Date: *4/19/11*
Student Name: Matt Morgan
TAs: Devin Hughes
Tim Martin
Ryan Stevens
Josh Weaver

Instructors: Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz

University of Florida
Department of Electrical and Computer Engineering
EEL 4665/5666
Intelligent Machines Design Laboratory

Final Report

Robot :
The Iron HouseMaiden

Table of Contents

Abstract

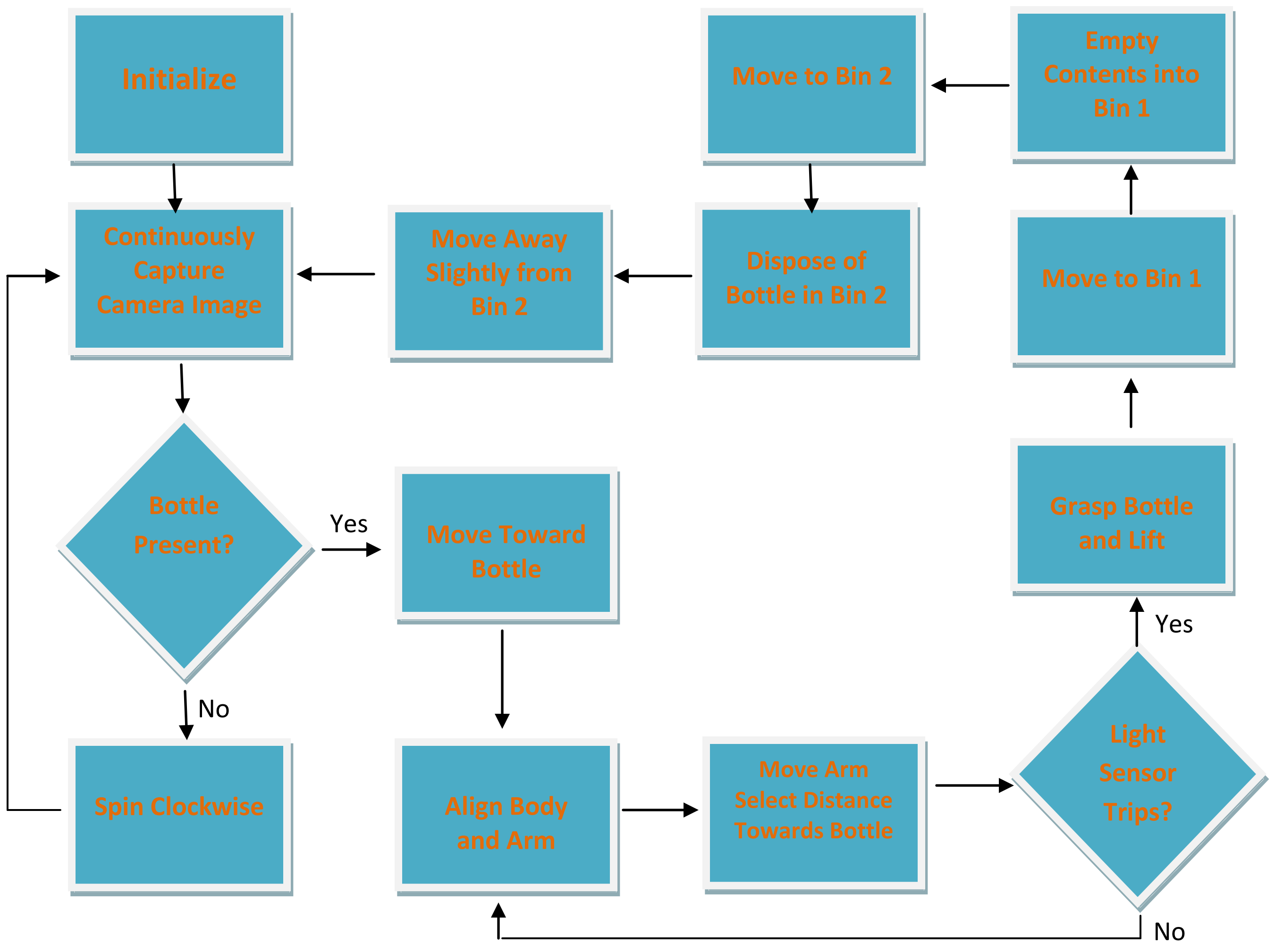
This report details the design and implementation of an autonomous robot that collects green colored bottles, empties their contents into a red, specified bin, and proceeds to dispose of the bottles into a separate, blue bin. A powerful microprocessor makes up the core of this robot's functionality and controls it's motion control and maneuverability. It further utilizes an IP webcam, along with a Bluetooth connection to an external laptop, to capture images for vision and image processing where an even more powerful processor is needed.

Introduction

Have you ever decided to have a few friends over to enjoy a game on TV, yet woke up ten minutes before work to find out that your living room floor is now completely enshrouded in half empty bottles? The purpose of the project is to create an autonomous robot, designed primarily for indoor use, that is, first and foremost, capable of seeking out and collecting green colored bottles. It then uses its claw-like mechanism to lift the bottles, one at a time, to a specified disposal site. The robot empties any remaining liquids contained in the bottles into a specified, red bin, and then carries them to another specified, yet separate, blue bin where the bottles will ultimately be disposed of. The scope of the project primarily focuses on precise motion control and functionality, as well as the use of an IP webcam for advanced image processing.

Integrated System

Process Flow


Initialize
Move to Bin 2
Empty Contents into Bin 1
Continuously Capture Camera Image
Move Away Slightly from Bin 2
Dispose of Bottle in Bin 2
Move to Bin 1
Bottle Present?
Yes
Move Toward Bottle
Grasp Bottle and Lift
Yes
No
Spin Clockwise
Align Body and Arm
Move Arm Select Distance Towards Bottle
Light Sensor Trips?
No

Mobile Platform

The platform consists of a stack of two wooden, circular disks. Two motors along with a dual motor controller are mounted along with the IP webcam between the disks towards the front of the robot. A metal ball caster is placed underneath the bottom disk towards the back of the robot along its midpoint. Two bump sensors (one in the front, and one in the back) are located along the edge of the lower platform. Finally, three midrange IR sensors are mounted on top of the upper deck along with the microcontroller [1], an LCD [2], and the shoulder joint of the arm on the top of the upper disk. Two additional short range IR sensors are mounted on the gripper as well.

Actuation

The actuation on this robot consists primarily of its method of movement along with the motion and maneuverability of the arm. Two Pololu gearmotors [3] with 90x10 mm wheels [4] attached to the shafts are controlled by a Pololu Qik 2s12v10 Dual Serial Motor Controller [5]. The arm is made out of wood and has a slightly arced shape to allow for grappling of bottles standing upright. It uses one Hitec HS-985MG and two Hitec HS-645MG servos [6], modified for 180° rotation, to achieve two degrees of freedom, one in the shoulder joint and one in the wrist, as well as the ability to clamp the "hand."

Sensors

IP Webcam

The camera used in the project is a Linksys Wireless-N+RJ45 [7]. A BlueSMiRF Gold [8] is integrated into the PVR board to allow for wireless communication with an external laptop via Bluetooth technology. Images are repeatedly captured via http protocol and then analyzed and modified via OpenCV libraries that contain numerous image processing, matrix operation, and transformation algorithms. With the help of these functions, the Iron HouseMaiden is able to recognize, and locate, green objects with very specific hue and saturation values. Several outputs to pertinent algorithms can be seen in Appendix A.

IR Sensors

Three Sharp GP2Y0A02YK0F Analog Distance Sensor [9] with a specified range between 20 and 150cm along with Two Sharp GP2Y0A21YK0F Analog Distance Sensor [10] with a range between 10 and 80 cm is used for object avoidance. The three long distance sensors are located on the upper portion of the platform assembly, while the two shorter range sensors are located on the gripper / arm assembly itself. The ADC located on the Atmel Xmega is what is used to convert the analog signal into digital information. The voltage vs. distance plots for each sensor are located in Appendix A.

Bump Sensors
Bump sensors are utilized on the front and rear of the robot. The rear bump sensor is also used to initiate the robot's actions. Other than the "start bump," they are all interrupt driven by software. The basic design of the simple circuit is shown below in Appendix A.

Light Intensity Sensor
A bright white LED [11] is paired with Jameco MAX LITE photocell [12] to create a light intensity sensor on the gripper. The photocell is in series with a 10 kΩ resistor so that when a bottle is far enough inside the clamp, the light is blocked enough to bring the ADC integer value from around 1000 to approximately 2500 (max value = 4095). Thus, the robot knows to close its grip.

Behaviors

The basic behaviors of the Iron HouseMaiden require a somewhat significant amount of sophistication. It first rotates in search of its target (a bottle), which it finds with the use of several different compares and a combination of both transforms and processing algorithms (important code snippets can be found in Appendix B; however, the complete source code can be found on my website: plaza.ufl.edu/mmblind10). When it discovers what it believes to be a bottle, it moves towards it. Re-calibration takes place as a second check and a buffer for itself. As it moves, obstacle avoidance is primarily governed by the five IR sensors; however, there does exist a small amount which is governed by webcam processes.

When the Iron HouseMaiden finally decides that it has found what it is looking for by using a blob area algorithm where it compares the overall area of the green blob to the size of the frame, it initiates an algorithm to position itself at the correct place in front of its target. With the help of the light intensity sensor, it knows when to close its grip and proceed to lift the bottle off of the ground.

The initial behavior then re-occurs, this time looking for the red bin. Once it believes it has found it, it moves towards the bin, and stops when the area of the bin has surpassed a specified value. It initiates its servos and lifts the bottle high enough to dispose of its contents. It slightly turns left until it has positioned itself correctly relative to the red blob, and then proceeds to move forward a given distance. When it has moved forward far enough, it stops, initiates its wrist servo and empties the remaining contents of the bottle into the red bin. It then backs up, quickly locates the blue bin close by, and moves toward it as it did with the red bin. Again it properly aligns itself relative to the blob and proceeds forward until it has decided it is close enough to the bin. It initiates its servos and proceeds to dispose of the bottle in the bin. It then backs away from the bins where it proceeds to repeat the process all over.

Experimental Layout and Results

The most experimental portion of the project was the image processing done with the IP webcam and OpenCV. My initial algorithm involved using contour matching in RGB (red, green, blue) color space. The contour matching was extremely successful (seen in Appendix A), and although the recognition of green appeared to be successful as well, there was a ton of noise inherent in the design and the robot would almost always locate false positives. The transition to HSV color space (hue, saturation, and value) eliminated a lot of the noise and with the addition of a Gaussian smoothing filter, a binary erode filter, and a binary dilate filter, I was able to almost completely eliminate both outside noise and false positives in the image. However, due to the implementation of the contour matching combined with the fact that the camera was sending image wirelessly, the processing was too slow for the robot to react quick enough.

Next, I redeveloped my code and integrated a blob tracking technique into my own code that ran at a much faster rate. Although, it was less precise, it allowed for my robot to react in real time and made communication between my laptop and my robot (via Bluetooth) much easier to code. The blob tracking is still done in HSV color space and the additional filters are still needed. I further implemented a start-up calibration algorithm for the robot to calibrate the colors it is looking for in memory. Screenshots of the output of the blob tracking algorithm can be seen in Appendix A, and portions of code where the algorithm was used can be seen in Appendix B.

After the tracking algorithm was secure, the next challenge was creating a system for my robot to react to what my laptop is processing. With the help of Bluetooth, I was able to send different characters to my robot depending on the response that it needed to perform. Essentially, there were designated characters (Hex values) for the robot to perform such behaviors as "turning slight right," "brake," "turn hard left," "lift bottle," etc. A good portion of this code can be found in Appendix B.

The last and final challenge involved the fact that the IP webcam would continuously capture images at a rate of approximately 40 Hz. This became a problem when the robot was asked to initiate a chain of commands that took a relatively large amount of time (like dumping the bottle). When, the robot was ready to proceed to the next objective, the computer was still querying frames from right after the robot started its previous objective. This caused an enormous amount of lag and made the robot travel in false directions (for the images were not real time). This was overcome by the combination of a delay counter and a function that would query frames and dispose of them immediately.

Conclusion

I believe the project was a great success and demonstrated a substantial amount of image processing capability. The robot was successfully able to locate the green bottles, grab hold of them, empty their contents into the specified red bin, and ultimately dispose of the bottle into the blue bin.

There were a great deal of situations for which I was unprepared and could have done much better. First of all, it would have been much smarter to cover the bottles with paper or some kind of dull, or unfinished material before painting them. The glare from the lights would reflect off the bottles and cause the source code to have to be slightly altered whenever different lighting conditions would arise. This was only apparent with the bottle, however, and was not witnessed with the dull finish of the painted cardboard bins. A separate approach would have been to provide my own lighting that matched my test area. That would have eliminated the additional variables that arose from the various environments in which my robot was demonstrated.

Next, I spent far too long perfecting the image processing algorithms before the robot was completely built. I wanted to guarantee that they worked correctly before setting my design in stone; however, I was unprepared for how much the code would have to change after the systems were integrated. This further led to additional hardware problems. I completely underestimated how much load was going to be placed on the shoulder joint and was forced to both upgrade my servo, as well as provide it with the maximum voltage (6 V) for additional torque and power.

Finally, I spent far too much time trying to debug and fix a broken webcam. Originally, I was trying to make my camera work on an ad-hoc network with my PC; however, after countless attempts and a multitude of resets I failed. When I finally settled on using it wirelessly over a network, my camera was having power issues and would turn itself off after transmitting around 200 frames (at 40 frames/second). It took me three, long days and several different circuits to realize that it was not a power problem and that the wireless adapter in the camera had gone bad. Luckily, a classmate had an extra camera for sale, and within a couple days a new camera was hooked up and working properly.

I believe that, as a whole, this project provided countless benefits in both aided and brute force learning. I learned a multitude of lessons in various engineering aspects including: electrical, mechanical, and computer science. I believe my application can be worked on, and perfected to allow for a quick and clean, un-manned clean-up of virtually any kind of environment that is littered with, essentially, any color bottle made from any type of material. This project has shed some light on the power that advanced image processing and wireless communication have when working together. I ultimately believe that, when the two concepts are combined, they can stretch the limit on the capabilities of autonomous robotics, and make the visions of so many into actual realities.

Documentation

Parts:

[1] PVR Microcontroller : Thomas Vermeer and Mike Pridgen
[2] Basic 16x2 Character LCD : Sparkfun sku: LCD-9051
[3] 131:1 Metal Gearmotor 37Dx57L mm : Pololu Part #1107
[4] Pololu Wheel 90x10mm Pair - Black : Pololu Part #1435
[5] Pololu Qik 2s12v10 Dual Serial Motor Controller : Pololu Part #1112
[6] Hitec HS-985MG Servos : Servocity Part #32985S
Hitec HS-645MG Servos : Servocity Part #32645S
[7] Linksys Wireless-N+RJ45 : Newegg Item #N82E16881334006
[8] Bluetooth Modem - BlueSMiRF Gold : Sparkfun sku: WRL-00582
[9]Sharp GP2Y0A02YK0F Analog Distance Sensor : Pololu Part #1137
[10] Sharp GP2Y0A21YK0F Analog Distance Sensor : Pololu Part #136
[11] Super Bright White LED : RadioShack Part #276-017
[12] Jameco MAX LITE Photocell : Jameco : Manufacturer # CDS001-8001

Atmel Documentation:

[1] AVR Xmega A1 Microcontroller datasheet:
http://www.atmel.com/dyn/resources/prod_documents/doc8067.pdf
[2] AVR Xmega A Manual Microcontroller datasheet:
http://www.atmel.com/dyn/resources/prod_documents/doc8077.pdf

BlueSMiRF Gold Documentation:

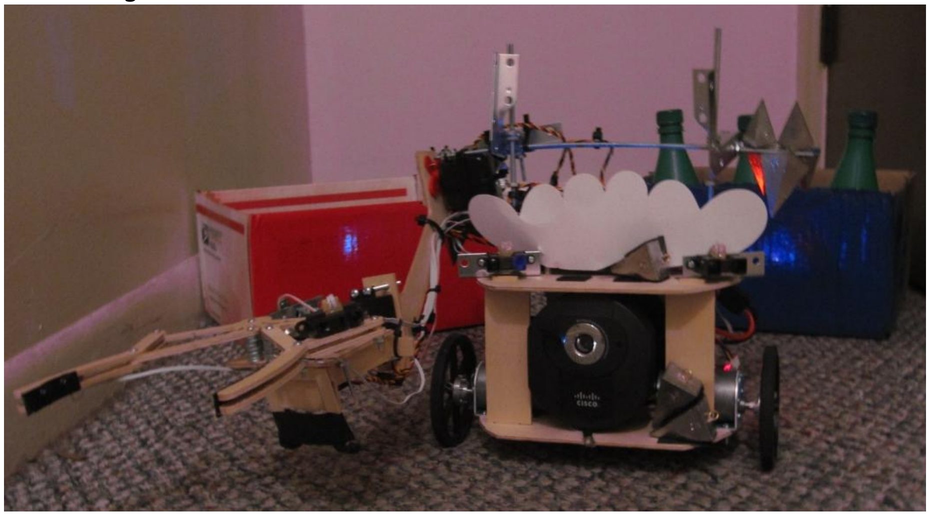
[1] Bluetooth Advanced Users Manual
http://www.sparkfun.com/datasheets/Wireless/Bluetooth/rn-bluetooth-um.pdf
[2] Bluetooth module datasheet:
http://www.rovingnetworks.com/documents/RN-41.pdf

OpenCV Documentation:

[1] Download / Install:
http://opencv.willowgarage.com/wiki/
[2] Blob Tracking:
http://code.google.com/p/cvblob/

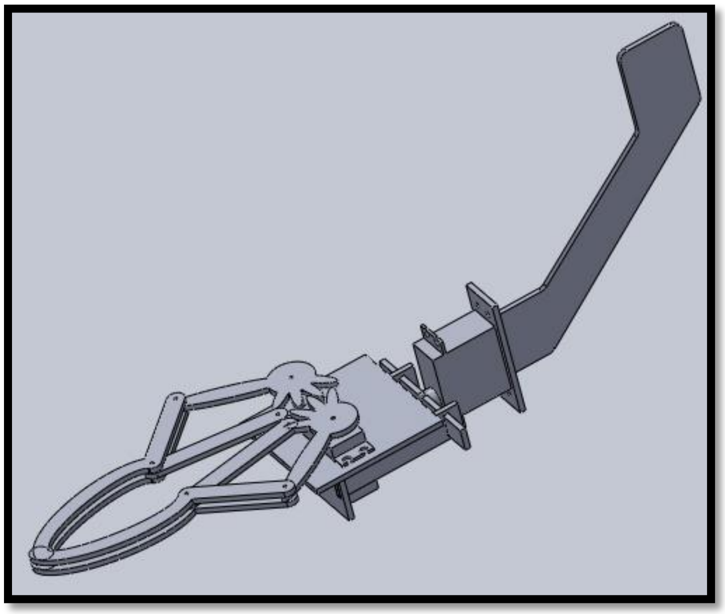
Appendix A

Final Design



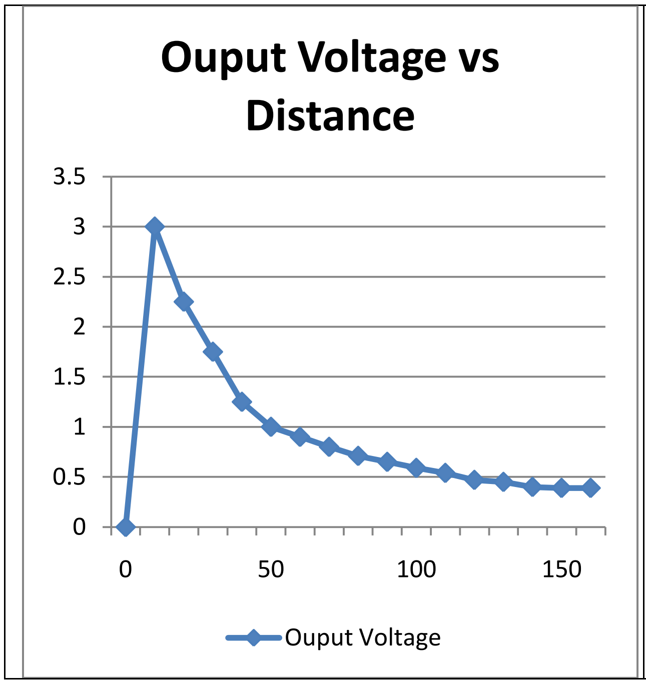
Note: The white object on top of the robot acts as both a maid reference and as a shield to block LED light coming from the PVR board and the LCD screen from reflecting off of the glossy bottles

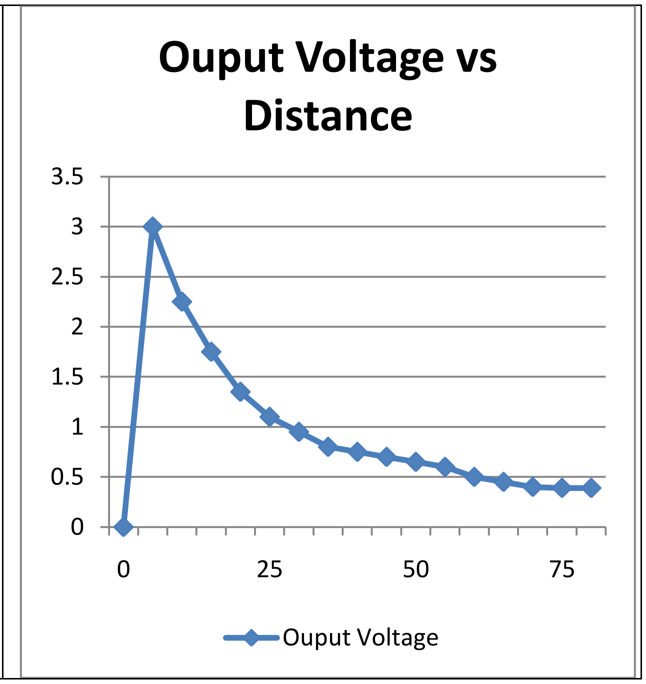
Gripper CAD Assembly



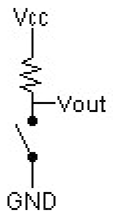
Note: Developed with SolidWorks 2010

IR Senor Plots



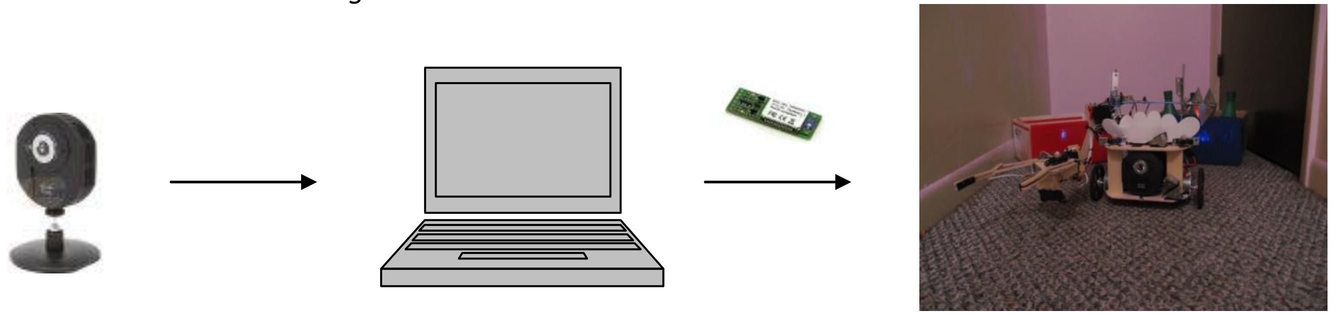


Bump Sensor Circuit

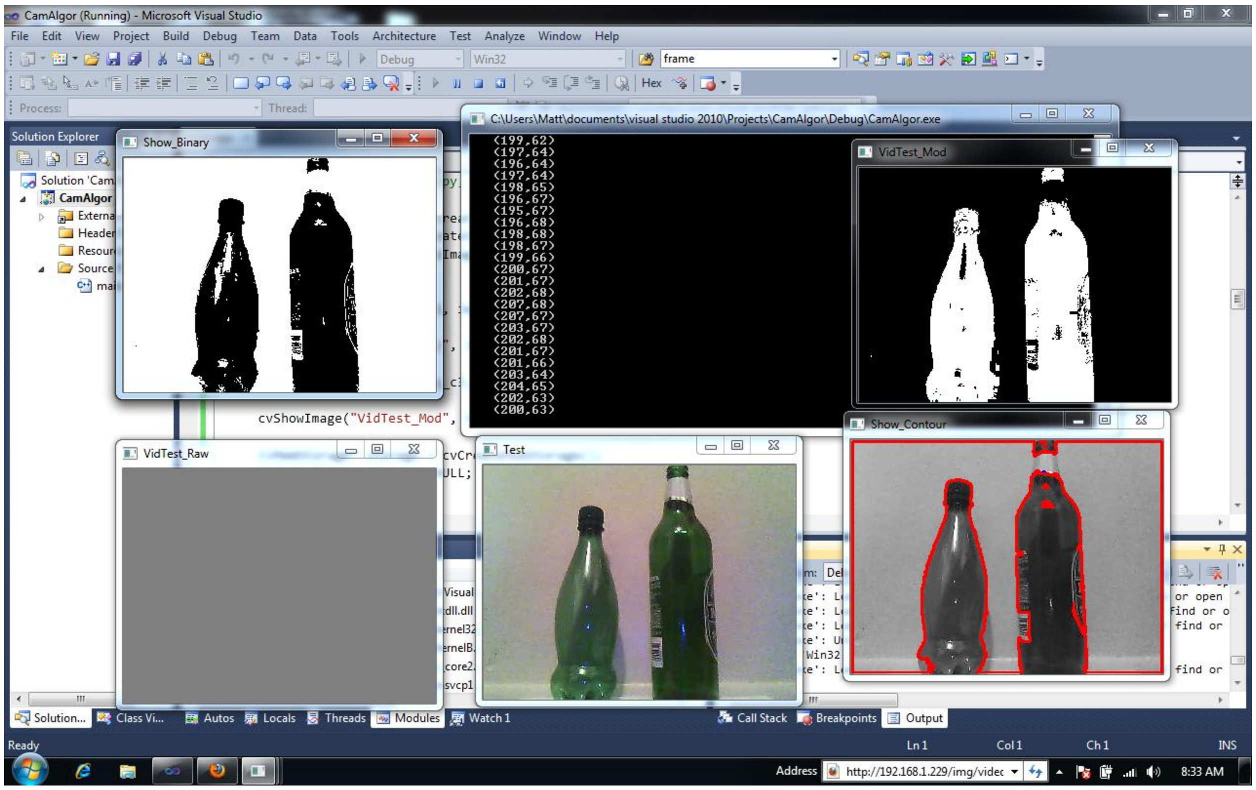


*V_{out} is pulled to a V_{cc} when the switch is closed (when the sensor is triggered).

Special Sensor Functional Diagram



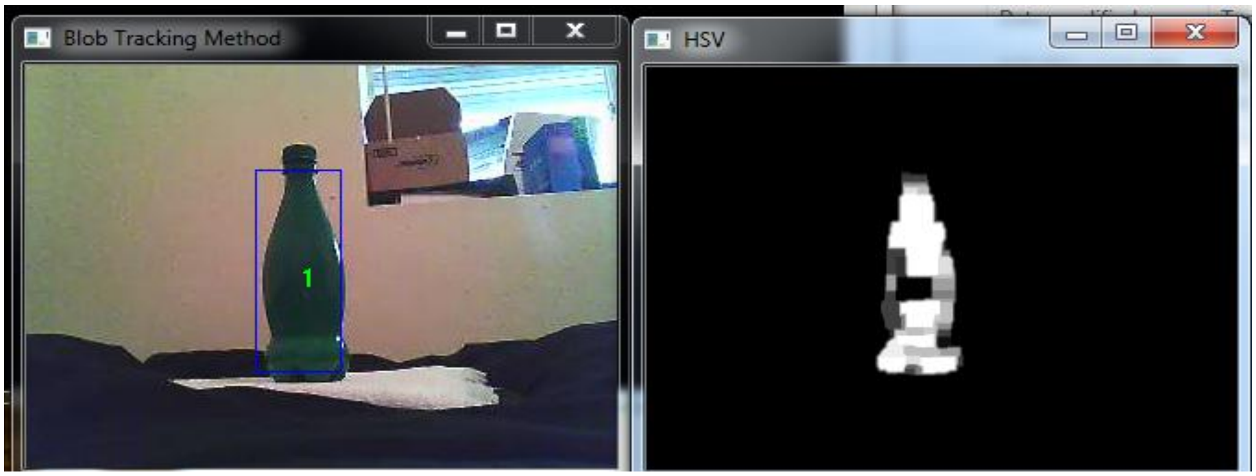
Screenshot of Various IP Webcam Modified Outputs


CamAlgor (Running) - Microsoft Visual Studio
Show_Binary
VidTest_Mod
VidTest_Raw
Test
Show_Contour
C:\Users\Matt\documents\visual studio 2010\Projects\CamAlgor\Debug\CamAlgor.exe

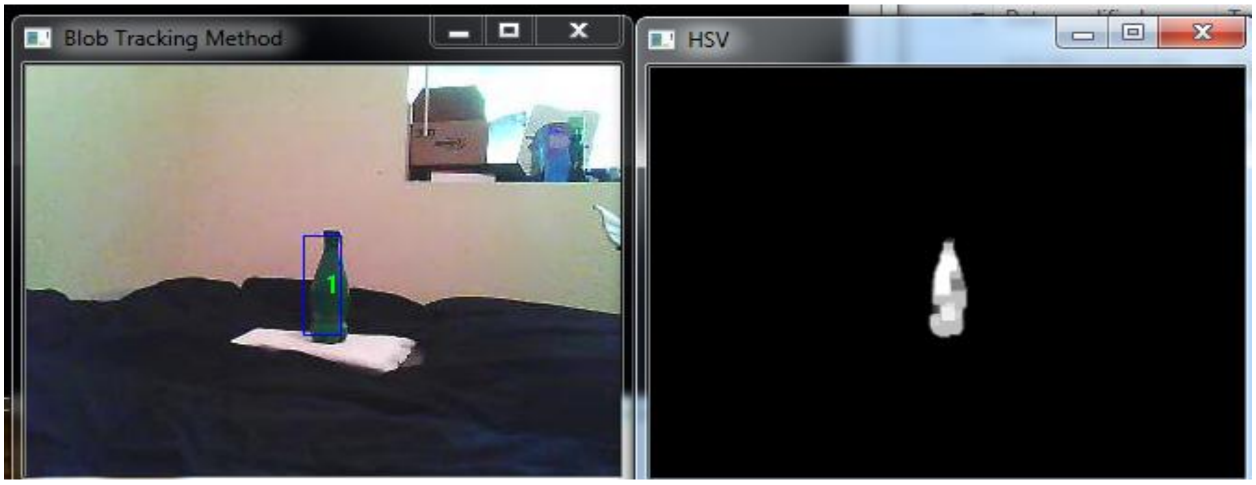
```
(199,62)
(197,64)
(196,64)
(197,64)
(198,65)
(196,67)
(195,67)
(196,68)
(198,68)
(198,67)
(199,66)
(200,67)
(201,67)
(202,68)
(207,68)
(207,67)
(203,67)
(202,68)
(201,67)
(201,66)
(203,64)
(204,65)
(202,63)
(200,63)
```

```
cvShowImage("VidTest_Mod",
```

Clockwise starting from the top-left -> Binary Threshold Output, Pixel coordinates of the Contour Drawing, ColorFinder (green), Contour Drawer / Locator, Test Feed



Blob Track Method -> Real-time and HSV color space images utilizing tracking method with dilate and erode filters



Blob Tracking Method -> 3 channel real-time and HSV color space images shown utilizing dilate, erode, and smoothing filtering algorithms

Appendix B: Code

*The bulk of the code used in this project can be found in the zip file located here:
http://plaza.ufl.edu/mmblind10/IronHouseMaiden.zip

**The following are several important sections of code performing the main functions

Main Code Implemented on Robot:

```
/***************************\
 Glass Bottle Glutton Code v3
 By Matt Morgan
 IMDL Spring 2011
\***************************/

#include <stdlib.h>
#include <stdio.h>
#include "servo.h"
#include "interrupts.h"

/*********
 * Xmega *
 *********/

int FLir, BLir, FRir, GRir, GMir;
int rand1;
int count = 0;
int nextCount = 0;
int speed = 30;
int IRflag = 1;
int array[3];

int main()
{
	extern int ServoSH;
	extern int ServoWR;
	extern int ServoGR;

	xmegaInit();						//setup XMega

	delayInit();						//setup delay functions
	ADCAInit();							//setup PORTA analog readings
	ServoCInit();						//setup Servos on Port C
	ServoC0(ServoSH);						//shoulder servo ->
	ServoC1(ServoWR);						//wrist servo ->
	ServoC2(ServoGR);						//gripper servo ->
	usart_init();						//setup usart
	bluetooth_init();					//setup bluetooth
	lcdInit();
	PortHSet();							//enable PortH pin 0 as output

	while ((PORTF.IN & 0x01) == 1);	//compensate for motor driver being turned on
	_delay_ms(100);						//setup LCD on PORTK
	PortFinterrupt0_init();						//setup portF interrupt 0 on pin 1

	while ((PORTF.IN & 0x01) == 1);

	lcdString("Calibrate");
	calibrateCamera();

	LightOn();
	lcdGoto(0,0);
	lcdString("    GBGv3");

	while(1)
	{
		bluetooth_receive_act(array, speed, count, IRflag);

		count = array[0];
		speed = array[1];
		IRflag = array[2];

		if (IRflag)
		{
			IR_react(speed, FLir, FRir, BLir, GRir, GMir);
		}
	}
}

SIGNAL(PORTF_INT0_vect)
{
	lcdGoto(0,0);
	lcdString("Go!        ");

	ebrake(127);

	_delay_ms(200);

	back_up(15, 500);
	back_up(30, 2000);
	back_up(15, 500);
	back_up(5, 500);
	coast(200);

	ebrake(127);

	_delay_ms(100);

	rand1 = random();
	if (rand1 < 0) rand1 = -rand1;

	if ((rand1 & 0x1) == 1)
	{
		turn_right(15, (rand1 % 2000));
	}
```

```
        else
        {
                turn_left(15, (rand1 % 2000));
        }

        StraightForward(speed/2);
        _delay_ms(100);
        StraightForward(speed);
}
```

Bluetooth Reaction Function Implemented on Robot:

```
void bluetooth_receive_act(int array[], int speed, int count, int IRflag)
{
        unsigned char rec;

        rec = BluetoothReadChar();

        if (((rec >> 4) & 0xF) == 0x3)
        {
                if (rec == 0x30) //go left
                {
                        turn_soft_left(speed, 50);
                }
                else if (rec == 0x31) //go right
                {
                        turn_soft_right(speed, 50);
                }
                else if (rec == 0x32) //go straight
                {
                        StraightForward(speed);
                        _delay_ms(50);
                }

                IRflag = 1;
        }
        else if (((rec >> 4) & 0xF) == 0x4)
        {
                if (rec == 0x40) //brake
                {
                        ebrake(50);
                        _delay_ms(50);
                        ebrake(127);

                        IRflag = 0;

                        bluetooth_transmit(0x13);

                }
        }
        else if (((rec >> 4) & 0xF) == 0x5)
```

```
{
        if (rec == 0x50) //align bottle for grip
        {
                turn_hard_left(speed, 0);
                _delay_ms(50);
                coast(0);
        }
        else if (rec == 0x51)
        {
                turn_hard_right(speed, 0);
                _delay_ms(50);
                coast(0);
        }
        else if(rec == 0x52)            //grip bottle
        {
                StraightForward(20);
                while (!checkLight());

                ebrake(127);

                grabBottle();

                _delay_ms(100);

                bluetooth_transmit(0x10);

                turn_hard_right(15, 0);
        }
}
else if (((rec >> 4) & 0xF) == 0x6)
{
        if (rec == 0x60)                //dump bottle
        {
                bottleDump1();

                turn_soft_left(speed, 1050);
                StraightForward(speed);
                _delay_ms(2750);

                ebrake(127);

                bottleDump2();
                _delay_ms(1000);

                back_up(speed, 3500);

                bottleDump3();

                back_up_left(speed, 1300);
                back_up(speed, 3500);

                ebrake(80);
                _delay_ms(100);
```

```
				ebrake(127);

				bluetooth_transmit(0x11);

				turn_hard_right(15, 0);
			}
			else if (rec == 0x61)				//trash bottle
			{

				ebrake(127);

				bottleDump1();

				turn_soft_left(speed, 200);
				 StraightForward(speed);
				 _delay_ms(4000);

				ebrake(127);

				ebrake(127);

				release_grip();
				_delay_ms(500);

				back_up(speed, 3500);

				bottleDump3();

				back_up(speed, 2500);

				ebrake(80);
				_delay_ms(100);
				ebrake(127);

				bluetooth_transmit(0x12);

				IRflag = 1;

				turn_hard_right(15, 0);
			}
		}
		else if (((rec >> 4) & 0xF) == 0x7)	//for calibration
		{
			if (rec == 0x70)
			{
				turn_hard_left(speed, 0);
				_delay_ms(100);
				coast(0);
			}
			else if (rec == 0x71)					//for calibration
			{
				turn_hard_right(speed, 0);
				_delay_ms(100);
```

```
                    coast(0);
                }
            }
            else if (rec == 0xFE)
            {
                coast(0);
            }
            else if (rec == 0xFF)
            {
                turn_hard_right(15, 0);
                IRflag = 0;
            }
            else
            {
                lcdGoto(0,0);
                lcdString("Invalid Character!");
            }

            array[0] = count;
            array[1] = speed;
            array[2] = IRflag;

            return;
}
```

Main loop of program run on laptop:

```
//main loop
  while (!quit&&cvGrabFrame(capture))
  {
    IplImage *img = cvRetrieveFrame(capture);

      if (drop)
      {
              clearFrames(capture, img, 1);
              drop = false;
      }

    cvConvertScale(img, frame, 1, 0);

      IplImage* HSVOut = cvCreateImage(cvGetSize(frame), 8, 1);
      cvConvertImage(frame, HSVOut, 0);
      IplImage* HSVFrame = cvCreateImage(cvGetSize(frame), 8, 3);
      cvConvertImage(frame, HSVFrame, 0);

      cvSmooth(HSVFrame, HSVFrame);
      cvCvtColor(HSVFrame, HSVFrame, CV_BGR2HSV);

      if (choice == 0)
              CvFindHSV(HSVFrame, HSVOut, H, S, V, HSensibility, SSensibility,
VSensibility);
```

```
        else if (choice == 1)
                CvFindHSV(HSVFrame, HSVOut, H2, S2, V2, HSensibility, SSensibility + 50,
VSensibility + 50); //set robot vision based on what it's looking for
        else if (choice == 2)
                CvFindHSV(HSVFrame, HSVOut, H3, S3, V3, HSensibility, SSensibility + 50,
VSensibility + 50);
        else;

        cvSmooth(HSVOut, HSVOut);
        cvMorphologyEx(HSVOut, HSVOut, NULL, morphKernel, CV_MOP_CLOSE, 1);
    cvMorphologyEx(HSVOut, HSVOut, NULL, morphKernel, CV_MOP_OPEN, 1);

    cvShowImage("HSV", HSVOut);

    IplImage *labelImg = cvCreateImage(cvGetSize(frame), IPL_DEPTH_LABEL, 1);

    CvBlobs blobs;
    unsigned int result = cvLabel(HSVOut, labelImg, blobs);

        if (select == 3)
        {
                if (choice == 1)
                {
                        if (hcount < 30)
                        {
                                hcount++;
                                write_char(hCom, 0xFE);
                                drop = true;
                        }
                        else
                        {
                                hcount = 0;
                                select = 4;
                                write_char(hCom, 0xFE);
                                clearFrames(capture, frame, 15);
                                drop = true;
                        }
                }
                else
                {
                        if (hcount < 65)
                        {
                                hcount++;
                                write_char(hCom, 0xFE);
                                drop = true;
                        }
                        else
                        {
                                hcount = 0;
                                select = 4;
                                write_char(hCom, 0xFE);
                                clearFrames(capture, frame, 15);
                                drop = true;
                        }
                }
```

```
}

else if (result != 0)
{
        label = cvGreaterBlob(blobs);
        cvFilterByLabel(blobs, label);

        bottle = *blobs[label];

        center.x = frame->width/2;
        center.y = frame->height/2;
        centroid = bottle.centroid;

        if (choice == 1 || choice == 2) choiceOffset = 4500;
        else choiceOffset = 0;

        if(bottle.area > 200+(choiceOffset/2))
        {
                switch(select)
                {
                        case 0: //handle approaching bottle or
                        {
                                if (bottle.area > (5000 + choiceOffset))
                                {
                                        //printf("\n%d\n", bottle.area);
                                        write_char(hCom, 0x40);
                                        if (choice == 0)
                                                select = 1;
                                        else if (choice == 1 || choice == 2)
                                                select = 2;

                                        clearFrames(capture, frame, 20);

                                        message = read_char(hCom);

                                        if (message == 0x13)
                                                break;
                                        else
                                        {
                                                printf("Error reading message");
                                                break;
                                        }
                                }
                                else if (center.x > centroid.x)
                                {
                                        write_char(hCom, 0x30);
                                        break;
                                }
                                else if (center.x < centroid.x)
                                {
                                        write_char(hCom, 0x31);
                                        break;
                                }
                                else if (center.x == centroid.x)
                                {
```

```
                write_char(hCom, 0x32);
                break;
            }
            else
            {
                break;
            }
        }
        case 1: //handle aligning gripper with bottle
        {
            if (centroid.x <= frame->width/1.6)
            {
                write_char(hCom, 0x50);
                break;
            }
            else if (centroid.x >= frame->width/1.01)
            {
                write_char(hCom, 0x51);
                break;
            }
            else if (centroid.x > frame->width/1.6 && centroid.x
< frame->width/1.01)
            {
                write_char(hCom, 0x52);

                clearFrames(capture, frame, 30);
                message = read_char(hCom);

                if (message == 0x10)
                {
                    choice = 1;
                    select = 3;
                }
                else printf("\nError! Invalid message
received\n");

                clearFrames(capture, frame, 1);
                break;
            }
            else
            {
                break;
            }
        }
        case 2: //handle dump and trash
        {
            if (choice == 1)
            {
                write_char(hCom, 0x60);
            }
            else if (choice == 2)
            {
                write_char(hCom, 0x61);
            }
            else printf("\nError with choice value!\n");
```

```
                    clearFrames(capture, frame, 30);
                    message = read_char(hCom);

                    if (message == 0x11)
                    {
                        choice = 2;
                        select = 3;
                    }
                    else if (message == 0x12)
                    {
                        choice = 0;
                        select = 3;
                    }
                    else printf("\nError! Invalid message received\n");
                    break;
                }
                case 4:
                {
                    if (centroid.x <= frame->width/2.6)
                    {
                        write_char(hCom, 0x50);
                        break;
                    }
                    else if (centroid.x >= frame->width/1.85)
                    {
                        write_char(hCom, 0x51);
                        break;
                    }
                    else
                    {
                        write_char(hCom, 0xFE);
                        select = 0;
                        break;
                    }
                    drop = true;
                }
                default:
                {
                    printf("\nError: Invalid case\n");
                    break;
                }
            }
        }
        else
        {
            write_char(hCom, 0xFF);
            drop = true;
        }
    }
    else
    {
        write_char(hCom, 0xFF);
        drop = true;
    }
    cvRenderBlobs(labelImg, blobs, frame, frame, CV_BLOB_RENDER_BOUNDING_BOX);
    cvUpdateTracks(blobs, tracks, 200.0, 5);
    cvRenderTracks(tracks, frame, frame, 
CV_TRACK_RENDER_ID|CV_TRACK_RENDER_BOUNDING_BOX);

    cvShowImage("Blob Tracking Method", frame);

    cvReleaseImage(&labelImg);
    cvReleaseImage(&HSVOut);
      cvReleaseImage(&HSVFrame);

    k = cvWaitKey(25)&0xff;
    switch (k)
    {
      case 27:
      case 'q':
      case 'Q':
        quit = true;
        break;
    }

    cvReleaseBlobs(blobs);

  }
```