

The Shop-Mulinator

Name: Marina Yokota

Robot Name: Shop-Mulinator

EEL 4665/5666: Intelligent Machines Design Laboratory (IMDL)

TAs: Tim Martin and Ryan Stevens

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

12/6/2011

Table of Contents

1	Abstract.....	2
2	Executive Summary.....	3
3	Introduction	4
4	Integrated System	5
5	Mobile Platform.....	6
6	Actuation	7
6.1	Driving and Steering	7
6.2	DC Motor and ESC	7
6.3	Servo.....	8
7	Sensors.....	9
7.1	SharpIR Sensors	9
7.2	Bumper Switches.....	12
7.3	Webcam.....	14
8	Lessons Learned	18
9	Conclusion.....	19
10	Documentation.....	20
11	Appendices.....	21

1 **Abstract**

The Shop-Mulinator is an autonomous mule robot designed to follow and carry a small, medium, or large amount of items while the user or shopper is shopping. The robot should be able to avoid walls and objects that are in its way. It also needs to be able to follow the user where ever they go indoors. Finally, since this robot is used for shopping, being able to carry a weight is a necessity specification for the Shop-Mulinator.

2 Executive Summary

This report has detailed information about a robot named Shop-Mulinator. The Shop-Mulinator was named after its' purpose, which is a mule used for shopping. This mule has 3 main objectives that it needs to accomplish, and they are accomplished by using sensors such as the Sharp IR sensors, bumper switches, and a webcam, as well as components such as shocks. The 3 main objectives are to avoid obstacles, follow a user, and withstand the shopping weight.

The Shop-Mulinator is essentially a metal framed box shielding the mother board with a 4 wheel base and a shopping basket placed on the very top. The metal frame was chosen to withstand the weight of the shopping items. Extra shocks were installed to enforce it as well.

For obstacle avoidance, the Sharp IR sensors are used to detect walls and the bumper switches are used to kick start a motor as well as avoid any obstacles that surpass the IR sensors. Finally the webcam is used for color tracking of the user (shopper).

The major issue with the robot was when dealing with the ESC for the motor. Since it is a heavy robot, it needed a strong enough motor, so it using a RC car DC motor. The RC car DC motor is very fast, so to control the speed better, the ESC was used also.

By spending a lot of time tweaking the ESC and motor, the Shop-Mulinator was finally complete. It completes obstacle avoidance very successfully. The color tracking on the robot is very accurate and clean. Finally the mule can withstand a lot of weight. Therefore the robot is a success and can accomplish all its' purposes.

3 Introduction

Shopping is one of the most common activities in today's society. No matter where a person travels or reside in the world, goods are always being bought every second. Some carry many shopping bags and some less than others but everyone has to carry something when they make a purchase. Imagine how much easier shopping would be with an automated robot carrying all the weight. The economy might have a slight or significant increase due to shoppers buying more items worry free of the weight. Well that distant future of that possibility is not so far. The "Shop-Mulinator" is the future of shopping made easy. It is designed to follow a selected user or shopper without any remote control enabling the shopper to concentrate more on shopping and less on worrying about the carrying amount of the purchased items.

4 Integrated System

The Shop-Mulinator will consist of the following:

1. Epiphany DIY Circuit Board
2. Sensors
3. 4 Wheeled Build
4. Shopping Basket

The basic Epiphany DIY circuit board will be used as the motherboard for the robot. It includes the ATMEGA64A1 Microcontroller from Atmel, ATMEGA8U2 also from Atmel and PIC16F1825. The board also includes 4 different power sources available; one for the motor, one for the main power source, another for the servo, and the power for the USB port. For feedback, there are 4 LEDs mounted on the board to indicate power from the main source, USB plug, debugging process, and programming process. It also comes with an Op amp for voltage regulation and finally a reset button to erase the board of any codes.

For the Shop-Mulinator to function accordingly, it will need to have some sensors. The sensors include: SharpIR sensors, bumper switches, and webcam for sensors.

The Epiphany DIY circuit board with all the wiring will be at the bottom layer of the Shop-Mulinator, which will be enclosed in an aluminum framing. The circuit container will be on 4 medium sized wheels running on one actuator with one servo for the front wheels. There will be shocks installed on top of the enclosed box to with stand the weight of the shopping items.

Finally the shopping basket will be placed on top of the aluminum frame. The shopping basket will not be attached permanently, but instead will have a socket to be placed in and to keep stable.

5 Mobile Platform

The main concern for the Shop-Mulinator's build is how it needs to be able to withstand the weight from all the shopping items. To overcome the weight, the mobile platform is made out of aluminum frame with steel brackets fitted on so that a shopping basket can be placed on the top. The circuit board is embedded within the aluminum frame and then the whole body is raised up with wheels and shocks.

The aluminum frame for the robot is from a monster truck by Redcat. Since a specific computer aided machine is needed to cut out a metal sheet strong enough, the monster truck frame was the sufficient way to go. Since the aluminum frame is made for a monster truck, there is already a place for a battery, but there still needs to be a place for a circuit board and of course, the shopping basket.

The circuit board is raised on top of a steel platform equipped from Lowe's. Finally, on top of the aluminum frame and the circuit board, pieces of steel brackets were put together to make a tray like platform to hold the shopping basket in place. These steel brackets are also from Lowe's.

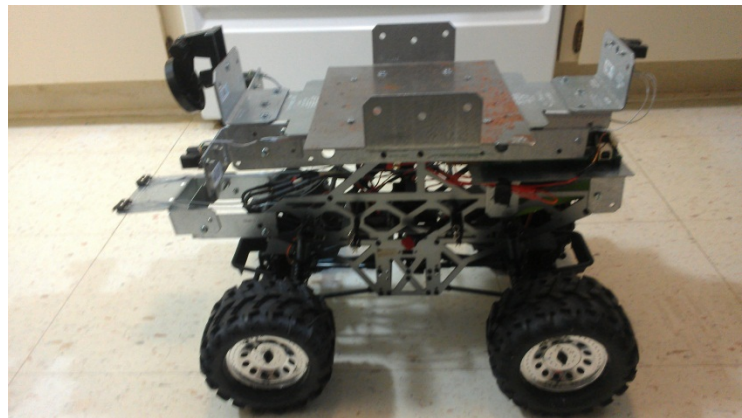


Image 5.1 what the robot looks like with all the aluminum frame, steel brackets placed in the right position. In Image 5.2 is what the robot looks like in a complete form with the basket on top.



6 Actuation

The motor that will be used on this robot is a monster truck DC motor. Since the Shop-Mulinator's build is made of aluminum and steel frames, an Epiphany DIY kit for a DC motor is too small and too weak. Another alternative was to use a car window actuator, but the actuator was too large for the frame size.

6.1 Driving and Steering

The steering system goes from the ESC to the motor, which has gears for differentials and runs to the 4 wheels. Each pairs of wheels, 1 on the front and 1 on the back are connected by 1 servo each. The ESC controls the motor's speed and direction and the servo control the turning. For the Shop-Mulinator, the front wheel is the only one with the servo connected. This is to keep the programming simpler. Image 6.1 shows the driving system for the robot.



6.2 DC Motor and ESC

The remote controlled car's DC motor comes with an ESC, an electronic speed controller. This ESC is from CastleLink and comes with a programmer for setting adjustments such as speed or adding reverse mode. Since the monster truck DC motor is too fast, the ESC is an important component to the robot in order to control the speed of the movement.

The coding for the motor is done by using a Pulse Width Modulation, PWM for short. Code 6.2 shows the set up for the PWM. To set up the PWM, one must know the prescaler and the top value for compare and capture. The prescaler and the top value are found by using the main microprocessor's system clock speed, which is 32MHz for the Epiphany DIY board. Finally the timer of the ESC's port must be set to the neutral point, which is a value, 745, for this specific ESC. As the values for the timer changes for the motor, the speed also changes. The larger the number, the faster the motor will rotate. The smaller the number, the faster the motor will rotate in reverse mode.


```

/*****
*The below initializes the servo and the motor. The ESC for the motor works the same way as the servo
*This is setting up for Pulse Width Modulation.
*32MHz=system clock
*timer toggles 0-2^16 = 65536 (overflow 20ms=50Hz); this is one window, for this board
*how many CPU cycles in 32M/50=clocks per window = 640000
*640000 is way bigger than 65536 so scale it down by dividing it by 64 (trial an error)
*640000/64 is good, but the timers do not have prescaler of 16, they have 64 so use 64
*timer initialization for motor=Port C0, Port D0 for front servo
*****/
PORTC_DIR = 0x0F; //set portc0 to output breakdown-->0000 11-for usb output, 11-for pin0 and 1
PORTD_DIR = 0x01; //set portd0 to output breakdown-->0000 0001 for pin0

TCC0_PER = 10000; //the top value if you divide 640000/64
TCD0_PER = 10000;

TCC0_CTRLA = 0x05; //prescaler is 64
TCD0_CTRLA = 0x05;

TCD0_CCA = 750; //the window for PWM window is 1ms to 2ms
//compare to 0 to 10000 count --> half way 1.5ms = 7500
//move a decimal place over so its 750.
//set servo to an angle to 0 (1.5ms mark) for initialization
TCC0_CCA = 745; //set the number corresponding to neutral for motor. same as 1.5ms
//look up on the esc programmer
//this initializes the motor at stop to pass through arming time for esc

TCC0_CTRLB = 0x13; // select capture compare for PortC & D and single slope wave
TCD0_CTRLB = 0x13;

```

Code 6.2

6.3 Servo

This robot is a front wheel drive, with 1 standard servo controlling the turning of the 2 front wheels. The servo's turning range is 90° to -90° and is also programmed by the PWM, but instead of 745 being the stop time, the servo is set to 750 for 0°. As shown in Code 6.2, there are 2 repeated set ups for different ports. Port C pin 0 is for the motor and Port D pin 0 is for the servo.



As the values for the timer changes, the servos are set to different angles for turning. The smaller the timer value the more negative the angle, and the larger the timer value the more positive the angle will be. In Image 6.3 is a picture of the servo used in the mule.

In the Code 6.4, the timer values are converted into angle values to make more sense of the timer values. Since the servos only go as far as 90° to -90°, the code also has limitations to keep the angle bounds within the servo's capability.

```

/*****
*This is a separate function which bound the upper and lower angle limit, convert the angle to timer count,
*and set the servo angle in the timer counter PortD Pin 0
*****/
void frontservoangle(int angle)
{
    if (angle>90){ //limit upper angle bound to 90
        angle = 90;
    }
    else if (angle<-90){ //limit lower angle bound to -90
        angle = -90;
    }

    angle *= 5; //converting angle degrees to timer count number
    angle /= 2;
    TCD0_CCA = 750 + angle; //set the servo to the specified angle
}

```

Code 6.4

7 Sensors

The sensors on the Shop-Mulinator consist of 3 main types; IR sensors, bumper switches, and a webcam. The robot will have 3 IR sensors. The Sharp IR sensors will be used to detect any walls around the robot. Another sensor will be bumper switches, and there will be 2 bumper switches in each side of the front panel to sense any objects that are in its path way. A webcam will be used to track the user. For the camera to follow, the user will need to be wearing a brightly unique color.

7.1 SharpIR Sensors

One of the main goals for the “Shop-Mulinator” is to accomplish obstacle avoidance. To do so the mule is equipped with 3 Sharp IR sensors. Since the “Shop-Mulinator” is larger in scale, a 20 to 150cm range IR sensor is preferred. There are 2 IR sensors in the front and 1 in the back of the mule.

The IR sensors are connected through the analog to digital ports on the microprocessor board. Since the IR sensors use a 5V input, a voltage regulator has been added in addition to the microprocessor board. The circuit for the regulator is shown in Figure 7.1. The voltage regulator will take in 9V from the main power supply and make it output a 5V which is then connected to a servo power pin to the rest of the board. The signals are read in from the IR sensors into the board and displayed on the laptop screen in decimal values. The higher the value the closer the wall is to the sensor. Figure 7.2 shows the test coding for the IR sensor.

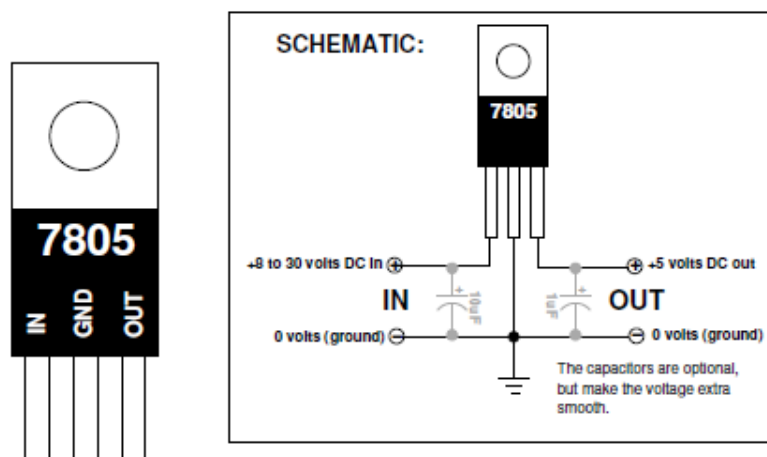


Figure 7.1

The code starts out with files to include and define functions. In the main code, there are initializations for the ADC ports. The IR sensors are in ADC port A pins 0 to 2 and reading values off of these pins will display numbers in decimal on the laptop screen. The number in decimal corresponds to the distance between the obstacle and the IR sensor.

```

10 | * AVR Software Framework (ASF).
11 | */
12 | #include <asf.h>
13 | #include <avr/io.h>
14 | #include <ctype.h>
15 | #include <stdint.h>
16 | #include <stdio.h>
17 | #include <util/delay.h>
18 | #include <math.h>
19 | #include "motor.h"
20 | #include "lcd.h"
21 | #include "uart.h"
22 | #include "RTC.h"
23 | #include "picServo.h"
24 | #include "ADC.h"
25 | //#include "sonar.h"
26 |
27 | #define DbLedOn()      (PORTR.OUTCLR = 0x02)      //Turns the debug led on. The led is connected with inverted logic
28 | #define DbLedOff()    (PORTR.OUTSET = 0x02)      //Turns the debug led off. The led is connected with inverted logic
29 | #define DbLedToggle() (PORTR.OUTTGL = 0x02)      //Toggles the debug led off. The led is connected with inverted logic
30 | //#define RAND_MAX
31 |
32 | int main (void)
33 | {
34 |     board_init(); /*This function originates in the file init.c, and is used to initialize the Epiphany DIY
35 |                  motorInit() is declared within because by default you the user should define what your
36 |                  motor setup is to prevent hurting the Epiphany. You can do this by
37 |                  */
38 |     RTC_DelayInit();//initializes the Real time clock this seems to actually take an appreciable amount of time
39 |     DbLedOn(); //I like to do this by default to show the board is no longer suspended in the bootloader.
40 |     servoControlInit();//initializes the servo module ***including enabling global interrupts*** required for the servo con
41 |     uartInit(&USARTC0,57600);//as can be seen in the schematic. This uart is connected to the USB port. This function ini
42 |     ADCsInits();
43 |     LCDInit();
44 |     motorInit();
45 |     stdout = &lcd_str;//This function points stdout to the LCD. So this means printf will send data out the LCD.
46 |
47 |     while (1){
48 |         asm("nop");
49 |
50 |         ADCA_request(0, ExternalChannel0); //front IR
51 |         ADCA_request(1, ExternalChannel1); //front IR
52 |         ADCA_request(2, ExternalChannel2); //back IR
53 |
54 |         if (ADCA_CH0_ConvComplete==1, ADCA_CH1_ConvComplete==1, ADCA_CH2_ConvComplete==1){ //if the flag for ch0 is true
55 |
56 |             uint16_t front1 = ADCA_getVal(0); //get value of the front IR sensor
57 |             uint16_t front2 = ADCA_getVal(1); //get value of the second front IR sensor
58 |             uint16_t back1 = ADCA_getVal(2); //get value of the IR sensor in the back
59 |
60 |             printf("front1 = %d, front2 = %d, back = %d\r\n",front1, front2, back1);
61 |         }
62 |         _delay_ms(150);
63 |     }
64 | }
65 |
66 |
67 |
68 |
69 |
70 |
71 |
72 |
73 |
74 |
75 |
76 |
77 |
78 |
79 |
80 |
81 |
82 |
83 |
84 |
85 |
86 |
87 |
88 |
89 |
90 |
91 |
92 |
93 |
94 |
95 |
96 |
97 |
98 |
99 |
100|
101|
102|
103|
104|
105|
106|
107|
108|
109|
110|
111|
112|
113|
114|
115|
116|
117|
118|
119|
120|
121|
122|
123|
124|
125|
126|
127|
128|
129|
130|
131|
132|
133|
134|
135|
136|
137|
138|
139|
140|
141|
142|
143|
144|
145|
146|
147|
148|
149|
150|
151|
152|
153|
154|
155|
156|
157|
158|
159|
160|
161|
162|
163|
164|
165|
166|
167|
168|
169|
170|
171|
172|
173|
174|
175|
176|
177|

```

Code 7.2

For this particular robot, these IR sensors are designed to detect mainly walls. To see how well these sensors detect white walls, an experiment using white sheet of paper was used to produce the data in Table 7.3. Figure 7.4 shows the data with distance on the x axis and decimal values on the y axis.

Distance (cm)	Decimal Values
10	1100
20	2260
30	1870
40	1460
50	1220
60	1020
70	905
80	820
90	750
100	690
110	650
120	620
130	600
140	580
150	580
160	580

Table 7.3

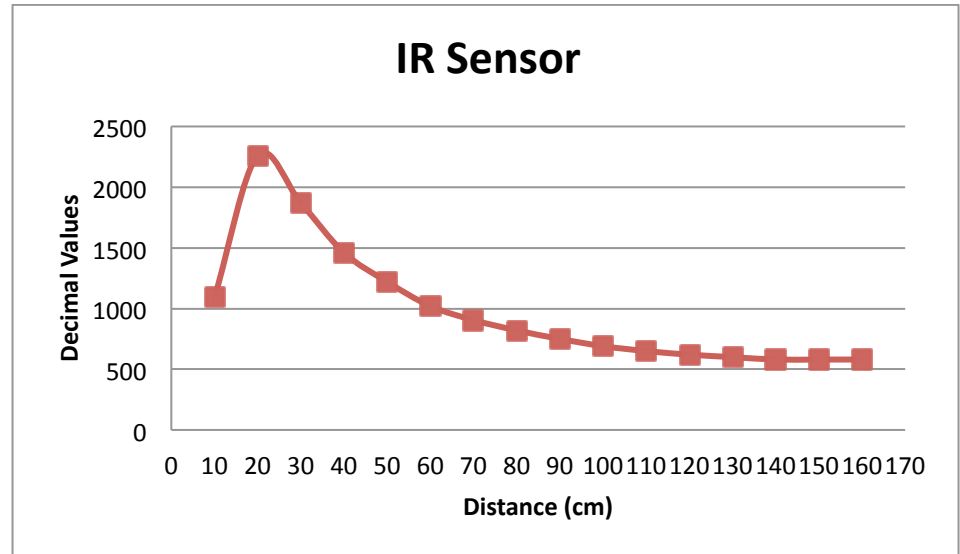


Figure 7.4

As shown in Figure 7.4, the IR sensor will respond best between the distances of 20 cm to about a 100cm. If the robot uses the distance in the range of 100cm to 150cm, it will have a difficult time detecting exactly how far the wall is therefore it will not be a very accurate reading. Similarly, the IR sensor is not designed to detect anything closer than 20cm, so it will be best to avoid using these IR sensors for detecting obstacles in close distance below 20cm.

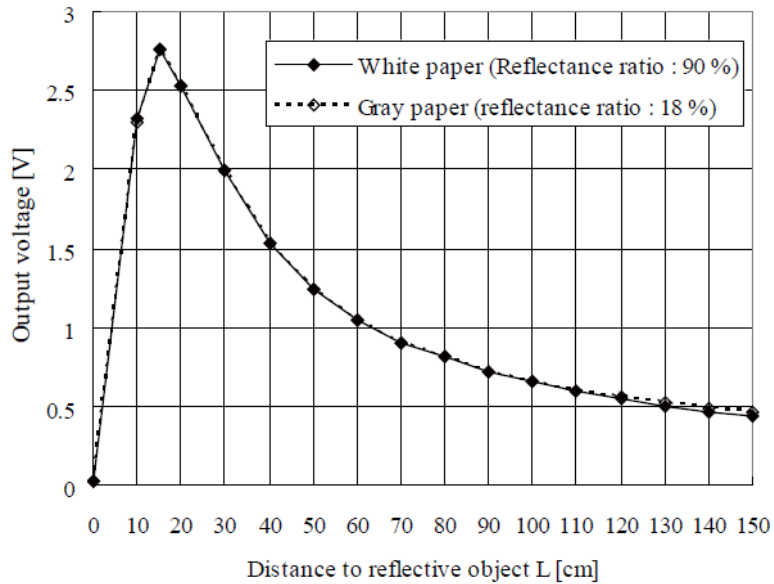


Figure 7.5

Figure 7.5 is the actual data from the Sharp IR sensor. Compared to the Figure 7.4, which is the experimental data, the 2 graphs are very similar. As Figure 7.5 shows, the IR sensors respond better with white color than any other color, hence the experiment was done with white sheet of paper to show best results.

7.2 Bumper Switches

The second type of sensors for the robot to accomplish obstacle avoidance is the bumper switches, also known as micro switches. These micro switches are used to detect object that surpassed the IR sensors. Since the IR sensors cannot detect anything closer than 20cm, the robot needs to have another sensor to detect obstacles close by. For detecting objects closer than 20cm, the robot has 2 micro switches installed in the front extension. The micro switches has 3 connections; common, normally open, and normally closed. The pin, normally closed, is connected to ground on the voltage regulator and the pin, normally open, is connected to the 5V out of the voltage regulator. The common pin is connected to the signal pin on Epiphany DIY board Port D pin 1 and for the other bumper switch the common is connected to Port D pin 2.

14.3.3 Pull-up

Figure 14-3. I/O configuration - Totem-pole with pull-up (on input)

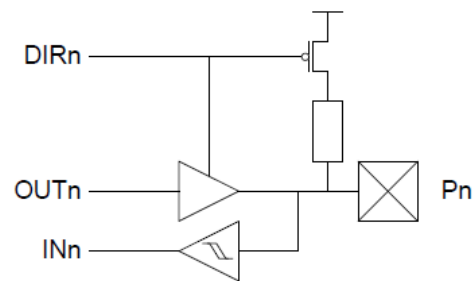


Figure 7.6

As shown in Figure 4.1, the ATXMEGA64A1 has ready to use pull-up configurations built in internally in the chip. The code for the micro switch will be simply comparing whether the input of the switch is 1 or 0. If it is 1, then it means the switch has been pressed and there is an object in the way. If it is 0 on the input, then it means the switch has not been pressed and it is clear to drive forward.

```

96  /******
97  *The bumpers are connected to the PORTD pins. 2 at the front of the robot.
98  *NO (normally open=green) = 5V, NC (normally closed=brown) = gnd, C (common=red) = signal
99  *PORTD Pin 1 and 2 are for bumpers, and needs to be set to pull up for input
100  ******/
101  PORTD_PIN1CTRL = 0x19; //00 011=pullup for input, 001=rising edge
102  PORTD_PIN2CTRL = 0x19;
103
104  uint8_t count = 0; //used for the bumper count. the bumper is pressed 0 times in the beginning for initialization

```

Code 7.7

The above code, Code 7.7, is the set up for the bumper switches. For the bumper switches to work when pressed down, the pins need to be in a pull up form. Also the bumper switches would be most practical if they were triggered as soon as they are pressed, so the pins need to trigger on the rising edge of the signal. The count value is used just to keep track of whether the motor has started yet or not.

Since one of the bumper switches is used for a trigger for the motor to start, Code 7.8 is to start the motor. It senses the bumper being hit and as soon as there is a signal, the ESC will receive the signal and the motor will start spinning. This a more practical way since to start the motor, the timer value must be in a neutral position for some time. So instead of making a delay loop having a switch to trigger is more efficient.

```

122 /to start the motor, hit the left bumper
123 /motor values: full reverse=1.048ms=524, neutral=1.479ms=739, full forward=1.910ms=955
124 if (count<1){ //if the left bumper is pressed down for the first time, count is less than 1
125     if ((PORTD.IN &= 0x02) == 0x02){ //0000 0010 left side bumper
126         printf("!!!!START Motor!!!!\r\n");
127         //!!!!!!!!!!!!!!!!!!!!!!!!shoot to 100% for very little time and then slow it down low, like giving a kick start
128         //do the same for backwards and experiment
129         TCC0_CCA=815;
130         _delay_ms(1);
131         TCC0_CCA=955;
132         _delay_ms(1.5);
133         TCC0_CCA=820; //timer counter 870 controls the speed. this number has to be high enough to get enough
134                       //energy for the motor. if too low, it will be a neutral, or it will spin stop spin stop
135         count++; //add 1 to count so the bumper will not trigger the start of the motor the second time.
136     }
137 }

```

Code 7.8

Finally, the main code for the bumper is shown in Code 7.9. This part of the code is when the robot hits an object or obstacle with its front bumpers, the robot will turn left or right to avoid it.

```

183     else if ((PORTD.IN &= 0x02) == 0x02){ //0000 0010 left side bumper
184         printf("LEFT bump %d\r\n", count); //if it hits the left bumper
185         frontservoangle(80); //turn the servo
186         TCC0_CCA=745; //delay to wait until it moves back
187         _delay_ms(5);
188         TCC0_CCA=510;
189         _delay_ms(1); //then start moving forward again
190         TCC0_CCA=600;
191         _delay_ms(5); //and go backwards
192     }
193     else if ((PORTD.IN &= 0x04) == 0x04){ //0000 0100 right side bumper
194         printf("RIGHT bump %d\r\n", count); //if it hits the right bumper
195         frontservoangle(-80); //set the servo to turn
196         TCC0_CCA=745; //delay to wait until it moves back
197         _delay_ms(5);
198         TCC0_CCA=510;
199         _delay_ms(1); //then start moving forward again
200         TCC0_CCA=600;
201         _delay_ms(5); //and go backwards
202     }

```

Code 7.9

7.3 Webcam

The final sensor is considered a special sensor. The “Shop-Mulinator’s” special sensor is the webcam. The webcam is used to accomplish the user following behavior. It is connected through the laptop and will be coded to search for a specific color and will then follow that color. The color must be very distinctive and bright so it is easy for the camera to detect, but it also cannot be a common color since the robot needs to follow one user and not multiple.

Codes 7.10 and 7.11 are the main part of the codes for the webcam. Since the webcam is connected to the laptop, the software used for the webcam color tracking is Visual Studios and OpenCV. This code is transmitted using XBEE from the laptop to the Epiphany DIY board and analyzed by the code in AVR Studios. The Code 7.10 is from Visual Studios. The code takes in position readings of a unique color of the user and sends the position to the Epiphany DIY board.

When the position readings have been taken in, Code 7.11 will analyze the position and turn the servo in the specified position. In the beginning the robot only analyzed 3 positions, straight, right and left, but it was not precise enough for the robot to follow the user. So instead of 3 positions, it is now 5 positions: hard left, slight left, hard right, slight right, and straight.

```
// Print the position of x and y coordinates
printf("position (%d,%d)\n", posX, posY);

output = posX/4;          //the x value goes from 0 to 1000 as it goes from right to left
                        //so scale it down to get it in the 0 to 255 range
                        //divide by 4 will do 0 to 250 range

if (posX<0){} //do nothing if the position is negative
if (posX>420&posX<520)
{
  output=21;
  printf("slight RIGHT\r\n");
  WriteCom(comport,&output,1);
}
if (posX>520)
{
  output=22;
  printf("hard RIGHT\r\n");
  WriteCom(comport,&output,1);
}
if (posX>80&posX<230)
{
  output=11;
  printf("slight LEFT\r\n");
  WriteCom(comport,&output,1);
}
if (posX>0&posX<80)
{
  output=12;
  printf("hard LEFT\r\n");
  WriteCom(comport,&output,1);
}
if (posX>230&posX<400)
{
  output=0;
  printf("Straight\r\n");
  WriteCom(comport,&output,1);
}

//send the position of item to the comport through xbee
WriteCom(comport,&output,1);
```

Code 7.10

```
else {
  printf("no wall\r\n");
  printf("no bump, count= %d\r\n", count);
  printf("data = %x\r\n",USARTE1.DATA);
  TCC0_CCA=827;
  //nothing is detected
  //display count for debugging purposes, should be 1 right here
  //if nothis is detected continue color tracking
  //right now it is in hex.
  //change it to decimal might be more helpful.

  if (USARTE1.DATA ==11){
    frontservoangle(50);
    printf("go slight left\r\n");
  }
  if (USARTE1.DATA ==12){
    frontservoangle(75);
    printf("go hard left\r\n");
  }
  if (USARTE1.DATA ==21){
    frontservoangle(-50);
    printf("go slight right\r\n");
  }
  if (USARTE1.DATA ==22){
    frontservoangle(-75);
    printf("go hard right\r\n");
  }
  if (USARTE1.DATA ==0){
    frontservoangle(0);
    printf("go straight\r\n");
    //if (front1>600&front1<700||front2>600&front2<700)
    //TCC0_CCA=745;
  }
}
```

Code 7.11

The following images are from the experiment for color tracking. The Shop-Mulinator is programmed to track a shocking pink color. This color was chosen because of the brightness and the ease of spotting, also it is a unique color not many people will be wearing. Also, in the code for the color tracking, the range of hue is narrower to eliminate any noise of other pixels of pinkish colors around the actual target.

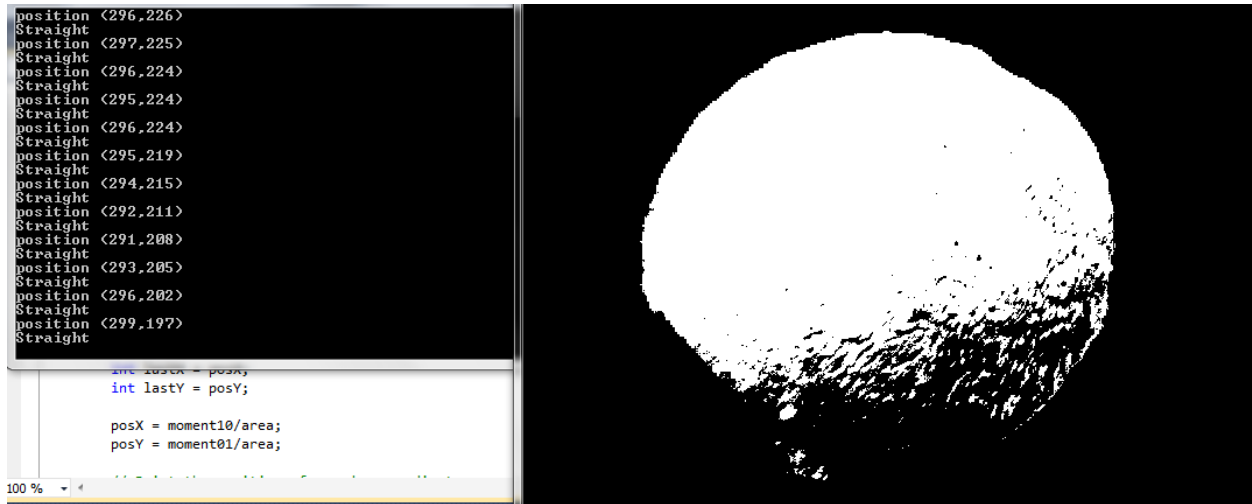


Image 7.17.2



Image 7.17.3

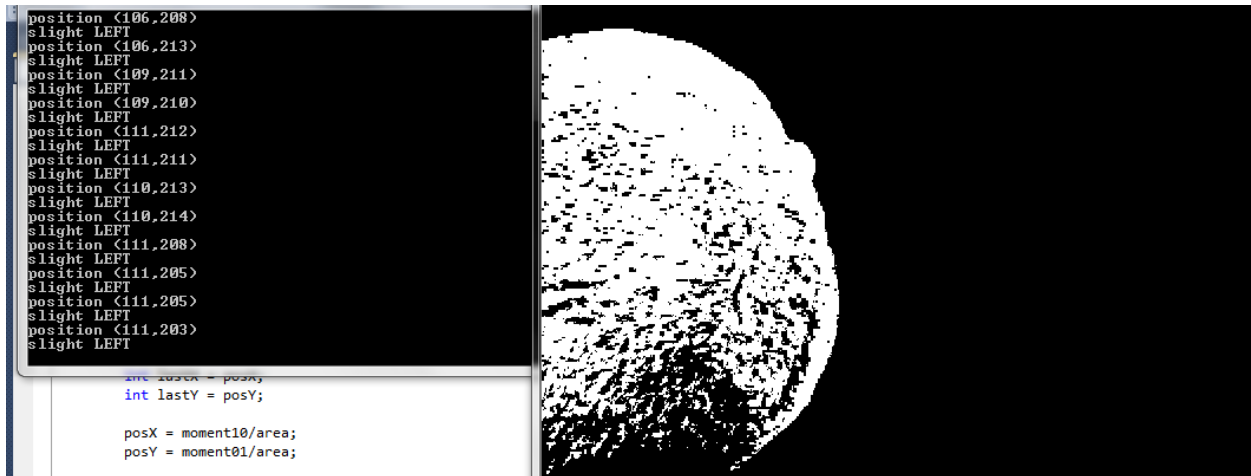


Image 7.17.4

8 Lessons Learned

There are a couple of lessons learned from this process in robot making. The biggest lesson was tweaking the ESC just to get the right speed on the motor. Since the Shop-Mulinator runs on a RC car DC motor, it is very fast without adjusting and without an ESC. Unfortunately, even with an ESC, the motor is still pretty strong for a smooth slow cruising drive.

Also, since this robot is for carrying items, weight is a major issue for the robot speed. The robot had to be fully loaded every time it needed a test run to check motor speed. Having no basket and no shopping items would make the robot move very fast, but putting more shopping items would slow down the motor too much and did not run smoothly.

Another factor to the fast motor is to tweak the IR sensors. Since the robot was moving too fast, the IR sensors had to be triggered way earlier than the desired distance. The reaction time from the IR sensor took longer than the wheels driving forward and the Shop-Mulinator had a difficult time dodging any obstacles or walls.

9 **Conclusion**

The Shop-Mulinator is an automated robot that eases the load of shopping items of the owner. Its sensors and functions include IR sensors to avoid walls and any object that gets in its way, bumper switch is enabled if for any reason the IR sensors fail its obstacle avoidance, and a webcam that color tracks the owners set color in order to track and follow the programmed color. For future projects, the robot may be better if it was not made form steel brackets but a lighter piece of metal. Also it might be better to find a motor other than a RC car motor for smoother operation.

10 Documentation

[1] S. Utkarsh. (2010) *Tracking colored objects in OpenCV*. Available: <http://www.aishack.in/2010/07/tracking-colored-objects-in-opencv/>

[2] Chris from IMDL Fall Class 2011. *Code for XBee Communication*.

[3] T. Martin. (2011) *Epiphany DIY*. Available: <https://sites.google.com/site/epiphanydiy/home>

[4] Atmel XMEGA64A1 Manual:

http://www.atmel.com/dyn/resources/prod_documents/doc8077.pdf

http://www.atmel.com/dyn/resources/prod_documents/doc8075.pdf

[5] Pololu. *Sharp: GP2Y0A02YK0F*. Available:

http://www.pololu.com/file/0J156/gp2y0a02yk_e.pdf

11 Appendices

Appendix A

```

/*****
*****
*---Main Program
*       Controls obstacle avoidance using IR sensors through ADC and bumper switches
through regular I/O port, PortD
*       and color tracking for the robot
*Code by: Tim Martin
*Editted by: Marina Yokota

*****
*****/

/* 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 <util/delay.h>
#include <math.h>
#include "motor.h"
#include "lcd.h"
#include "uart.h"
#include "RTC.h"
#include "picServo.h"
#include "ADC.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

/*****
*****
*This is a separate function which bound the upper and lower angle limit, convert the
angle to timer count,
*and set the servo angle in the timer counter PortD Pin 0
*****/
void frontservoangle(int angle)
{
    if (angle>90){ //limit upper angle bound to 90
        angle = 90;
    }
    else if (angle<-90){ //limit lower angle bound to -90
        angle = -90;
    }

    angle *= 5; //converting angle degrees to timer count number
    angle /= 2;
    TCDD0_CCA = 750 + angle; //set the servo to the specified angle

```

```

}

/*****
*This is where the main code is for the robot
*****/
int main (void)
{
    //This is where all the initialization for the board goes
    board_init(); /*This function originates in the file init.c, and is used to
initialize the Epiphany DIY
the user should define what your
motorInit() is declared within because by default you
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.
    servoControlInit();//initializes the servo module ***including enabling global
interrupts*** required for the servo control module
    uartInit(&USARTC0,57600);//as can be seen in the schematic. This uart is
connected to the USB port. This function initializes this uart
    uartInit(&USARTE1,9600);//initialize the Xbee port with the same baud rate as xbee.
    ADCsInits(); //Initialization for Analog to Digital, this is used for the IR
sensors
    LCDInit(); //LCD initialization
    motorInit(); //motor initialization, if motor drivers were added from Tim
    stdout = &USB_str;//lcd_str This function points stdout to the LCD. So this means
printf will send data out the LCD.
//uartE1_str

/*****
*The below initializes the servo and the motor. The ESC for the motor works the same way
as the servo
*This is setting up for Pulse Width Modulation.
*32MHz=system clock
*timer toggles 0-2^16 = 65536 (overflow 20ms=50Hz); this is one window, for this board
*how many CPU cycles in 32M/50=clocks per window = 640000
*640000 is way bigger than 65536 so scale it down by dividing it by 64 (trial an error)
*640000/64 is good, but the timers do not have prescaler of 16, they have 64 so use 64
*timer initialization for motor=Port C0, Port D0 for front servo
*****/
    PORTC_DIR = 0x0F; //set portc0 to output breakdown-->0000 11-for usb output, 11-for
pin0 and 1
    PORTD_DIR = 0x01; //set portd0 to output breakdown-->0000 0001 for pin0

    TCC0_PER = 10000; //the top value if you divide 640000/64
    TCD0_PER = 10000;

    TCC0_CTRLA = 0x05; //prescaler is 64
    TCD0_CTRLA = 0x05;

    TCD0_CCA = 750; //the window for PWM window is 1ms to 2ms
//compare to 0 to 10000 count --> half way 1.5ms = 7500
//move a decimal place over so its 750.
//set servo to an angle to 0 (1.5ms mark) for
initialization
    TCC0_CCA = 745; //set the number corresponding to neutral for motor. same as 1.5ms
//look up on the esc programmer

```

```

//this initializes the motor at stop to pass through
arming time for esc

TCC0_CTRLB = 0x13; // select capture compare for PortC & D and single slope wave
TCD0_CTRLB = 0x13;

/*****
*The bumpers are connected to the PORTD pins. 2 at the front of the robot.
*NO (normally open=green) = 5V, NC (normally closed=brown) = gnd, C (common=red) = signal
*PORTD Pin 1 and 2 are for bumpers, and needs to be set to pull up for input
*****/
PORTD_PIN1CTRL = 0x19; //00 011=pullup for input, 001=rising edge
PORTD_PIN2CTRL = 0x19;

uint8_t count = 0; //used for the bumper count. the bumper is pressed 0 times in
the beginning for initialization

/*****
*This is where the program loops infinitely, all main actions/program is done here
*****/
while (1){
    asm("nop");

    ADCA_request(0, ExternalChannel0); //front IR
    ADCA_request(1, ExternalChannel1); //front IR
    ADCA_request(2, ExternalChannel2); //back IR

    if (ADCA_CH0_ConvComplete==1, ADCA_CH1_ConvComplete==1, ADCA_CH2_ConvComplete==1){
        //if the flag for ADC's are true

        uint16_t front1 = ADCA_getVal(0); //get value of the front IR sensor:
right
        uint16_t front2 = ADCA_getVal(1); //get value of the second front IR
sensor: left
        uint16_t back1 = ADCA_getVal(2); //get value of the IR sensor in
the back

        //to start the motor, hit the left bumper
        //motor values: full reverse=1.048ms=524, neutral=1.479ms=739, full
forward=1.910ms=955
        if (count<1){ //if the left bumper is pressed down for the first time,
count is less than 1
            if ((PORTD.IN &= 0x02) == 0x02){ //0000 0010 left side bumper
                printf("!!!!START Motor!!!!\r\n");
                //!!!!!!!!!!!!!!!!!!!!!!!!shoot to 100% for very little time and then
slow it down low, like giving a kick start
                //do the same for backwards and experiment
                TCC0_CCA=815;
                _delay_ms(1);
                TCC0_CCA=955;
                _delay_ms(1.5);
                TCC0_CCA=820; //timer counter 870 controls
the speed. this number has to be high enough to get enough

                //energy for the motor. if too low, it will be a neutral, or it will spin stop
spin stop repeatedly

                count++; //add 1 to
count so the bumper will not trigger the start of the motor the second time.

```



```

    }
}

//if the motor is already running, then do obstacle avoidance and color tracking
//IR high value = wall, turn away from the wall and then keep moving forward
if (count==1){
    if (front1>=700){ //!!!!!!!!!!!!!!!!!!!!!!have the ir sense further away
to be safe for obstacle avoidance
        if (front2>=850){
            printf("WALLLLL, %d, %d\r\n", front1, front2);
//walls detected by both IR, left and right.
            frontservoangle(-85);
//go backwards, away from wall.
            TCC0_CCA=745;
//delay to wait until it moves back
            _delay_ms(1);
            TCC0_CCA=524;
            _delay_ms(1.5);
//then start moving forward again
            TCC0_CCA=610;
        }
        else {
            printf("LEFT ir %d, %d\r\n", front1, front2);
//wall detected by just the left IR,
            frontservoangle(-85);
//so just turn away from the left.
        }
    }
    else if (front2>=700){
        if (front1>=850){
            printf("WALLLLL, %d, %d\r\n", front1, front2);
//walls detected by both IR, left and right.
            TCC0_CCA=745;
            frontservoangle(85);
//so go backwards.
            _delay_ms(1);
            TCC0_CCA=524;
            _delay_ms(1.5);
//wait to move back enough space
            TCC0_CCA=610;
//then start moving forward again
        }
        else {
            printf("RIGHT ir %d, %d\r\n", front1, front2);
//wall detected only on the right side
            frontservoangle(85);
//just turn away from the right.
        }
    }

    else if (back1>=1800){
        printf("BACK ir %d\r\n",back1);
//back ir detected the wall
        TCC0_CCA=745;
//stop the motor
        frontservoangle(0);
//make wheels straight
        _delay_ms(5);
    }
}

```

```

        TCC0_CCA=815;
        _delay_ms(1);
        TCC0_CCA=955;
        _delay_ms(1.5);
        //wait a little
        TCC0_CCA=820;
//then go in the forward direction.
    }
    else if ((PORTD.IN &= 0x02) == 0x02){           //0000 0010 left
side bumper
        printf("LEFT bump %d\r\n", count);         //if it hits the
left bumper
        frontservoangle(80);                       //turn the
servo
        TCC0_CCA=745;
//delay to wait until it moves back
        _delay_ms(5);
        TCC0_CCA=510;
        _delay_ms(1);
//then start moving forward again
        TCC0_CCA=600;
        _delay_ms(5);                               //and
go backwards
    }
    else if ((PORTD.IN &= 0x04) == 0x04){           //0000 0100 right
side bumper
        printf("RIGHT bump %d\r\n", count);         //if it hits
the right bumper
        frontservoangle(-80);                       //set
the servo to turn
        TCC0_CCA=745;
//delay to wait until it moves back
        _delay_ms(5);
        TCC0_CCA=510;
        _delay_ms(1);
//then start moving forward again
        TCC0_CCA=600;
        _delay_ms(5);                               //and go
backwards
    }
    else {
        printf("no wall\r\n");
//nothing is detected
        printf("no bump, count= %d\r\n", count);
//display count for debugging purposes, should be 1 right here
        printf("data = %x\r\n",USARTE1.DATA);
//if nothis is detected continue color tracking
        TCC0_CCA=827;
//right now it is in hex.

        //change it to decimal might be more helpful.
        if (USARTE1.DATA ==11){
            frontservoangle(50);
            printf("go slight left\r\n");
        }
        if (USARTE1.DATA ==12){
            frontservoangle(75);
            printf("go hard left\r\n");
        }
    }

```

```

    }
    if (USARTE1.DATA ==21){
        frontservoangle(-50);
        printf("go slight right\r\n");
    }
    if (USARTE1.DATA ==22){
        frontservoangle(-75);
        printf("go hard right\r\n");
    }
    if (USARTE1.DATA ==0){
        frontservoangle(0);
        printf("go straight\r\n");
        //if (front1>600&front1<700||front2>600&front2<700)
        //TCC0_CCA=745;
    }
}
}
    }
    _delay_ms(50);
} //close the if statement for the ADC flag
} //close the while loop
} //close the main loop

```

Appendix B

```

/*
This code is made by Utkarsh
The link: http://www.aishack.in/2010/07/tracking-colored-objects-in-opencv/
With small edits by Marina Yokota
This code is just for color tracking from the camera
and send that data to the board.
*/

#include <cv.h>
#include <highgui.h>
#include <COMfunct.h> //Code for comport communication by Chris.

IplImage* GetThresholdedImage(IplImage* img)
{
    // Convert the image into an HSV image
    IplImage* imgHSV = cvCreateImage(cvGetSize(img), 8, 3);
    cvCvtColor(img, imgHSV, CV_BGR2HSV);
    IplImage* imgThreshed = cvCreateImage(cvGetSize(img), 8, 1);
    cvInRangeS(imgHSV, cvScalar(150, 100, 100), cvScalar(168, 255, 255), imgThreshed);
    cvReleaseImage(&imgHSV);
    return imgThreshed;
}

int main()
{
    char output; //the name for output data, which in this case
    is the position in x coordinate
    HANDLE comport; //to call to the comport
}

```

```

    comport = OpenCom(7);           //open the comport for the xbee (which is
comport 7) and name it comport

    // Initialize capturing live feed from the camera
    CvCapture* capture = 0;
    capture = cvCaptureFromCAM(0);   //this is for the webcam
    //capture =
cvCreateFileCapture("http://192.168.137.216:8090/videostream.cgi?user=admin&pwd=&a=.mjpg"
); // For IP cam

    // Couldn't get a device? Throw an error and quit
    if(!capture)
    {
        printf("Could not initialize capturing...\n");
        return -1;
    }

    // The two windows we'll be using
    cvNamedWindow("video");
    cvNamedWindow("thresh");

    // This image holds the "scribble" data...
    // the tracked positions of the ball
    IplImage* imgScribble = NULL;

    // An infinite loop
    while(true)
    {
        // Will hold a frame captured from the camera
        IplImage* frame = 0;
        frame = cvQueryFrame(capture);

        // If we couldn't grab a frame... quit
        if(!frame)
            break;

        // 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* imgYellowThresh = GetThresholdedImage(frame);

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

    // Print the position of x and y coordinates
printf("position (%d,%d)\n", posX, posY);

    output = posX/4;           //the x value goes from 0 to 1000 as it
goes from right to left                                           //so scale it down to get
                                                                    //divide by 4 will do 0 to
it in the 0 to 255 range
                                                                    250 range

if (posX<0){} //do nothing if the position is negative
if (posX>420&posX<520)
{
    output=21;
    printf("slight RIGHT\r\n");
    WriteCom(comport,&output,1);
}
if (posX>520)
{
    output=22;
    printf("hard RIGHT\r\n");
    WriteCom(comport,&output,1);
}
if (posX>80&posX<230)
{
    output=11;
    printf("slight LEFT\r\n");
    WriteCom(comport,&output,1);
}
if (posX>0&posX<80)
{
    output=12;
    printf("hard LEFT\r\n");
    WriteCom(comport,&output,1);
}
if (posX>230&posX<400)
{
    output=0;
    printf("Straight\r\n");
    WriteCom(comport,&output,1);
}

    //send the position of item to the comport through xbee
WriteCom(comport,&output,1);

    // 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("thresh", imgYellowThresh);
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(&imgYellowThresh);
delete moments;
}

        // We're done using the camera. Other applications can now use it
cvReleaseCapture(&capture);
return 0;
}
```