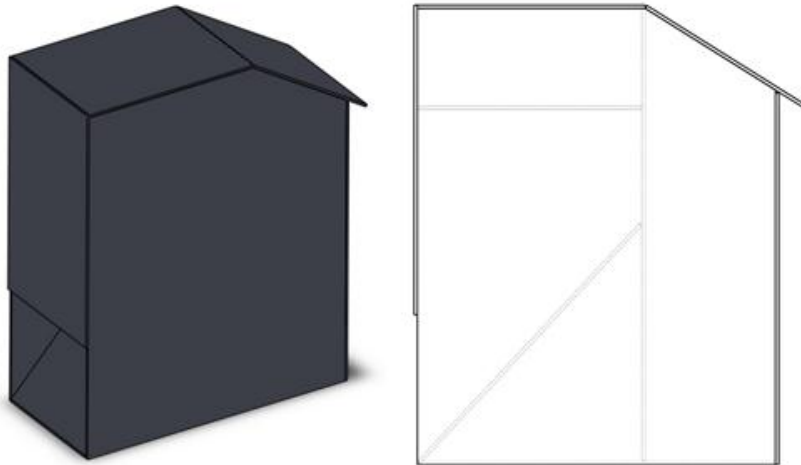


Figure 2: Original mobile platform design



Figure 3: Actual implementation

Motors

The motors used in the robot are made by Pololu. They are 37x54 mm metal gear motors with 67:1 gear ratio. The two motors run on 12V and are screwed into the bottom panel and are adjacent to the middle panel.



www.pololu.com

Figure 4: Motors used in the robot

Wheels

The wheels used on the robot are also made by Pololu. They are 90 mm diameter plastic wheels with silicone tires. A caster may be added later to increase stability if needed.



Figure 5: Wheels used in the robot

Actuation

The actuation of the robot is in the tennis ball collector. As tennis balls are funneled into the collection mechanism, a series of four belts feed tennis balls into a compartment in the robot. The collection mechanism is powered by four motors. Two of the belts have been modified to have Velcro attached to them. This assists in picking up tennis balls, giving the actuator a greater grip.

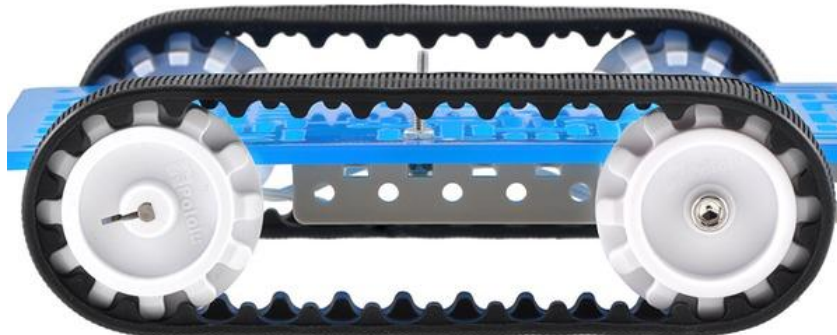


Figure 6: Belts used for the actuator

Sensors

The sensors that will be used in this robot for obstacle avoidance are sonar and bump switches. The special sensor that will be used is a camera for tennis ball detection.

Camera

The camera I purchased is the Cisco-Linksys Wireless-N Internet Home Monitoring Camera, via Amazon.com. I chose this since it was recommended by Josh Weaver and was informed that it does work with OpenCV. It will not be installed on the robot, but will be placed in a static position overlooking the entire court. The camera easily performs as to what is needed, but some lag issues are definitely present.



Figure 7: IP camera

Sonar

The sonar sensor that is used is the Maxbotix LV-EZ1 Ultrasonic Range Finder. This was purchased from Sparkfun and is installed in the front lid of the robot. The sensor runs on 5V and can detect objects from 0 to 6.45 meters. The sensor's lowest reported value is for six inches.



Figure 8: Sonar sensor

Bump Switches

The bump switches used in the robot are single pull double throw switches. They were purchased from Radio Shack and run on a 5V supply from the microcontroller board. They have been manually wired with a pull-up resistor, which was recommended by Josh Weaver.



Figure 9: Bump switch. The actual switch used also has a roller on the end

Behaviors

Currently, two behaviors have been written for the robot: calibration and collection. This section will include the code used during robot operation. Some code is omitted for space considerations, however all code is available in the appendix.

Robot Code

Calibration Mode

When the robot is turned on, it does the board initializations and enters calibration mode.

```
while(analogRead_ADCA(6) > 400)
{
    LCDSetCursor(1,1);
    fprintf(&lcd_str, "\r");
    fprintf(&lcd_str, "%d %d
%d\nCalibration", analogRead_ADCA(0), analogRead_ADCA(6), analogRead_ADCA(7));
    _delay_ms(200);
}
```

After initialization, the robot moves into a while loop that refreshes every 0.2 seconds. It prints all the values of the sensors so that the user can check to make sure that all sensors are working accordingly. When the value of the sensor plugged into ADC Port A channel 6 goes below 400, the loop is exited and the obstacle avoidance mode is entered. This is currently mapped to the right bump switch, so when this is pressed the robot will begin collecting balls.

Collection Mode

After calibration mode is exited, the robot currently enters collection mode.

```
while (1)
{
    if(dataInBufE1())
    {
        fscanf(&Xbee_str, "%c", msg);
    }
    LCDSetCursor(1,1);
    fprintf(&lcd_str, "\r");
    fprintf(&lcd_str, "%d %d
%d\nCollection", analogRead_ADCA(0), analogRead_ADCA(6), analogRead_ADCA(7));
    _delay_ms(200);
    if(analogRead_ADCA(0) < 400)
    {
        setMotorDuty(1,0, MOTOR_DIR_FORWARD_gc);
    }
}
```

```

        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
        _delay_ms(400);
        setMotorDuty(1,600,MOTOR_DIR_BACKWARD_gc);
        setMotorDuty(2,600,MOTOR_DIR_FORWARD_gc);
        _delay_ms(500);
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
        _delay_ms(300);
        j=1;
    }
else if(analogRead_ADCA(6) < 400)
{
    setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
    setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
    _delay_ms(400);
    setMotorDuty(1,600,MOTOR_DIR_BACKWARD_gc);
    setMotorDuty(2,600,MOTOR_DIR_BACKWARD_gc);
    _delay_ms(1000);
    setMotorDuty(1,600,MOTOR_DIR_FORWARD_gc);
    _delay_ms(500);
    setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
    setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
    _delay_ms(300);
    j=1;
}
else if(analogRead_ADCA(7) < 400)
{
    setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
    setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
    _delay_ms(400);
    setMotorDuty(1,600,MOTOR_DIR_BACKWARD_gc);
    setMotorDuty(2,600,MOTOR_DIR_BACKWARD_gc);
    _delay_ms(1000);
    setMotorDuty(2,600,MOTOR_DIR_FORWARD_gc);
    _delay_ms(500);
    setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
    setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
    _delay_ms(300);
    j=1;
}
else if(j==1)
{
    setMotorDuty(1,600,MOTOR_DIR_FORWARD_gc);
    setMotorDuty(2,600,MOTOR_DIR_FORWARD_gc);
    _delay_ms(300);
    j=0;
}
else
{
    if(strcmp(msg,"W")==0)
    {
        setMotorDuty(1,600,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,600,MOTOR_DIR_FORWARD_gc);
    }
    else if(strcmp(msg,"D")==0)
    {
        setMotorDuty(1,600,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,600,MOTOR_DIR_BACKWARD_gc);
    }
}

```

```

        _delay_ms(100);
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
    }
    else if(strcmp(msg,"V")==0)
    {
        setMotorDuty(1,600,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,600,MOTOR_DIR_BACKWARD_gc);
        _delay_ms(200);
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
    }
    else if(strcmp(msg,"B")==0)
    {
        setMotorDuty(1,600,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,600,MOTOR_DIR_BACKWARD_gc);
        _delay_ms(300);
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
    }
    else if(strcmp(msg,"N")==0)
    {
        setMotorDuty(1,600,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,600,MOTOR_DIR_BACKWARD_gc);
        _delay_ms(400);
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
    }
    else if(strcmp(msg,"A")==0)
    {
        setMotorDuty(1,600,MOTOR_DIR_BACKWARD_gc);
        setMotorDuty(2,600,MOTOR_DIR_FORWARD_gc);
        _delay_ms(100);
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
    }
    else if(strcmp(msg,"F")==0)
    {
        setMotorDuty(1,600,MOTOR_DIR_BACKWARD_gc);
        setMotorDuty(2,600,MOTOR_DIR_FORWARD_gc);
        _delay_ms(200);
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
    }
    else if(strcmp(msg,"G")==0)
    {
        setMotorDuty(1,600,MOTOR_DIR_BACKWARD_gc);
        setMotorDuty(2,600,MOTOR_DIR_FORWARD_gc);
        _delay_ms(300);
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
    }
    else if(strcmp(msg,"H")==0)
    {
        setMotorDuty(1,600,MOTOR_DIR_BACKWARD_gc);
        setMotorDuty(2,600,MOTOR_DIR_FORWARD_gc);
        _delay_ms(400);
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
    }

```

```

        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
    }
    else if(strcmp(msg,"S")==0)
    {
        setMotorDuty(1,600,MOTOR_DIR_BACKWARD_gc);
        setMotorDuty(2,600,MOTOR_DIR_BACKWARD_gc);
        _delay_ms(500);
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
    }
    else if(strcmp(msg,"X")==0)
    {
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
    }
}
}
}

```

The new collection mode is a modified version of the previous obstacle avoidance code. The robot checks the XBEE for new instructions before continuing with any obstacle avoidance or carrying out instructions. It will also update the LCD with current values of all sensors.

From here, obstacle avoidance is checked to make sure that the path is clear. Note that the obstacle avoidance code will only turn the robot and assign a variable "j" the value of 1. This was made due to problems with the robot getting caught by the camera forcing it to take the same path. The robot will now find a clear path and move forward before carrying out camera instructions to allow for a new path to be taken.

Once the robot has cleared all obstacle avoidance flags, it moves to carrying out instructions from the camera. The robot can be given instructions W, S, A, F, G, H, D, V, B, N, and X. W and S are used to move forward and backward. X is given if the robot is to sit still. Commands A, F, G, and H are given to turn left. Each command reflects an amount of degrees to turn, with A being the smallest turn and H being the largest. D, V, B, and N are used for turning right. A delay after each turn has been added in order to allow for the camera to catch up with the robot and send accurate instructions.

Originally, the robot would turn continuously in order to reach an acceptable vector with which to move forward. This caused problems due to camera lag which would cause the robot to turn too far and get caught in turning back and forth. The incremental turns will initially be slower, but should give more accuracy and allow for the robot to achieve an acceptable vector faster than before.

Camera Code

Main

The following is the main function for the laptop. This is what is looped in order to analyze the camera's live feed.

```

int main()
{
    CvCapture *capture;
    char key = 'd';
    String^ portChosen = "COM4";
    SerialPort^ _serialPort;

```

```

capture = cvCreateFileCapture("http://192.168.1.3/img/video.mjpeg");
if(!capture){
    printf("Can't initialize the video capture\n");
    return -1;
}
//cvNamedWindow("Balls",CV_WINDOW_AUTOSIZE);
//cvNamedWindow("Live",CV_WINDOW_AUTOSIZE);
//cvNamedWindow("Robot",CV_WINDOW_AUTOSIZE);
//cvMoveWindow("Live",0,100);
//cvMoveWindow("Balls",650,100);
while(key!='Q' && key!='q')
{
    image = cvQueryFrame(capture);
    IplImage *hsv = cvCloneImage(image);
    findcenters(image,hsv);
    findrobot(image,hsv);
    finddir(portChosen,_serialPort);
    key = cvWaitKey(10);
    //cvShowImage("Live",image);
}
}

```

At first, the serial port and video stream are initialized. If the video stream is not initialized, the program will automatically quit. The code that is commented out is for showing the actual live feed and filtered feeds. These are not being shown due to lag issues. However, they are left in to allow for debugging and demos. In each loop, the positions of the targets are found, followed by the robot's position and the instruction.

Finding Targets

```

void findcenters(IplImage *image, IplImage *hsv){
    cvCvtColor(image,hsv,CV_BGR2HSV);
    IplImage *mask = cvCreateImage(cvGetSize(image), image->depth,1);
    cvInRangeS(hsv, cvScalar(hb - tolerance, sb - tolerance, 0), cvScalar(hb +
tolerance, sb + tolerance, 255), mask);

    IplConvKernel *kernel = cvCreateStructuringElementEx(5, 5, 2, 2,
CV_SHAPE_ELLIPSE);

    cvDilate(mask, mask, kernel, 1);
    cvErode(mask, mask, kernel, 1);
    cvErode(mask,mask, kernel, 1);
    cvDilate(mask,mask,kernel,1);
    cvReleaseStructuringElement(&kernel);

    balls = CBlobResult(mask, NULL, 0);
    balls.Filter(balls,B_EXCLUDE,CBlobGetArea(),B_LESS,120);
    int num_blobs = balls.GetNumBlobs();

    //IplImage *filtered = cvCreateImage(cvGetSize(mask),IPL_DEPTH_8U,3);

    //cvMerge(mask,mask,mask,NULL,filtered);
    for(int i=0;i<num_blobs;i++)
    {
        currentBlob = balls.GetBlob(i);
        //currentBlob->FillBlob(filtered,CV_RGB(255,0,0));
        center[i][0] = (currentBlob->MaxX() + currentBlob->MinX())/2;
    }
}

```

```

        center[i][1] = (currentBlob->MaxY() + currentBlob->MinY())/2;
    }
    //cvShowImage("Balls",filtered);
    cvReleaseImage(&mask);
    //cvReleaseImage(&filtered);
}

```

This is the code for finding the balls. A snapshot from the live feed is taken and converted into an HSV image. The HSV image is then converted into binary using the HSV ranges for tennis balls given in the initialization section. The binary image is then filtered to reduce noise. Once filtered, blob detection is used to find the number of tennis balls in the picture. From here, the center of each tennis ball is found and stored. The commented code is used to show the actual filtered image. All tennis balls will appear as red balls, while noise is shown as a black background and white blobs.

Finding the Robot

```

void findrobot(IplImage *image, IplImage *hsv){
    cvCvtColor(image,hsv,CV_BGR2HSV);
    IplImage *mask = cvCreateImage(cvGetSize(image), image->depth,1);
    cvInRangeS(hsv, cvScalar(hr - tolerance, sr - tolerance, 50), cvScalar(hr +
tolerance, sr + tolerance, 255), mask);
    cvReleaseImage(&hsv);

    IplConvKernel *kernel = cvCreateStructuringElementEx(5, 5, 2, 2,
CV_SHAPE_ELLIPSE);

    cvDilate(mask, mask, kernel, 1);
    cvErode(mask, mask, kernel, 1);
    cvErode(mask,mask, kernel, 1);
    cvDilate(mask,mask,kernel,1);
    cvReleaseStructuringElement(&kernel);

    robot = CBlobResult(mask, NULL, 0);
    robot.Filter(robot,B_EXCLUDE,CBlobGetArea(),B_LESS,100);
    int num_blobs = robot.GetNumBlobs();

    //IplImage *filtered = cvCreateImage(cvGetSize(mask),IPL_DEPTH_8U,3);

    //cvMerge(mask,mask,mask,NULL,filtered);
    if(num_blobs==2)
    {
        robot.GetNthBlob(CBlobGetArea(),robot.GetNumBlobs()-1,cb);
        robpos[0][0] = (cb.MaxX()+cb.MinX())/2;
        robpos[0][1] = (cb.MaxY()+cb.MinY())/2;
        //cb.FillBlob(filtered,CV_RGB(255,100,0));
        robot.GetNthBlob(CBlobGetArea(),0,cb);
        robpos[1][0] = (cb.MaxX()+cb.MinX())/2;
        robpos[1][1] = (cb.MaxY()+cb.MinY())/2;
        //cb.FillBlob(filtered,CV_RGB(255,0,0));
    }
    else
    {
        for(int i=0;i<num_blobs;i++)
        {
            currentBlob = robot.GetBlob(i);
            //currentBlob->FillBlob(filtered,CV_RGB(255,0,0));
            robpos[i][0] = (currentBlob->MaxX() + currentBlob->MinX())/2;

```

```

        robpos[i][1] = (currentBlob->MaxY() + currentBlob->MinY())/2;
    }
}
//cvShowImage("Robot",filtered);
cvReleaseImage(&mask);
//cvReleaseImage(&filtered);
}

```

This code is fairly similar to finding the targets. The same filtering method is used, except small objects are filtered out instead of large objects. The robot's position is made of two blobs, a square and a triangle. These are used to determine the orientation of the robot later on. In the commented code, the filtered image is shown. The triangle will be filled in as orange and the square will be filled in as red.

Determining Instructions

```

void finddir(String^ portChosen,SerialPort^ _serialPort)
{
    int i = 0, target = -1, num = balls.GetNumBlobs(), rob = robot.GetNumBlobs();
    if(rob == 0 || rob < 2)
    {
        //std::cout<<"\nRobot Not Found  CMD: S";
        char cmp[2] = "S";
        if(strcmp(cmp,msg) != 0)
        {
            _serialPort = gcnew SerialPort(portChosen, 115200, Parity::None, 8,
StopBits::One);
            _serialPort->Open();
            _serialPort->Write("S");
            _serialPort->Close();
            strcpy(msg,cmp);
        }
    }
    else if(num==0)
    {
        //std::cout<<"\nNo Targets Found  CMD: X";
        char cmp[2] = "X";
        if(strcmp(cmp,msg) != 0)
        {
            _serialPort = gcnew SerialPort(portChosen, 115200, Parity::None, 8,
StopBits::One);
            _serialPort->Open();
            _serialPort->Write("X");
            _serialPort->Close();
            strcpy(msg,cmp);
        }
    }
    else
    {
        double dist = 0, check = 0, robang = 0, ballang = 0;
        for(i;i<num;i++)
        {
            check = sqrt(pow((center[i][0]-
(robpos[0][0]+robpos[1][0])/2),2)+pow((center[i][1]-
(robpos[0][1]+robpos[1][1])/2),2));
            if(target==-1 || check<dist)
            {
                target = i;
                dist = check;
            }
        }
    }
}

```



```

    }
    robang = (atan2(robpos[0][1]-robpos[1][1],robpos[0][0]-
robpos[1][0])*180/3.14)+180;
    ballang = (atan2(center[target][1]-robpos[1][1],center[target][0]-
robpos[1][0])*180/3.14)+180;
    if(abs(robang-ballang)<10)
    {
        //std::cout<<"\nCorrect Angle.  CMD: W";
        char cmp[2] = "W";
        if(strcmp(cmp,msg) != 0)
        {
            _serialPort = gcnew SerialPort(portChosen, 115200,
Parity::None, 8, StopBits::One);
            _serialPort->Open();
            _serialPort->Write("W");
            _serialPort->Close();
            strcpy(msg,cmp);
        }
    }
    else if(ballang - robang<0)
    {
        //std::cout<<"\nAngle too small.  CMD: A"<<ballang-robang;
        char cmp[2] = "A";
        if(abs(ballang - robang)<15){}
        else if(abs(ballang - robang)<30){strcpy(cmp,"F");}
        else if(abs(ballang - robang)<45){strcpy(cmp,"G");}
        else{strcpy(cmp,"H");}
        if(strcmp(cmp,msg) != 0)
        {
            _serialPort = gcnew SerialPort(portChosen, 115200,
Parity::None, 8, StopBits::One);
            _serialPort->Open();
            if(abs(ballang - robang)<15) {_serialPort->Write("A");}
            else if(abs(ballang - robang)<30){_serialPort->Write("F"); }
            else if(abs(ballang - robang)<45){_serialPort->Write("G"); }
            else{_serialPort->Write("H"); }
            _serialPort->Close();
            strcpy(msg,cmp);
        }
    }
    else
    {
        //std::cout<<"\nAngle too large.  CMD: D"<<ballang-robang;
        char cmp[2] = "D";
        if(abs(ballang - robang)<15){}
        else if(abs(ballang - robang)<30){strcpy(cmp,"V");}
        else if(abs(ballang - robang)<45){strcpy(cmp,"B");}
        else{strcpy(cmp,"N");}
        if(strcmp(cmp,msg) != 0)
        {
            _serialPort = gcnew SerialPort(portChosen, 115200,
Parity::None, 8, StopBits::One);
            _serialPort->Open();
            if(abs(ballang - robang)<15) {_serialPort->Write("D");}
            else if(abs(ballang - robang)<30){_serialPort->Write("V"); }
            else if(abs(ballang - robang)<45){_serialPort->Write("B"); }
            else{_serialPort->Write("N"); }
            _serialPort->Close();
        }
    }
}

```

```

        strcpy(msg, cmp);
    }
}
}

```

This code is used to determine the orientation of the robot with respect to a target and the instruction that the robot should carry out. To begin, the program will search for the robot. If the number of blobs found in the previous code is less than 2, then the robot has run off of the camera. The program sends the instruction "S" to have the robot back up until it reappears on camera. Originally the code would only check if the amount of blobs was equal to zero, which would cause problems if only the square or triangle was available.

If the robot is found, the camera searches for tennis balls. If no tennis balls have been found, the robot will be given the "X" command, which tells it to stop. This will be changed to have the camera search for a pre-determined position or object and send the robot in this direction. This will be considered a home base for the player to move to in order to collect tennis balls from the robot.

If both the robot and at least one tennis ball have been found, the distance between the robot and each tennis ball is calculated. The closest ball is determined to be the target to head towards. Once a target has been determined, the orientation of the robot is determined. This is done using the thought that the camera creates a 2D plane with three points on it: the square, the triangle, and the target. Using the x-distance and y-distance between two points to create two lines, the angle of the tangent line is found using atan2. The robot angle is found using the triangle and the square. The target angle is found using the square and the target. These angles are found in degrees.

Once both angles are found and stored, they are compared with each other. If the difference between the angles is found to be less than 10°, the robot will be given the "W" command to move forward. Otherwise, the difference is used to determine which way the robot will turn. If the target angle minus the robot angle is negative, the robot will turn left. If this difference is positive, the robot turns right.

The degree of the turn completed will be determined by how large the difference is. If the robot angle is greater than 10° but less than 15°, the "A" or "D" command will be given to perform a 5° turn. "F" and "V" are given for 15° turns. "G" and "B" are given for 30° turns. "H" and "N" are given for 45° turns.

Finally, before a message is sent via XBEE, the previously sent message is compared to the new message. If these are the same, then no message is sent. This has been done to reduce lag and keep the XBEEs from getting clogged. Originally when the sent instructions every time, some instructions would get lost and the laptop would crash from time to time. This check has highly increased stability on both the laptop and the robot.

Experimental Layout and Results

Initial Board Testing

The initial test of the microcontroller board consisted of plugging it into a power source and testing the USB power. Initially, a spare 5V power supply was used to power up the board. This was found to be inadequate and failed to power up the board. In lab, a power supply was used to test the board and load a small program that would blink the debug LED.

Initial Sonar Testing

Once soldered, the sonar was plugged into two separate devices. First, the sonar was plugged into an oscilloscope and tested to make sure that the values varied correctly. Once satisfied, the sensor was crimped and plugged into the microcontroller board. Using X-CTU to output the values of the sonar, it was found to work fine for its purposes.

Initial System Testing

Once the LCD had been wired up, a system test was implemented. The board was powered up and the values of the sonar were sent to the LCD. Once the contrast had been correctly adjusted, the code was adjusted to refresh the LCD instead of printing new values until space ran out on the LCD.

Obstacle Avoidance Initial Testing

Once the bump switches had been wired up and plugged in, the first batch of obstacle avoidance code had been tested. The wheels had not been attached at the time, but the motors were plugged in. The code would start the motors when all sensors were clear and stop the motors when one or more of the sensors dropped below their defined minimum value.

Obstacle Avoidance Beta Testing

Obstacle avoidance beta testing included finding the right values for motor duty cycle and time delays to allow for the motors to smoothly transition from moving forward to moving backward. Initially the motors were performing at 1024 duty cycle, the highest value allowed. Also, each flag reinitialized the motors to move forward when the robot finished adjusting its placement. This was found to cause unreliable obstacle avoidance and cause hard starts and stops. After adjusting values and code, the current obstacle avoidance code was made.

Dead Zone Testing

Using the calibration mode of the robot, it was left on in order to view the sonar values. The areas where the sonar value was about 400 were taped off to show the relative zone for which the robot would stop and adjust its course. Looking straight forward, it was found to be about 2 feet. Using this method, the dead zones of the sonar were found.



Figure 10: Sonar dead zone test

A good portion of these zones are not troublesome, since they are outside the width of the robot and likely will never be hit. However, there are two small triangular areas that could lead to an impact. The bump switches will be installed to cover these problem areas.

Camera Testing

Stock code was used in Visual Studio to test the camera. The camera performs well enough at low settings. It was found that the green of the tennis ball blends in with the wood panels of the robot using current filters.

XBEE Testing

Stock code was used to test the XBEE. The laptop would send a different instruction for each button pressed. XBEEs have been working fantastically.

Orientation Testing

Orientation of the robot is found using a square and a triangle. Code was written to show where a simulated robot was pointing with respect to a tennis ball. The simulated robot was moved by taping fishing line to it and pulling it around from outside the camera's view.

Integrated Orientation Testing

Tested orientation detection with the robot turned on. The wheels were suspended in midair and observed while the simulated robot moved accordingly in the camera's view. This was done to make sure that the correct instructions were being sent and that the XBEE was working in a timely manner.

Collection Testing

Two belts were connected to two separate pieces of wood. The belts were placed in parallel and in series and turned on to see if they would pick up tennis balls on a ramp. The belts worked great in parallel, but in series they lacked the torque to actually pick up tennis balls.

Conclusion

Overall, the robot has made great progress in being fully functional. It moves easily now that casters have been added and picking up balls has become easier with the addition of the arms. Changes in the camera code should make the robot perform better than ever. The largest hurdle to overcome so far has been lag from the camera. It has forced me to change the code from continuous movement to incremental movement. Additionally, the current setup for collecting tennis balls requires less precision than it previously has. The funnel allows for a larger collection area while still giving the actuator the precision it requires.

Documentation

Sonar: <http://www.sparkfun.com/products/639>

LCD: <http://www.sparkfun.com/products/9054>

Epiphany DIY Documentation: <https://sites.google.com/site/epiphanydiy/file-cabinet>

Appendices

Robot Code

This is the full code that is currently used on the robot.

```
/**
 * \file
 *
 * \brief Empty user application template
 *
 */

/*
 * Include header files for all drivers that have been imported from
 * AVR Software Framework (ASF).
 */

#include <asf.h>
#include <avr/io.h>
#include <ctype.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <util/delay.h>
#include "motor.h"
#include "lcd.h"
#include "uart.h"
#include "RTC.h"
#include "ATTinyServo.h"
```

```

#include "ADC.h"
#include "sonar.h"

#define DbLedOn()          (PORTR.OUTCLR = 0x02)           //Turns the debug led on.
The led is connected with inverted logic
#define DbLedOff()        (PORTR.OUTSET = 0x02)           //Turns the debug led off.
The led is connected with inverted logic
#define DbLedToggle()     (PORTR.OUTTGL = 0x02)           //Toggles the debug led
off. The led is connected with inverted logic
int main (void)
{
    board_init(); /*This function originates in the file init.c, and is used to
initialize the Epiphany DIY
                                motorInit() is declared within because by default you
the user should define what your
                                motor setup is to prevent hurting the Epiphany. You
can do this by
                                */
    RTC_DelayInit();//initializes the Real time clock this seems to actually take an
appreciable amount of time
    DbLedOn(); //I like to do this by default to show the board is no longer
suspended in the bootloader.
    ATtinyServoInit();//this function initializes the servo controller on the Attiny
uartInit(&USARTC0,115200);//as can be seen in the schematic. This uart is
connected to the USB port. This function initializes this uart
uartInit(&USARTE1,115200);//as can be seen in the schematic. This uart is
connected to the Xbee port. This function initializes this uart
    ADCsInits();//this function initializes the ADCs inside the Xmega
    sei();
    LCDInit();
    //stdout = &lcd_str;

    char msg = "X";
    int j = 0;

    /*while(1)
    {
        if(strcmp(msg,"S")==0)
        {
            DbLedOn();
        }
        else if(strcmp(msg,"X")==0)
        {
            DbLedOff();
        }
        if(dataInBufE1())
        {
            fscanf(&Xbee_str,"%c",msg);
        }
    }
    */
    while(analogRead_ADCA(6) > 400)
    {
        LCDSetCursor(1,1);
        fprintf(&lcd_str,"\r");
        fprintf(&lcd_str,"%d %d
%d\nCalibration",analogRead_ADCA(0),analogRead_ADCA(6),analogRead_ADCA(7));
        _delay_ms(200);
    }
}

```

```

}

setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
_delay_ms(300);

while (1)
{
    if(dataInBufE1())
    {
        fscanf(&Xbee_str,"%c",msg);
    }
    LCDSetCursor(1,1);
    fprintf(&lcd_str,"\r");
    fprintf(&lcd_str,"%d %d
%d\nCollection",analogRead_ADCA(0),analogRead_ADCA(6),analogRead_ADCA(7));
    _delay_ms(200);
    if(analogRead_ADCA(0) < 400)
    {
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
        _delay_ms(400);
        setMotorDuty(1,600,MOTOR_DIR_BACKWARD_gc);
        setMotorDuty(2,600,MOTOR_DIR_FORWARD_gc);
        _delay_ms(500);
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
        _delay_ms(300);
        j=1;
    }
    else if(analogRead_ADCA(6) < 400)
    {
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
        _delay_ms(400);
        setMotorDuty(1,600,MOTOR_DIR_BACKWARD_gc);
        setMotorDuty(2,600,MOTOR_DIR_BACKWARD_gc);
        _delay_ms(1000);
        setMotorDuty(1,600,MOTOR_DIR_FORWARD_gc);
        _delay_ms(500);
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
        _delay_ms(300);
        j=1;
    }
    else if(analogRead_ADCA(7) < 400)
    {
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
        _delay_ms(400);
        setMotorDuty(1,600,MOTOR_DIR_BACKWARD_gc);
        setMotorDuty(2,600,MOTOR_DIR_BACKWARD_gc);
        _delay_ms(1000);
        setMotorDuty(2,600,MOTOR_DIR_FORWARD_gc);
        _delay_ms(500);
        setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
        _delay_ms(300);
    }
}

```

```

        j=1;
    }
    else if(j==1)
    {
        setMotorDuty(1,600,MOTOR_DIR_FORWARD_gc);
        setMotorDuty(2,600,MOTOR_DIR_FORWARD_gc);
        _delay_ms(300);
        j=0;
    }
    else
    {
        if(strcmp(msg,"W")==0)
        {
            setMotorDuty(1,600,MOTOR_DIR_FORWARD_gc);
            setMotorDuty(2,600,MOTOR_DIR_FORWARD_gc);
        }
        else if(strcmp(msg,"D")==0)
        {
            setMotorDuty(1,600,MOTOR_DIR_FORWARD_gc);
            setMotorDuty(2,600,MOTOR_DIR_BACKWARD_gc);
            _delay_ms(100);
            setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
            setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
        }
        else if(strcmp(msg,"V")==0)
        {
            setMotorDuty(1,600,MOTOR_DIR_FORWARD_gc);
            setMotorDuty(2,600,MOTOR_DIR_BACKWARD_gc);
            _delay_ms(200);
            setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
            setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
        }
        else if(strcmp(msg,"B")==0)
        {
            setMotorDuty(1,600,MOTOR_DIR_FORWARD_gc);
            setMotorDuty(2,600,MOTOR_DIR_BACKWARD_gc);
            _delay_ms(300);
            setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
            setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
        }
        else if(strcmp(msg,"N")==0)
        {
            setMotorDuty(1,600,MOTOR_DIR_FORWARD_gc);
            setMotorDuty(2,600,MOTOR_DIR_BACKWARD_gc);
            _delay_ms(400);
            setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
            setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
        }
        else if(strcmp(msg,"A")==0)
        {
            setMotorDuty(1,600,MOTOR_DIR_BACKWARD_gc);
            setMotorDuty(2,600,MOTOR_DIR_FORWARD_gc);
            _delay_ms(100);
            setMotorDuty(1,0,MOTOR_DIR_FORWARD_gc);
            setMotorDuty(2,0,MOTOR_DIR_FORWARD_gc);
        }
        else if(strcmp(msg,"F")==0)
        {

```



```

using namespace System::Threading;

// Maths methods
#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) < (b) ? (a) : (b))
#define abs(x) ((x) > 0 ? (x) : -(x))
#define sign(x) ((x) > 0 ? 1 : -1)
// Step moving for object min & max
#define STEP_MIN 5
#define STEP_MAX 100
// HSV ball values
int hb = 167, sb = 90, vb = 230, tolerance = 40;
//HSV "robot" values. Currently pink
int hr = 173, sr = 164, vr = 230;
//array for center of all tennis balls
double center[10][10];
//robot position
double robpos[30][30];
IplImage *image;
CBlobResult balls, robot;
CBlob *currentBlob, cb;
char msg[2] = "X";

void findcenters(IplImage *image, IplImage *hsv){
    cvCvtColor(image,hsv,CV_BGR2HSV);
    IplImage *mask = cvCreateImage(cvGetSize(image), image->depth,1);
    cvInRangeS(hsv, cvScalar(hb - tolerance, sb - tolerance, 0), cvScalar(hb +
tolerance, sb + tolerance, 255), mask);

    IplConvKernel *kernel = cvCreateStructuringElementEx(5, 5, 2, 2,
CV_SHAPE_ELLIPSE);

    cvDilate(mask, mask, kernel, 1);
    cvErode(mask, mask, kernel, 1);
    cvErode(mask,mask, kernel, 1);
    cvDilate(mask,mask,kernel,1);
    cvReleaseStructuringElement(&kernel);

    balls = CBlobResult(mask, NULL, 0);
    balls.Filter(balls,B_EXCLUDE,CBlobGetArea(),B_LESS,120);
    int num_blobs = balls.GetNumBlobs();

    //IplImage *filtered = cvCreateImage(cvGetSize(mask),IPL_DEPTH_8U,3);

    //cvMerge(mask,mask,mask,NULL,filtered);
    for(int i=0;i<num_blobs;i++)
    {
        currentBlob = balls.GetBlob(i);
        //currentBlob->FillBlob(filtered,CV_RGB(255,0,0));
        center[i][0] = (currentBlob->MaxX() + currentBlob->MinX())/2;
        center[i][1] = (currentBlob->MaxY() + currentBlob->MinY())/2;
    }
    //cvShowImage("Balls",filtered);
    cvReleaseImage(&mask);
    //cvReleaseImage(&filtered);
}

```

```

void findrobot(IplImage *image, IplImage *hsv){
    cvCvtColor(image,hsv,CV_BGR2HSV);
    IplImage *mask = cvCreateImage(cvGetSize(image), image->depth,1);
    cvInRangeS(hsv, cvScalar(hr - tolerance, sr - tolerance, 50), cvScalar(hr +
tolerance, sr + tolerance, 255), mask);
    cvReleaseImage(&hsv);

    IplConvKernel *kernel = cvCreateStructuringElementEx(5, 5, 2, 2,
CV_SHAPE_ELLIPSE);

    cvDilate(mask, mask, kernel, 1);
cvErode(mask, mask, kernel, 1);
cvErode(mask,mask, kernel, 1);
cvDilate(mask,mask,kernel,1);
cvReleaseStructuringElement(&kernel);

    robot = CBlobResult(mask, NULL, 0);
    robot.Filter(robot,B_EXCLUDE,CBlobGetArea(),B_LESS,100);
    int num_blobs = robot.GetNumBlobs();

    //IplImage *filtered = cvCreateImage(cvGetSize(mask),IPL_DEPTH_8U,3);

    //cvMerge(mask,mask,mask,NULL,filtered);
    if(num_blobs==2)
    {
        robot.GetNthBlob(CBlobGetArea(),robot.GetNumBlobs()-1,cb);
        robpos[0][0] = (cb.MaxX()+cb.MinX())/2;
        robpos[0][1] = (cb.MaxY()+cb.MinY())/2;
        //cb.FillBlob(filtered,CV_RGB(255,100,0));
        robot.GetNthBlob(CBlobGetArea(),0,cb);
        robpos[1][0] = (cb.MaxX()+cb.MinX())/2;
        robpos[1][1] = (cb.MaxY()+cb.MinY())/2;
        //cb.FillBlob(filtered,CV_RGB(255,0,0));
    }
    else
    {
        for(int i=0;i<num_blobs;i++)
        {
            currentBlob = robot.GetBlob(i);
            //currentBlob->FillBlob(filtered,CV_RGB(255,0,0));
            robpos[i][0] = (currentBlob->MaxX() + currentBlob->MinX())/2;
            robpos[i][1] = (currentBlob->MaxY() + currentBlob->MinY())/2;
        }
        //cvShowImage("Robot",filtered);
        cvReleaseImage(&mask);
        //cvReleaseImage(&filtered);
    }
}

void finddir(String^ portChosen,SerialPort^ _serialPort)
{
    int i = 0, target = -1, num = balls.GetNumBlobs(), rob = robot.GetNumBlobs();
    if(rob == 0 || rob < 2)
    {
        //std::cout<<"\nRobot Not Found  CMD: S";
        char cmp[2] = "S";
        if(strcmp(cmp,msg) != 0)
        {

```



```

else if(abs(ballang - robang)<30){strcpy(cmp,"F");}
else if(abs(ballang - robang)<45){strcpy(cmp,"G");}
else{strcpy(cmp,"H");}
if(strcmp(cmp,msg) != 0)
{
    _serialPort = gcnew SerialPort(portChosen, 115200,
Parity::None, 8, StopBits::One);
    _serialPort->Open();
    if(abs(ballang - robang)<15) {_serialPort->Write("A");}
    else if(abs(ballang - robang)<30){_serialPort->Write("F"); }
    else if(abs(ballang - robang)<45){_serialPort->Write("G"); }
    else{_serialPort->Write("H"); }
    _serialPort->Close();
    strcpy(msg,cmp);
}
}
else
{
    //std::cout<<"\nAngle too large. CMD: D"<<ballang-robang;
    char cmp[2] = "D";
    if(abs(ballang - robang)<15){}
    else if(abs(ballang - robang)<30){strcpy(cmp,"V");}
    else if(abs(ballang - robang)<45){strcpy(cmp,"B");}
    else{strcpy(cmp,"N");}
    if(strcmp(cmp,msg) != 0)
    {
        _serialPort = gcnew SerialPort(portChosen, 115200,
Parity::None, 8, StopBits::One);
        _serialPort->Open();
        if(abs(ballang - robang)<15) {_serialPort->Write("D");}
        else if(abs(ballang - robang)<30){_serialPort->Write("V"); }
        else if(abs(ballang - robang)<45){_serialPort->Write("B"); }
        else{_serialPort->Write("N"); }
        _serialPort->Close();
        strcpy(msg,cmp);
    }
}
}

int main()
{
    CvCapture *capture;
    char key = 'd';
    String^ portChosen = "COM4";
    SerialPort^ _serialPort;

    capture = cvCreateFileCapture("http://192.168.1.3/img/video.mjpeg");
    if(!capture){
        printf("Can't initialize the video capture\n");
        return -1;
    }
    //cvNamedWindow("Balls",CV_WINDOW_AUTOSIZE);
    //cvNamedWindow("Live",CV_WINDOW_AUTOSIZE);
    //cvNamedWindow("Robot",CV_WINDOW_AUTOSIZE);
    //cvMoveWindow("Live",0,100);
    //cvMoveWindow("Balls",650,100);
    while(key!='Q' && key!='q')

```

```

    {
        image = cvQueryFrame(capture);
        IplImage *hsv = cvCloneImage(image);
        findcenters(image,hsv);
        findrobot(image,hsv);
        finddir(portChosen,_serialPort);
        key = cvWaitKey(10);
        //cvShowImage("Live",image);
    }
}

```

HSV Color Code

This code is used to determine the HSV values of any object. Just run the code and click on the object that is in camera view. The terminal will spit out the HSV value of this object. This code is a slight modification of the sample code given by Josh Weaver.

```

#include "opencv/highgui.h"
#include "opencv/cv.h"
#include <iostream>
#include <stdlib.h>
#include <stdio.h>

// Maths methods
#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) < (b) ? (a) : (b))
#define abs(x) ((x) > 0 ? (x) : -(x))
#define sign(x) ((x) > 0 ? 1 : -1)
// Step moving for object min & max
#define STEP_MIN 5
#define STEP_MAX 100
IplImage *image;
// Position of the object we overlay
CvPoint objectPos = cvPoint(-1, -1);
// Color tracked and our tolerance towards it
int h = 118, s = 131, v = 166, tolerance = 40;
/*
 * Transform the image into a two colored image, one color for the
 * color we want to track, another color for the others colors
 * From this image, we get two datas : the number of pixel detected,
 * and the center of gravity of these pixel
 */
CvPoint binarisation(IplImage* image, int *nbPixels) {
    int x, y;
    CvScalar pixel;
    IplImage *hsv, *mask;
    IplConvKernel *kernel;
    int sommeX = 0, sommeY = 0;
    *nbPixels = 0;
    // Create the mask & initialize it to white (no color detected)
    mask = cvCreateImage(cvGetSize(image), image->depth, 1);
    // Create the hsv image
    hsv = cvCloneImage(image);
    cvCvtColor(image, hsv, CV_BGR2HSV);
    // We create the mask
    cvInRangeS(hsv, cvScalar(h - tolerance -1, s - tolerance, 0), cvScalar(h +
tolerance -1, s + tolerance, 255), mask);

```

```

// Create kernels for the morphological operation
kernel = cvCreateStructuringElementEx(5, 5, 2, 2, CV_SHAPE_ELLIPSE);
// Morphological opening (inverse because we have white pixels on black
background)
cvDilate(mask, mask, kernel, 1);
cvErode(mask, mask, kernel, 1);
    cvErode(mask, mask, kernel, 1);
    cvDilate(mask, mask, kernel, 1);
// We go through the mask to look for the tracked object and get its gravity
center
for(x = 0; x < mask->width; x++) {
    for(y = 0; y < mask->height; y++) {
        // If its a tracked pixel, count it to the center of gravity's
calcul
        if(((uchar *)(mask->imageData + y*mask->widthStep))[x] == 255) {
            sommeX += x;
            sommeY += y;
            (*nbPixels)++;
        }
    }
}
// Show the result of the mask image
cvShowImage("Mask", mask);
// We release the memory of kernels
cvReleaseStructuringElement(&kernel);
// We release the memory of the mask
cvReleaseImage(&mask);
// We release the memory of the hsv image
cvReleaseImage(&hsv);
// If there is no pixel, we return a center outside the image, else we return the
center of gravity
if(*nbPixels > 0)
    return cvPoint((int)(sommeX / (*nbPixels)), (int)(sommeY / (*nbPixels)));
else
    return cvPoint(-1, -1);
}

/*
 * Add a circle on the video that follow your colored object
 */
void addObjectToVideo(IplImage* image, CvPoint objectNextPos, int nbPixels) {
    int objectNextStepX, objectNextStepY;
    // Calculate circle next position (if there is enough pixels)
    if (nbPixels > 10) {
        // Reset position if no pixel were found
        if (objectPos.x == -1 || objectPos.y == -1) {
            objectPos.x = objectNextPos.x;
            objectPos.y = objectNextPos.y;
        }
        // Move step by step the object position to the desired position
        if (abs(objectPos.x - objectNextPos.x) > STEP_MIN)
            {
                objectNextStepX = max(STEP_MIN, min(STEP_MAX, abs(objectPos.x -
objectNextPos.x) / 2));
                objectPos.x += (-1) * sign(objectPos.x - objectNextPos.x) *
objectNextStepX;
            }
        if (abs(objectPos.y - objectNextPos.y) > STEP_MIN) {

```

```

        objectNextStepY = max(STEP_MIN, min(STEP_MAX, abs(objectPos.y -
objectNextPos.y) / 2));
        objectPos.y += (-1) * sign(objectPos.y - objectNextPos.y) *
objectNextStepY;
    }
    // -1 = object isn't within the camera range
} else {
    objectPos.x = -1;
    objectPos.y = -1;
}
// Draw an object (circle) centered on the calculated center of gravity
if (nbPixels > 10)
    cvDrawCircle(image, objectPos, 15, CV_RGB(255, 0, 0), -1);
// We show the image on the window
cvShowImage("Color Tracking", image);
}

/*
 * Get the color of the pixel where the mouse has clicked
 * We put this color as model color (the color we want to tracked)
 */
void getObjectColor(int event, int x, int y, int flags, void *param = NULL) {
    // Vars
    CvScalar pixel;
    IplImage *hsv;
    if(event == CV_EVENT_LBUTTONDOWN) {
        // Get the hsv image
        hsv = cvCloneImage(image);
        cvCvtColor(image, hsv, CV_BGR2HSV);
        // Get the selected pixel
        pixel = cvGet2D(hsv, y, x);
        // Change the value of the tracked color with the color of the selected
pixel
        h = (int)pixel.val[0];
        s = (int)pixel.val[1];
        v = (int)pixel.val[2];
        // Release the memory of the hsv image
        cvReleaseImage(&hsv);
    }
}

int main() {
    // Image & hsvImage
    IplImage *hsv;
    // Video Capture
    CvCapture *capture;
    // Key for keyboard event
    char key = 'd';
    // Number of tracked pixels
    int nbPixels;
    // Next position of the object we overlay
    CvPoint objectNextPos;
    // Initialize the video Capture (200 => CV_CAP_V4L2)
    //capture = cvCaptureFromCAM(1);
    capture = cvCreateFileCapture("http://192.168.1.3/img/video.mjpeg");
    // Check if the capture is ok
    if (!capture) {
        printf("Can't initialize the video capture.\n");
    }
}

```



```

        return -1;
    }
    // Create the windows
    cvNamedWindow("Color Tracking", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("Mask", CV_WINDOW_AUTOSIZE);
    cvMoveWindow("Color Tracking", 0, 100);
    cvMoveWindow("Mask", 650, 100);
    // Mouse event to select the tracked color on the original image
    cvSetMouseCallback("Color Tracking", getObjectColor);
    // While we don't want to quit
    while(key != 'Q' && key != 'q') {
        // We get the current image
        image = cvQueryFrame(capture);
        // If there is no image, we exit the loop
        if(!image)
            continue;
        objectNextPos = binarisation(image, &nbPixels);
        addObjectToVideo(image, objectNextPos, nbPixels);
        // We wait 10 ms
        key = cvWaitKey(10);
        std::cout<<"h = "<<h<<" s = "<<s<<" v = "<<v<<"\n";
    }
    // Destroy the windows we have created
    cvDestroyWindow("Color Tracking");
    cvDestroyWindow("Mask");
    // Destroy the capture
    cvReleaseCapture(&capture);
    return 0;
}

```

XBEE Serial Port Test

This code was found on the Google group for IMDL. This is used to send a message from the laptop to the robot. This was used to make sure the XBEEs were operating, test how to manage messages, and find the optimal delay to allow for precision turning.

```

#pragma once
namespace Serial_Port_Tester {
    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;
    using namespace System::IO::Ports;
    /// <summary>
    /// Summary for Form1
    /// </summary>
    public ref class Form1 : public System::Windows::Forms::Form
    {
    public:
        SerialPort^ _serialPort; //global serial port variable
    private: System::Windows::Forms::Button^ Start_Robot;
    private: System::Windows::Forms::Button^ Stop_Robot;
    public:
        int x;
    }
}

```

```

Form1(void)
{
    InitializeComponent();
    //
    //TODO: Add the constructor code here
    //
}
protected:
    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    ~Form1()
    {
        if (components)
        {
            delete components;
        }
    }
private: System::Windows::Forms::Button^ List_Ports;
protected:
private: System::Windows::Forms::ListBox^ Ports_List;
private:
    /// <summary>
    /// Required designer variable.
    /// </summary>
    System::ComponentModel::Container ^components;
#pragma region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    void InitializeComponent(void)
    {
        this->List_Ports = (gcnew System::Windows::Forms::Button());
        this->Ports_List = (gcnew System::Windows::Forms::ListBox());
        this->Start_Robot = (gcnew System::Windows::Forms::Button());
        this->Stop_Robot = (gcnew System::Windows::Forms::Button());
        this->SuspendLayout();
        //
        // List_Ports
        //
        this->List_Ports->Location = System::Drawing::Point(35, 138);
        this->List_Ports->Name = L"List_Ports";
        this->List_Ports->Size = System::Drawing::Size(75, 23);
        this->List_Ports->TabIndex = 0;
        this->List_Ports->Text = L"List Ports";
        this->List_Ports->UseMnemonic = false;
        this->List_Ports->UseVisualStyleBackColor = true;
        this->List_Ports->Click += gcnew System::EventHandler(this,
&Form1::List_Ports_Click);
        //
        // Ports_List
        //
        this->Ports_List->FormattingEnabled = true;
        this->Ports_List->Location = System::Drawing::Point(12, 12);
        this->Ports_List->Name = L"Ports_List";
        this->Ports_List->Size = System::Drawing::Size(120, 95);
        this->Ports_List->TabIndex = 1;
    }

```

```

//
// Start_Robot
//
this->Start_Robot->Location = System::Drawing::Point(153, 12);
this->Start_Robot->Name = L"Start_Robot";
this->Start_Robot->Size = System::Drawing::Size(75, 23);
this->Start_Robot->TabIndex = 2;
this->Start_Robot->Text = L"Start Robot";
this->Start_Robot->UseVisualStyleBackColor = true;
this->Start_Robot->Click += gcnew System::EventHandler(this,
&Form1::Start_Robot_Click);
//
// Stop_Robot
//
this->Stop_Robot->Location = System::Drawing::Point(153, 41);
this->Stop_Robot->Name = L"Stop_Robot";
this->Stop_Robot->Size = System::Drawing::Size(75, 23);
this->Stop_Robot->TabIndex = 3;
this->Stop_Robot->Text = L"Stop Robot";
this->Stop_Robot->UseVisualStyleBackColor = true;
this->Stop_Robot->Click += gcnew System::EventHandler(this,
&Form1::Stop_Robot_Click);
//
// Form1
//
this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(261, 187);
this->Controls->Add(this->Stop_Robot);
this->Controls->Add(this->Start_Robot);
this->Controls->Add(this->Ports_List);
this->Controls->Add(this->List_Ports);
this->Name = L"Form1";
this->RightToLeft = System::Windows::Forms::RightToLeft::No;
this->Text = L"Serial Port Tester";
this->ResumeLayout(false);
}
#pragma endregion
private: System::Void List_Ports_Click(System::Object^ sender,
System::EventArgs^ e) {
    Ports_List->Items->Clear();
    array<String^> serialPorts = nullptr;
    try
    {
        serialPorts = SerialPort::GetPortNames();
    }
    catch (Win32Exception^ ex)
    {
        MessageBox::Show(ex->Message);
    }
    for each(String^ port in serialPorts)
    {
        Ports_List->Items->Add(port);
    }
}
private: System::Void Start_Robot_Click(System::Object^ sender,
System::EventArgs^ e) {

```

```

        if (Ports_List->SelectedIndex == -1)
        {
            MessageBox::Show("Please select a serial
port.");
        }
        else
        {
            String^ portChosen = Ports_List->SelectedItem-
>ToString();
            _serialPort = gcnew SerialPort(portChosen,
115200, Parity::None, 8, StopBits::One);
            _serialPort->Open();
            _serialPort->Write("W");
            _serialPort->Close();
        }
    }
private: System::Void Stop_Robot_Click(System::Object^ sender, System::EventArgs^ e) {
    if (Ports_List->SelectedIndex == -1)
    {
        MessageBox::Show("Please select a serial
port.");
    }
    else
    {
        String^ portChosen = Ports_List->SelectedItem-
>ToString();
        _serialPort = gcnew SerialPort(portChosen,
115200, Parity::None, 8, StopBits::One);
        _serialPort->Open();
        _serialPort->Write("X");
        _serialPort->Close();
    }
}
};
}

```