Formal Report

Project: “Murphy” Autonomous Robot

By Phong Truong

EEL 5666 Fall 2011

Dr. Arroyo, Dr. Schwartz
Tim Martin, Ryan Stevens

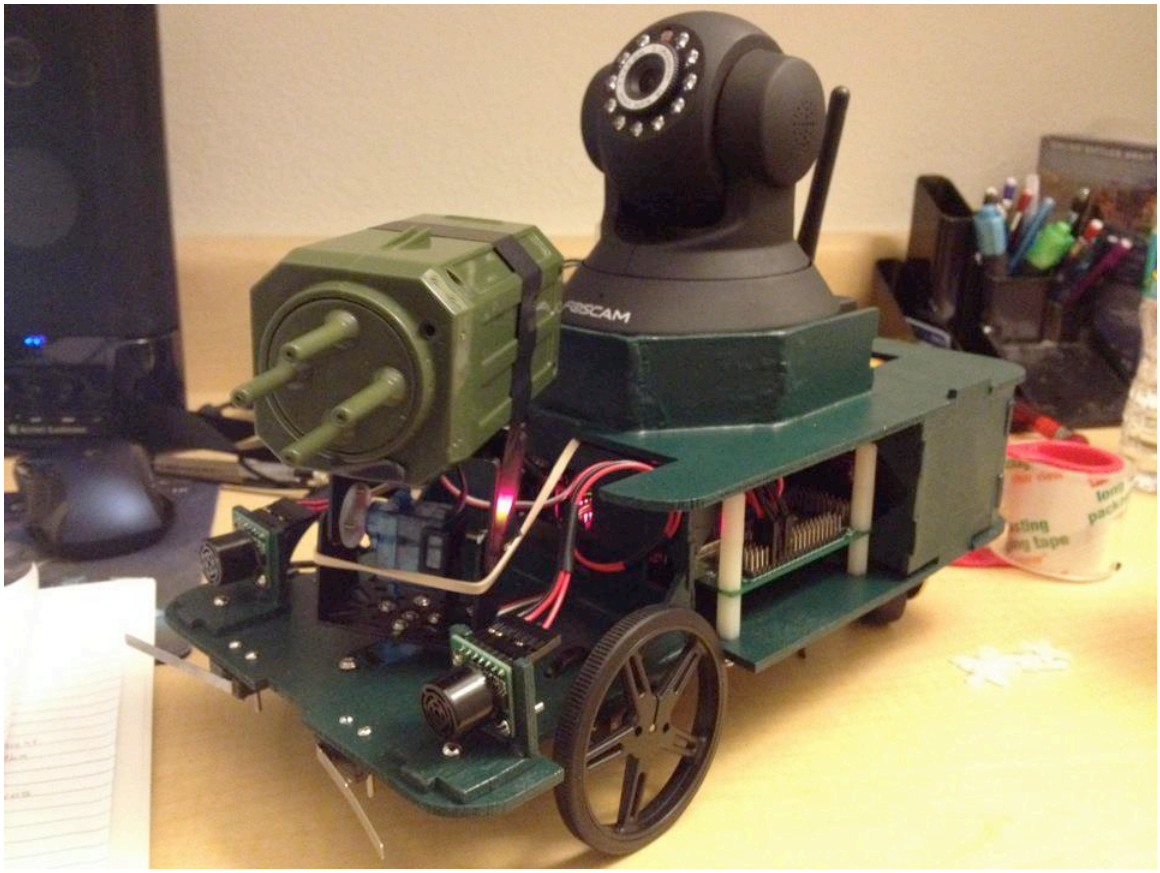

Table of Contents

I. Abstract

Autonomous robots are used in many dangerous situations to carry out functions to prevent loss of human life. Murphy was developed as an autonomous surveillance robot that is able to navigate in an indoor environment, find its target, and eliminate it. It is able to perform this function using an Epiphany DIY board, wireless IP camera, sonar and bump switches, and a dart launching system. The first objective of Murphy is to perform obstacle avoidance using sonar sensors for longer range obstacles and bump switches for close-range obstacles. Next, Murphy will track a target using a wireless IP camera that transmits video data over the Internet to a laptop for image processing. Pan and tilt commands from the camera are used to control the USB missile launcher's servos and the motors to navigate Murphy towards the target. Once the target has been established, the dart launching system will fire a dart at the target to eliminate it. In adapting this robot to real-world applications, the Murphy autonomous robot design could be retrofitted with a taser projectile system, rather than a dart launcher, to allow police officers to stun the suspect before moving in to bring them into custody. This would prevent injury and loss of life to both the suspect and the police officer.

II. Executive Summary

The purpose of this course project is to build an autonomous surveillance robot, which is able to detect targets and eliminate them with a dart launcher. The robot in this report, named Murphy, utilizes a Foscam Wireless IP camera along with OpenCV software to track blue-colored targets. Once a target is found, the position data is sent via Xbee to the robot's microcontroller board to control the pan/tilt servos and fire a dart. The pan/tilt servos are connected to the modified USB missile launcher. Obstacle avoidance is accomplished by using two front-mounted sonar sensors and four bump switches located on the front and rear bumpers of the robot. The left and right sonar analog values are compared against a set threshold, and the motors are set to either back up, turn hard left/right, turn left/right, move forward, or move forward fast. The robot platform was designed in SolidWorks and cut by hand using in-lab tools and equipment.

Overall, the robot's behavior met all of the desired objectives for this project. Obstacle avoidance performed smoothly, especially using proportional control to smooth out the motor speeds. Target tracking using color blob detection performed with little to no lag issues. However, utilizing more complicated algorithms resulted in much lag that rendered the code unusable at this time. The problem is a result of inherent limitations in Visual Studio decoding the camera's video stream.

Recommendations for the robot include an investigation into using other image processing methods, such as MATLAB. The center of gravity also needs to be moved forward. This can be done by moving the board to the rear and moving the battery packs towards the center of the platform. This will push the C.G. to a 50/50 distribution and eliminate skidding issues with the wheels during obstacle avoidance.

III. Introduction

A. Background

In many high-risk SWAT (Special Weapons and Tactics) Team missions, the objective is to find and apprehend a suspect as quickly and safely as possible. Unfortunately, not every suspect is cooperative, so SWAT members often have to enter the building to complete their mission and risk being killed by gunfire. Murphy, an autonomous surveillance and target-tracking robot, is the high-tech solution to this problem. Once inside the building, Murphy is capable of maneuvering around the building to find the suspects and eliminate them. It can also be used as a security robot for surveillance of the surrounding area.

B. Objectives

The objective of Murphy is to:

1. Perform obstacle avoidance using sonar sensors for distant objects and bump switches for closer objects.
2. Locate a specified colored target in a room using its wireless IP camera and process the position of the target with image processing software located on a separate laptop.
3. Relay the position data back to the robot via Xbee communication.
4. Use the position data to move the robot towards the target.
5. Fine-tune targeting with the pan/tilt servos and eliminate the target using the missile launcher.

IV. Integrated System

Theory of Operation:

The robot will begin by performing obstacle avoidance with the bump switches and sonar sensors when turned on. When an obstacle is detected by the sonar sensors, the robot will either back up or turn in a particular direction, depending on where the obstacle is located in relation to the two sensors. During obstacle avoidance, if an object is within close proximity to the robot and is out of range of the sonar sensors, the bump switches located at the front and rear of the robot will alert the robot to either move forward or backward. When a desired colored target is detected by the wireless IP camera, the robot will stop obstacle avoidance and begin target tracking. The video stream from the IP camera is sent to a laptop via a wireless router. Image processing software on the laptop will calculate the size of the color blobs and determine the position coordinates of the target. Once the image processing is complete, the position data will be sent to the robot via Xbee communication between the two Xbee modules, one of which is connected to the laptop's USB Xbee Explorer, and the other is connected to the microcontroller. See Figure 1 below for the component diagram.

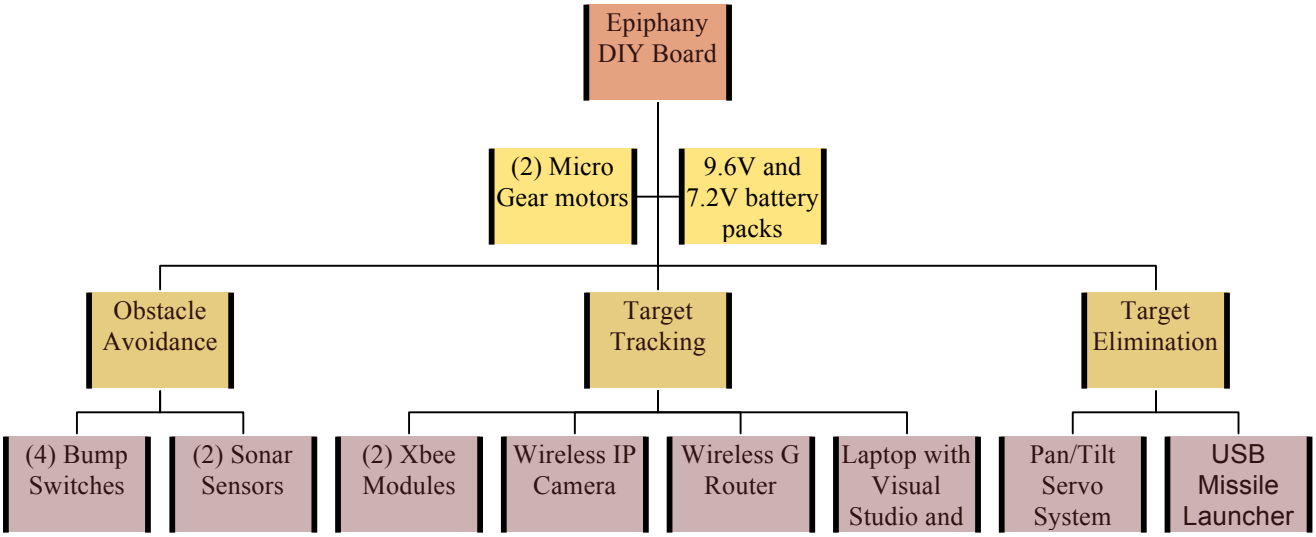

Figure 1: Component Diagram

After the robot receives the position data, the robot's behavior algorithm will have the servo pan/tilt system to position itself accordingly to shoot a dart at the target. If the target is out of range, the position data will be used to turn the robot to have the target be in range for shooting.

Microcontroller:

The Murphy autonomous robot is controlled by an Epiphany DIY microcontroller board sold by Out of the Box, Tim Martin's company. The Epiphany DIY has an 8/16 bit AVR XMEGA microcontroller and was purchased with the Servo Plus+ and Motor Quad packages, which include a 5V regulation circuit and two L6205 motor drivers with heat sinks. This allows the board to operate with up to 24 servos and run two DC motors up to 5.6A. Figure 2 shows the Epiphany DIY diagram.

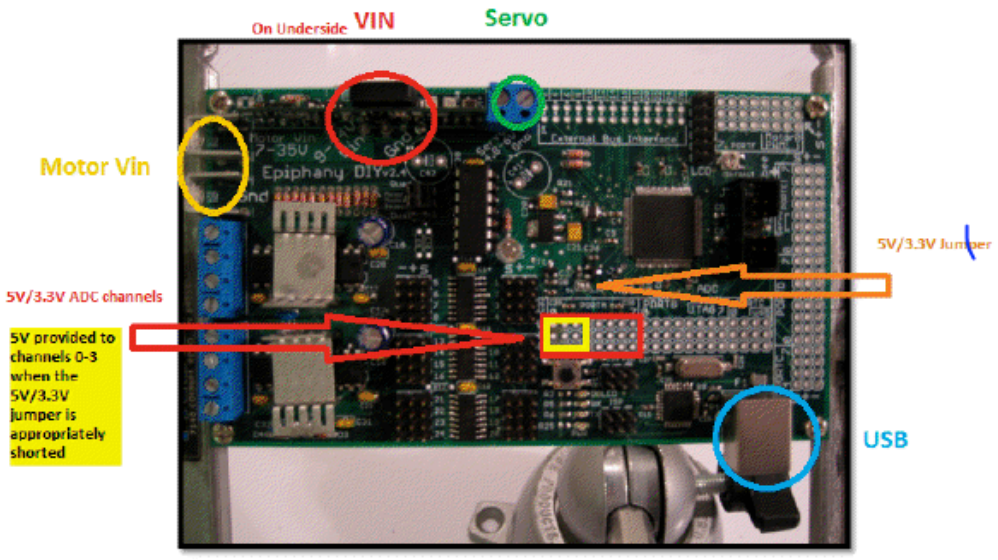

Figure 2: Epiphany DIY Microcontroller

Components:

For the robot to perform its duties, the following components were required:

- (2) Pololu 210:1 High-Power micro gearmotors with 80mm wheels
- (2) Close-range bump switches
- (2) MaxSonar EZ1 sonar sensors
- (2) Xbee chip antenna modules
- (1) USB Xbee Explorer
- (1) 9.6V NiMH battery pack
- (1) 7.2V NiMH battery pack
- (1) DreamCheeky USB missile launcher
- (1) Sparkfun mini servo pan/tilt system
- (1) Foscam Wireless IP Camera
- (1) Epiphany DIY microcontroller board
- (1) Dell XPS Laptop with AVR Studio 5 and Visual Studio 2010
- (1) Linksys Wireless G Router

Each component will be discussed in later sections of this report.

V. Software

The following software packages are required for this project:

- **Microsoft Visual Studio 2010:** Visual Studio is a C/C++ program that is used primarily to compile and run OpenCV code.

- **OpenCV 2.3:** OpenCV is a library of C/C++ or Python programming functions for real-time image processing. The functions are declared inside a Visual Studio main file. The proper library directories must be included in Visual Studio in order for the functions to run.

- **CMake 2.8:** CMake is an application that generates directories for OpenCV before running in Visual Studio.

- **AVR Studio 5:** AVR Studio is a program specifically for Atmel microcontrollers. All obstacle avoidance and target tracking code is written in an AVR Studio main file.

- **XCTU:** XCTU is a terminal program, which can be used to display data via USB or Xbee communication. Data such as sonar analog values, motor duty cycles, and target position data can be displayed in XCTU.

- **Chip45 Bootloader:** The bootloader is a program which uploads the hex file generated in AVR Studio to the microcontroller.

First and foremost, since the microcontroller is not performing any onboard image processing, it is critical to use a laptop that will have the proper performance requirements. The laptop of choice is a Dell XPS 1530 with a dual-core processor and NVIDIA 8600GT graphics card. It handled all programs simultaneously with no issues aside from the lag issues from the video stream in OpenCV. Image processing results will be discussed in Section X of this report.

Summary:

Since I had relatively no experience in C/C++, learning the programming language was difficult at first, but I found that it shared many of the same basic principles as MATLAB, a programming language which I had already mastered. Both Visual Studio and AVR Studio are C based. Another initial obstacle to overcome was understanding how to operate OpenCV inside Visual Studio. Since OpenCV is a library of programming functions, I needed to point Visual Studio to the correct directories, or the main code would not run. Also, running the OpenCV code with the Foscam wireless camera did cause some lag issues, which stemmed from the fact that Visual Studio could not efficiently processes the video stream from the camera.

In retrospect, learning Visual Studio and OpenCV did help me understand the C programming language better, but I may experiment with MATLAB's image processing in the future. MATLAB does not require any libraries to be installed or included, and many of the functions are already built into the software.

VI. Mobile Platform

The purpose of the mobile platform is to house the electronics and to provide a secure structure for the robot to perform its objectives, which are to perform object collision avoidance and target tracking. Murphy's mobile platform is a two-tiered wooden platform that uses two

80mm wheels, powered by DC gearbox motors, at the front and two 1 1/4" free-spinning caster wheels at the back. The platform was designed in SolidWorks (Figure 3) and then fabricated by hand using the in-lab cutting tools. The microcontroller, batteries, sonar sensors, and bump switches are located on the lower level. The motors and wheels are mounted on the underside of the lower platform using plastic brackets and screws. Also on the lower platform is the USB missile launcher, which is mounted on a mini servo pan/tilt system. The Foscam wireless IP camera is mounted on top to allow it to find the intended target.

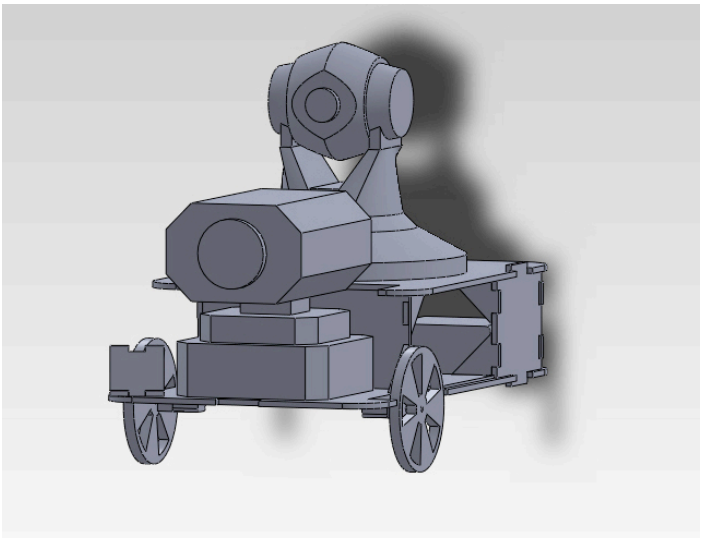

Figure 3: SolidWorks CAD model

Summary:

Originally, the platform was designed in SolidWorks with the intention of cutting it on the T-Tech machine in the lab. However, after talking with the TA's, I decided to cut the platform by hand because it would require at least 1-2 hours on the T-Tech machine. Cutting the platform manually became very labor intensive. The most difficult aspect was lining up the drill holes and ensuring all the parts are lined up correctly. However, after painting and refinishing the platform, it actually exceeded my expectations, despite the labor intensive process. If I were to redo the platform, I would have the TA's cut the board on the T-Tech machine in order to save valuable time.

Another design change I would make is to place the board towards the back of the platform and move the batteries towards the front. Most of the robot's weight is towards the rear, so I needed to place additional weight on the front of the platform, so the wheels would not skid when turning.

VII. Actuation

Motors:

The objective of using motors is to provide Murphy with movement. Murphy will navigate its environment using two Pololu 210:1 micro gear motors (Figure 4) as its front motors. The motors operate at 6V at 140 rpm, 70 mA free-run, 1.6A stall, and 50 oz-in stall torque. Since the operating torque was not listed on the website, the operating torque is assumed to be half of the stall torque (25 oz-in). The micro motors can operate safely in the range between 6-9V, so the robot will operate near the 9V value. A 12V motor was not selected because it would add additional weight to the robot and also require a much larger battery pack. Since the motor operates in the 6-9V range, a 7.2V battery pack was selected, which consists of six rechargeable NiMH AA batteries. The robot uses 80mm wheels to keep the center of gravity low in order to prevent the robot from tipping over.

Figure 4: 210:1 Pololu Micro Gearmotor with 80 mm wheel

The Epiphany board is capable of controlling up to four motors in total. The two Pololu motors were connected to the motor inputs on the board. Speed control of the motors is accomplished through adjusting the duty cycle in AVR Studio using the following code as an example:

```
setMotorDuty(1, x, MOTOR_DIR_FORWARD)
setMotorDuty(3, x, MOTOR_DIR_FORWARD)
```

The numbers 1 and 3 represent the motor input ports. Depending on how the motor was wired, MOTOR_DIR_FORWARD or MOTOR_DIR_BACKWARD will cause the robot to move forward or backwards. The variable x represents the motor duty cycle. The duty cycle ranges from 0-1024, but the minimum duty cycle to get the motor started is 600. Thus, 600 was set as the nominal speed rating. The purpose of the robot is to find targets and eliminate them using precise movements, so speed is not the goal. Therefore, the fastest duty cycle setting is at 800.

USB Missile Launcher:

Murphy utilizes a modified DreamCheeky USB missile launcher (Figure 5) purchased from Thinkgeek.com. Because the microcontroller cannot interface via USB, the missile launcher had to be disassembled and examined further. Inside its plastic housing were two gearmotors, which provide the pan and tilt controls. However, they were connected using a foreign motor driver, which would be too complicated to try to integrate to the Epiphany. It was

decided instead to utilize a mini servo pan/tilt system from Sparkfun.com for the missile launcher's pan and tilt capabilities. Since I purchased the Servo+ package, the servos were simply connected to the servo inputs on the microcontroller.

Only the actual launcher mechanism was used from the original product. Upon further analysis, it was a suction-spring device that was powered by a small motor. The launcher mechanism was mounted directly to the servo pan/tilt system and the motor connected to the Epiphany's motor inputs. As noted earlier, the Quad Power package was purchased, so the microcontroller can power a third motor for the missile launcher. The motor for the missile launcher is rated at 6V, so a 7.2V battery pack should be sufficient.

Figure 5: DreamCheeky missile launcher

Summary:

Using the micro gear motors for the robot was a great choice. They provide large amounts of torque while giving the robot more than enough speed. However, since they are much smaller than traditional gear motors, they are also much more fragile. While mounting the right motor after painting, one of the wire brackets broke off, which rendered the motor unusable. I ordered another motor from Pololu, which appeared much less powerful than the original motor. Then, I decided to solder the wires back onto the broken motor's damaged brackets and used enough solder to ensure the wires would not come loose. After powering the robot on, it turned out the broken motor was not broken after all, and I also had an extra motor in the event the original motor were to fail during Demo Day.

Regarding the missile launcher, it performed much better than I expected. By connecting its motor directly to the board, I could control when the missile launcher would fire. The only downside is that the robot did not know when each dart was fired, so I needed to specify the time between each shot in the main code. Although, I did come close to grinding the internal gears from testing various motor speeds, I would still use the same missile launcher for other robot applications.

VIII. Sensors

Sensors for Obstacle Avoidance:

Murphy will use both bump switches and sonar sensors for collision avoidance. Two Maxbotix LV MaxSonar EZ1 sensors (Sparkfun.com, Figure 6) are mounted on the front of the platform. Sonar sensors were used because it casts a much larger detection cone than an IR sensor, however, they are also more expensive. The sonar sensors have a 0-6.45m range, which is much larger than IR sensors. The sensors were originally placed facing forward, however, this created a dead zone directly in front of the robot. The sonar sensors were then rotated towards the center to eliminate this problem during obstacle avoidance.

Figure 6: MaxSonar EZ1 Sensor

Each sonar sensor is connected to an ADC port on the board and outputs an analog value between 210 – 2000, where 210 is the minimum range of detection for the sonar sensors. For obstacle avoidance purposes the following ranges were used:

Very Close: 250
Near: 300
Far: 350
Very Far: 400

Based on the left and right sonar readings, the robot will either back up, turn hard left or hard right, turn left or right, move forward, or move forward fast. Details on the actual behavior based on these readings will be discussed in Section VIII: Behaviors. A plot of the sonar analog values versus actual distance is shown below (Figure 7):

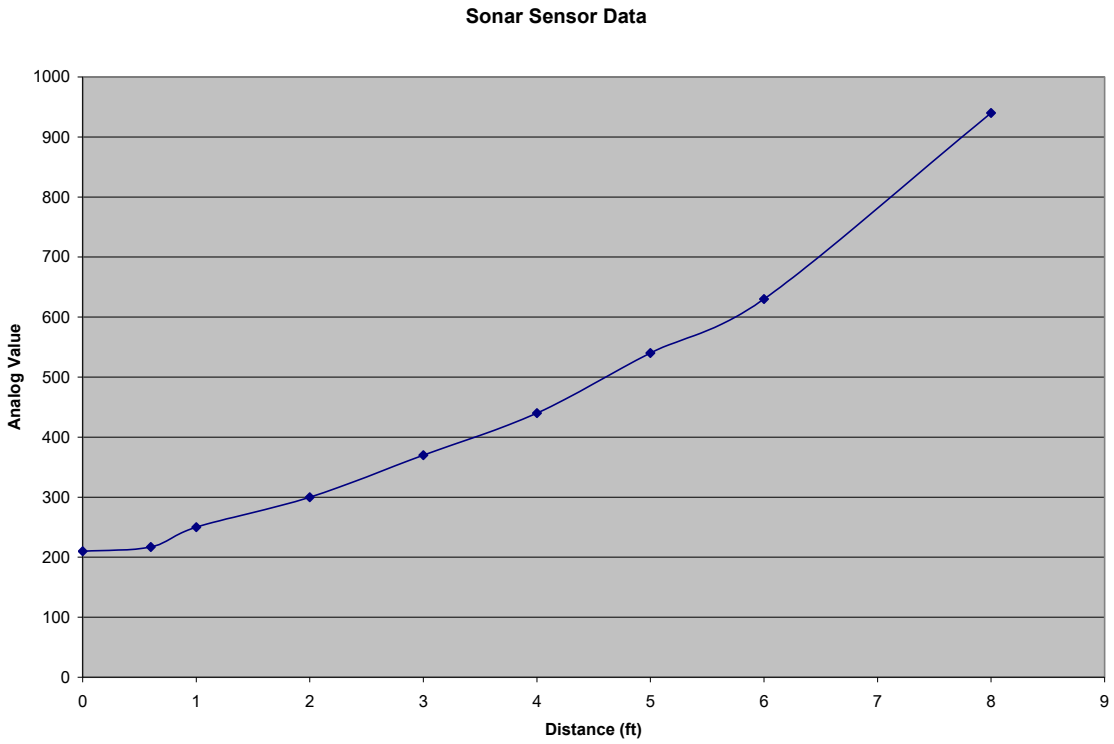

Figure 6: Sonar sensor analog values

As an added layer of protection, two bump switches are mounted on the front and back bumpers. Since sonar sensors will inherently have blind spots, the bump switches will help with collision avoidance in close-range situations. Each bump switch has three connections: normally open (NO), common (C), and normally closed (NC). The four bump switches are connected to Port D on the board, where the pins are set as inputs. NO is connected to GND (ground) and C is connected to S (signal) on Port D, such that triggering the bump switch will close the circuit and output a digital 1. Experimental data from these sensors are discussed later in Section X of this report.

Special Sensor System:

The Foscam IP wireless camera is Murphy's special sensor. Since the Foscam uses a 5V DC adapter, it is connected to the servo input on the microcontroller, which supplies regulated 5V power since I purchased the Servo+ package with the Epiphany board. The camera can transmit video footage via 802.11g Wi-Fi using a web interface or interfacing directly with the computer in ad-hoc mode. A Linksys Wireless G router was used to deliver the wireless connection between the laptop and the camera. Because of this fact, the user needs to log in to the camera's web interface when powering on the robot.

Using Visual Studio 2010 and OpenCV 2.3, a color blob detection code was developed, which tracks a blue colored object. The algorithm tracks all blue objects and calculates the average center of gravity of all detected blobs. Only the largest blobs are included in the calculation, and all small blobs are ignored. The average C.G. position is sent as an X and Y

coordinate to the robot through Xbee communication between the laptop's Xbee Explorer and the Xbee module on the board. The data is written to the Xbee's COM port (code by Chris Dobson) and stored in a ring buffer. Then, the robot pulls the position data from the ring buffer and moves the motors or the pan/tilt servos accordingly.

Running Visual Studio with the IPcamera did seem to cause lag problems when using very complicated algorithms, such as facial recognition. Although I did successfully write a code to have the robot recognize specific targets based on facial recognition, there were severe lag issues that existed with the IP camera. These problems did not occur when using the laptop webcam. After much research, it was concluded that Visual Studio could not use the ffmpeg libraries to decode the IP camera's video stream, which is vital for the image processing to function properly. Unfortunately, I could not use the facial recognition code and instead had to use simple color blob detection for the robot, which did not have much lag due to its simple algorithm.

The camera has a 67 degree viewing angle with 270 degrees pan and 120 degrees tilt. This gives the robot a large detection cone without requiring it to turn constantly to find a target. However, it was discovered quickly that movement of the camera would be extremely difficult without programming the camera's ARM processor. I decided to leave the camera stationary and instead move the pan/tilt servos to point the missile launcher at the intended target.

Summary:

At the outset, I originally intended for both the camera and missile launcher to move simultaneously when tracking the target. However, I could not find any information on programming the camera's ARM processor, so the camera had to be stationary. In retrospect, I would have preferred to purchase a stationary IP camera and mount it on a pan/tilt servo so that both the camera and missile launcher pan/tilt servos move simultaneously with little programming required.

Also, it was disappointing to not be able to utilize the facial recognition algorithm, which would have made the robot behavior much more interesting. Since the issue seemed to originate from Visual Studio's problems with the video stream, I would probably use MATLAB's image processing in the future to avoid the lag issues.

Algorithm

Target?

Yes

IX. Behaviors

No

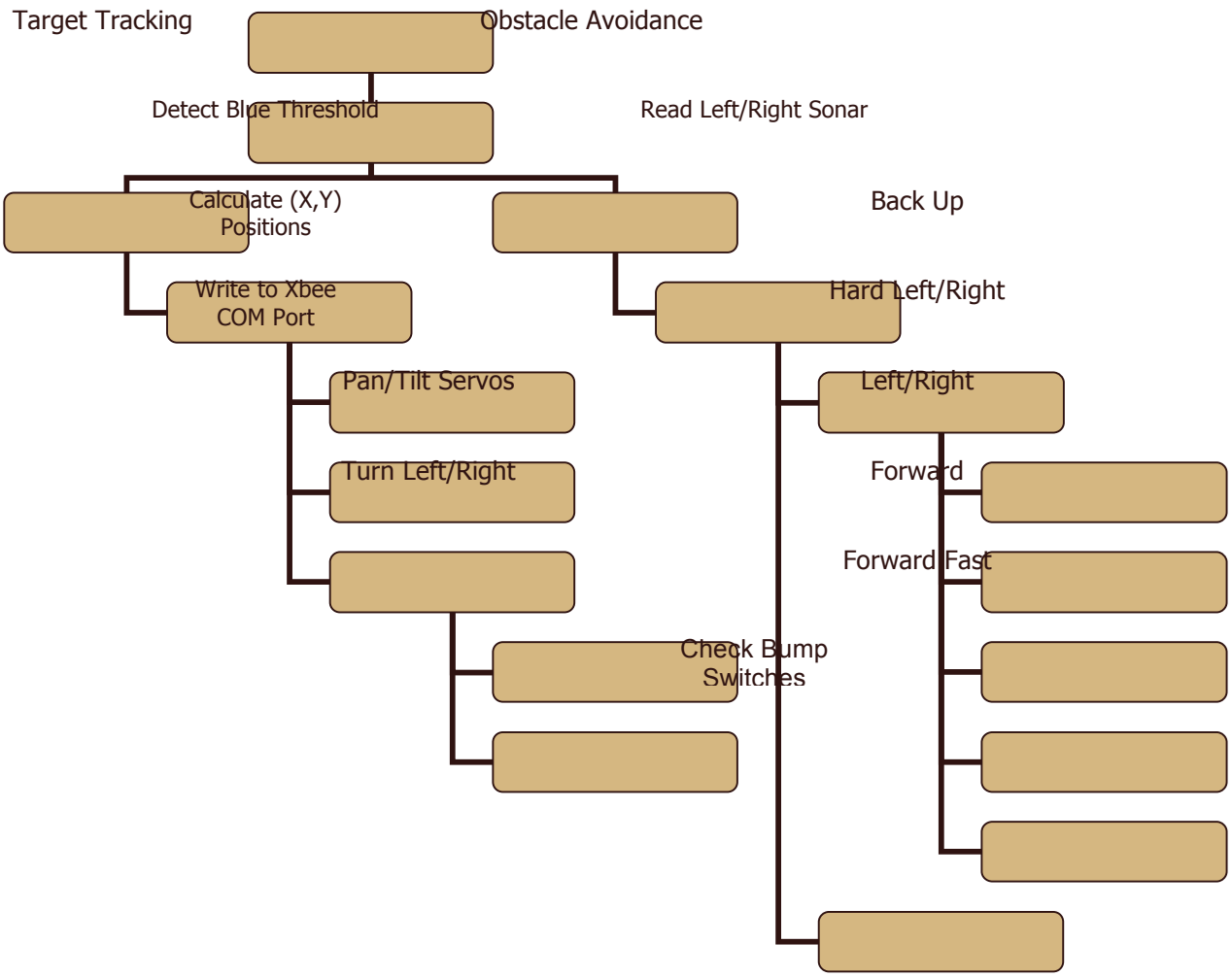

Figure 8: Behavior Diagram

Obstacle Avoidance:

Murphy has on/off switches for both the motors and the board. When the board is switched on, the robot will gather data from the sonar sensors and determine if a blue-colored target is in the vicinity. This is assuming the user has already logged into the camera's interface and run the code in Visual Studio. Then the motors are switched on. If no target is detected, the robot will enter obstacle avoidance mode where it will continuously gather data from the sonar sensors and move the robot in a particular direction. The sonar threshold levels are listed below:

Threshold:
Very_close: 250
Near: 300
Far: 350
Very_far: 400

Behavior:
Left_sonar < 250 && Right_sonar < 250: back up
Left_sonar < 250: hard right
Right_sonar < 250: hard left
250 < Left_sonar || Right_sonar < 300: turn left or right
300 < Left_sonar && Right_sonar < 350: forward
Left_sonar && Right_sonar > 400: forward fast

Since the minimum value for detection was 210, setting *Very_close* to 250 allowed the robot enough time to back up or turn hard left/right without hitting the obstacle. Proportional control was used to smooth out the motor speeds and minimize the jerkiness during obstacle avoidance.

speed = (K/(K+1))*oldvalue + (1/(K+1))*newvalue; K >= 0

If a bump switch is triggered during obstacle avoidance, the robot will stop and move in the opposite direction to avoid the object.

Target Tracking:

The X, Y position data is continuously being written to the ring buffer and sent to the robot's Xbee at all times. If there is no blue target available, the position data will be 0 for both X and Y, and obstacle avoidance will run while position == 0. If position /= 0, then target tracking will run while this statement remains true. The data sent to the ring buffer can only be 8 bytes long and be in the range of 0-255. The X coordinates were set in the range of 0-240, and the Y coordinates set in the range of 240-254 as shown below:

Position = 0 – 240; X coordinates
Position = 241 – 255; Y coordinates
Position = 255; target in crosshairs

Since the missile launcher does not have range much longer than 10 feet, the vertical direction does not need to be as precise as the horizontal direction. Lastly, since the camera operates at a 320 x 240 resolution, the X coordinates were multiplied by a factor of 240/320. The Y coordinates by a factor of 255/240 where 241 corresponds to the top of the screen and 254 corresponds to the bottom of the screen. If position = 255, then the target is already in the center of the screen, and the missile launcher is ready to shoot.

The code in AVR studio checks the ring buffer for position data and follows the following routine:

```
While (position == 0);
        Perform obstacle avoidance
        Check ring buffer

While (position /= 0);
```

```
Perform target tracking

If (position > 0 && position <= 40); //target is far left
        Turn left
If (position > 40 && position <= 200); //target is within range
        Adjust pan/tilt servos
        Shoot launcher
If (position >200 && position <= 240); //target is far right
        Turn right
If (position == 255); //target is within crosshairs
        Shoot launcher
```

Summary:

Murphy's obstacle avoidance code runs smoothly aside from the fact that I needed to turn the sonar sensors towards the center to correct for the dead zone. This improved the robot's response time and made obstacle avoidance more accurate. Also, the robot did experience some wheel skidding issues because of the C.G. imbalance towards the rear, but this was corrected with a counter weight near the front. Target tracking using the camera and missile launcher works fairly well. The missile launcher's motors need to be manually turned on and off in order for it to shoot the darts because it has no way to determine whether a dart has been launched or not. The only addition needed is a long-range IR sensor that would be mounted to the top of the missile launcher. This would allow the robot to verify whether it is firing at an actual target.

X. Experimental Layout and Results

Obstacle Avoidance:

To begin obstacle avoidance testing, several measurements were taken to set the thresholds for the sonar sensors as discussed in the previous section. Based on these thresholds for the left and right sonar sensors, the robot either backs up, turns hard left/right, turns left/right, moves forward, or moves forward fast. For each case, text is printed to the LCD screen or XCTU (Figure 9), so the user knows what the robot is doing.

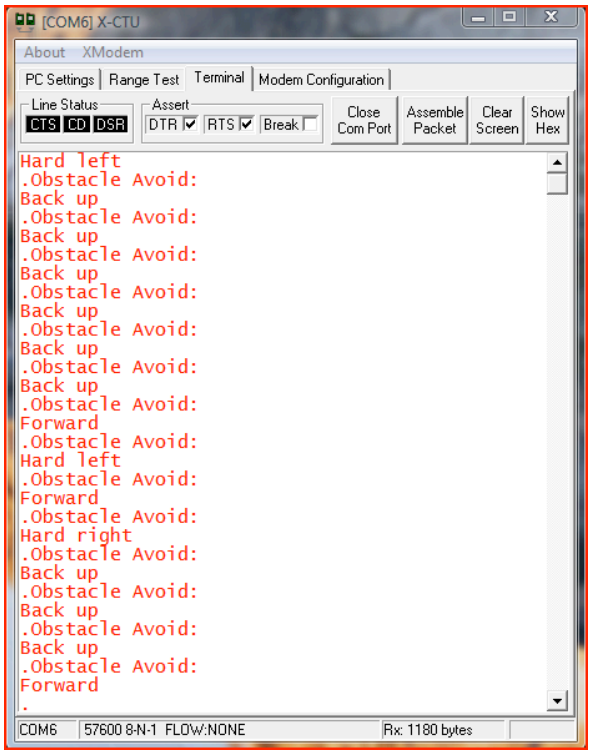

Figure 9: XCTU display

Image Processing:

The OpenCV 2.3 libraries and Visual Studio 2010 are used for image processing to determine the position of the colored targets. The image processing algorithm writes an (X, Y) average C.G. position of the colored target to the Xbee COM port. When running the code, three windows automatically open up: video (Figure 10), thresh (Figure 11), and a display console (Figure 12). The video window shows the real-time video stream with a bounded rectangle over the colored target, which is blue in this case. The thresh window shows the threshold of blue detected on the object. The threshold is a range of HSV values, which were selected in Microsoft Paint to detect a wide range of blue colors in a variety of lighting conditions. To minimize false positives, the camera detects only the larger color blobs in the image. When analyzing the thresholded image, the lighting has a noticeable impact on the color detection. In dim lighting, the camera had a more difficult time detecting the blue color. This is because the camera is only 0.3MP and has limited color schemes.

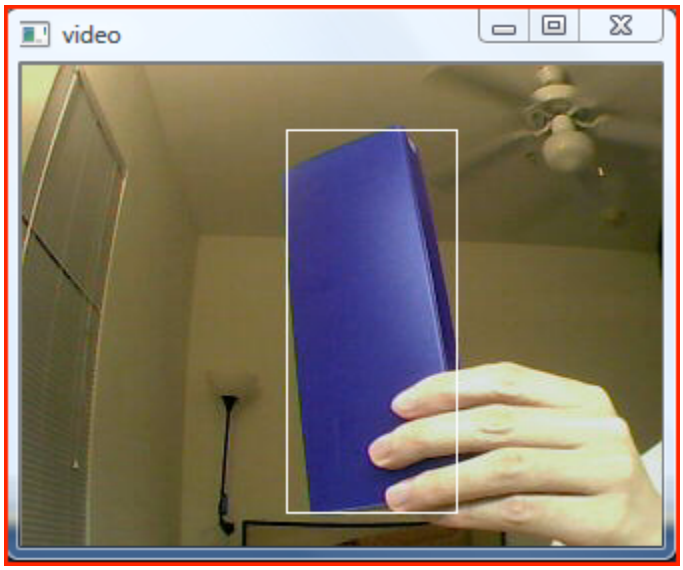

Figure 10: Real-time video stream

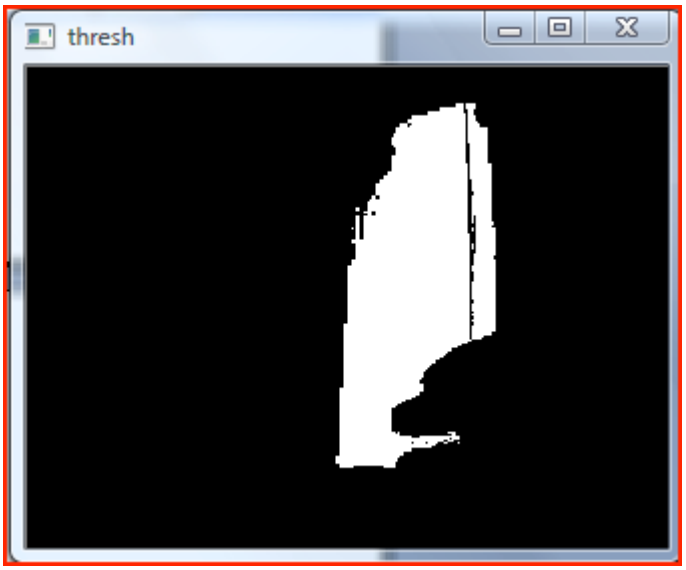

Figure 11: Thresholded image

The display window shows the position data calculated. The X value varies from 0-244, the Y value varies from 245-254. A value of 255 indicates the target is in the center and ready to be eliminated.

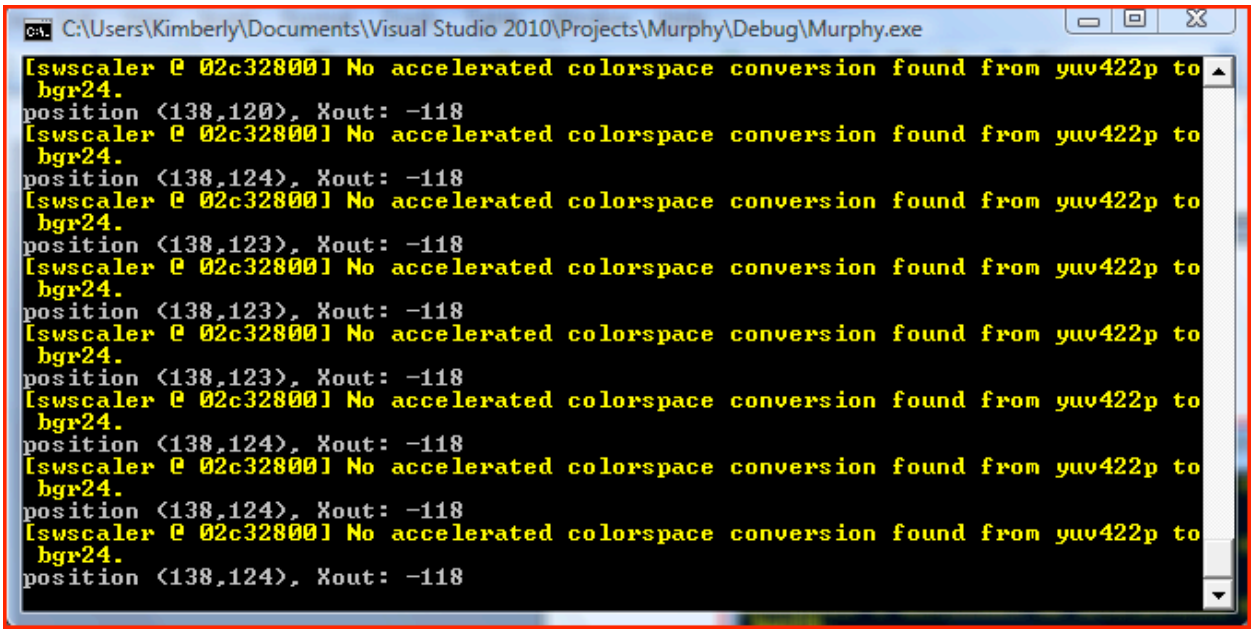

Figure 12: position output

Notice that in the display window (Figure 12) that there is a constant error that states, "No accelerated colorspace conversion found…" This is because of the lack of ffmpeg support in Visual Studio 2010. The ffmpeg library is a standard library used to decode a variety of media types, including video streams. While the code still runs with this error, it does cause some delay in the video stream. Unfortunately, this error cannot be corrected unless a different type of software is used, such as MATLAB.

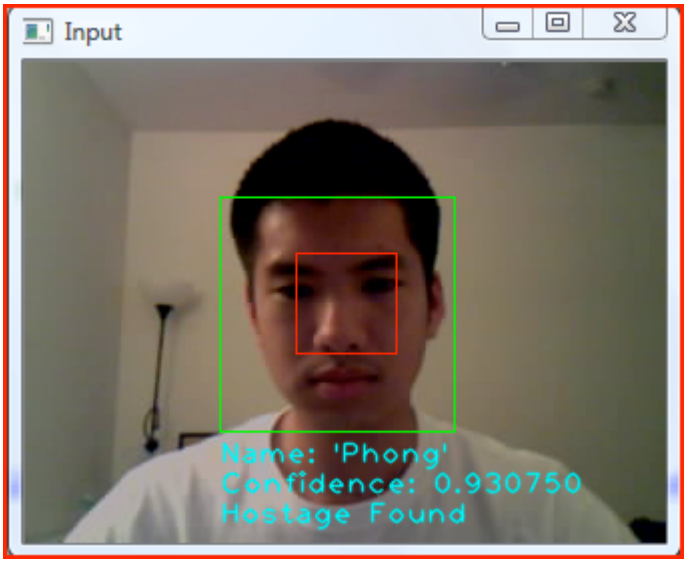

Figure 13: Facial recognition

As discussed earlier, facial recognition code was prepared for the robot (Figure 13). This code captures many images of the users face at different angles and compiles them into an average image. When a face is detected, it is compared against the average image for

compatibility, and a confidence value is shown. In the example above, I also included a red box to designate the bulls eye range for the missile launcher. Because of the ffmpeg errors, this code could not be run without major lag issues with the wireless IP camera, but it did run fairly smoothly with the laptop web cam. Future improvements with the coding and/or a switch to MATLAB would enable successful implementation of facial recognition.

Summary:

Obstacle avoidance for the robot runs fairly smooth with the proportional control allowing the motors to transition speeds without harmful voltage spikes. The image processing with color blob detection encountered some errors with ffmpeg compatibility, but it eventually ran without much lag issues. However, the lag only increased with more complex image processing algorithms.

One important lesson to learn is to ALWAYS back up your data. With less than 5 days before the Media Day demonstration, my laptop's hard drive failed, and the data was unrecoverable. Luckily, I have a younger sibling with another Dell XPS laptop with the same exact specifications who was willing to let me borrow it for the time being. It was a chore to reinstall AVR Studio, Visual Studio, and OpenCV again, but at least I did not lose my work.

XI. Conclusion

Summary:

Overall, I accomplished nearly all of the objectives set out at the beginning of the semester. My final hand-built platform looks almost exactly like the CAD model that I designed in September. It exceeded my expectations because all of the components fit inside the housing with no wasted space. The front area for the missile launcher provided enough clearance for the missile launcher to pan and tilt. Carefully modeling and dimensioning the parts in SolidWorks paid off in that respect. Obstacle avoidance for the robot works without any problems other than some occasional skidding issues with the wheels, but this is because of the C.G. imbalance. Target tracking for the robot is nearly one-to-one with color blob detection. Once a blue target is detected, the pan/tilt servos are able to move the missile launcher to the general location of the target and fire darts at it.

Limitations:

During the project, I could not program the camera to move to perform target tracking. Although this is not absolutely necessary, it would have made the targeting more versatile. Another problem I encountered was lag issues while running the image processing in Visual Studio, but this problem cannot be fixed at this time. Another obvious limitation is budget. Being an Aerospace Engineering student, I did not have any of the necessary tools or materials for this project, so I had to buy all of the tools, which was very expensive. After spending over

$750 total on materials, tools, and shipping, I simply did not have enough funds to purchase additional components, such as an IR sensor to mount on the missile launcher.

Future Work:

If given the opportunity to redo the project, I would have taken extra time to research all of the components needed for the robot and order them at all at once to save on shipping costs. I paid a lot of return shipping costs initially because I had ordered the wrong parts. I would also have the entire platform cut on the T-Tech or similar cutting machine to avoid spending time cutting and drilling the platform by hand. Then, I would relocate the board to the rear and move the batteries towards the center to have a 50/50 weight distribution.

Instead of the wireless IP camera, I would try using a web cam and building a wireless adapter for it to eliminate the need for a router altogether. MATLAB image processing would be investigated further to see if it could provide better performance as compared to OpenCV running in Visual Studio.

XII. Documentation

References:

1. Society of Robots – www.societyofrobots.com
2. IMDL - http://www.mil.ufl.edu/5666/
3. OpenCV 2.3 - http://opencv.willowgarage.com/wiki/
4. cvblobslib code - http://opencv.willowgarage.com/wiki/cvBlobsLib
5. Tim Martin (general TA assistance)
6. Ryan Stevens (general TA assistance)
7. Chris Dobson (COM port code)

Vendors:

1. Radioshack – www.radioshack.com
2. Pololu – www.pololu.com
3. Amazon – www.amazon.com
4. Sparkfun – www.sparkfun.com

XIII. Appendices

AVR Studio Code:

```
#include <asf.h>
#include <avr/io.h>
#include <ctype.h>
#include <stdint.h>
#include <stdio.h>
```

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

volatile uint8_t position = 0;

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.
	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);//as can be seen in the schematic.  This uart is connected
to the USB port.  This function initializes this uart
	ADCsInits();//this function initializes the ADCs inside the Xmega
	LCDInit();

	//you should add any further initializations here
	sei();
	fprintf_P(&lcd_str,PSTR("Out of the Box\nEpiphany DIY\r\n"));
	fprintf_P(&USB_str,PSTR("Out of the Box:\rElectronics and Robotics\rPresents
the\rEpiphany DIY\rSoftware Version %s%s\r\n"),sVersion,date);

	TCC0.PER = 31250;
	TCC0.CTRLA = 7;

	//you should add any further initializations here

setServoAngle(90, 1);
setServoAngle(70, 2);

while(1){

// //////////bump switch initialization/////////
```

```
//NO=ground, C=signal; open=1, closed=0

PORTD_DIR = 0x00;

//011 pullup; 001 rising edge
PORTD.PIN0CTRL = 0x18; //00011001
PORTD.PIN1CTRL = 0x18;
PORTD.PIN2CTRL = 0x18;
PORTD.PIN3CTRL = 0x18;
//001 both edges, 0x18

//////////obstacle avoidance/////////

//grab data from buffer
if (dataInBufE1()){
	fscanf(&uartE1_str,"%c",&position);
}
_delay_ms(20);

while (position == 0){

	 //////////check sonar values/////////
	int lsonar0;
	int rsonar0;
	int lsonar1;
	int rsonar1;

	//check left sonar
	ADCA_request(1,1);
	_delay_ms(15);

	while(!ADCA_CH1_ConvComplete);
		lsonar0 = ADCA_getVal(1);
		_delay_ms(15);

	ADCA_request(1,1);
	_delay_ms(15);

	while(!ADCA_CH1_ConvComplete);
		lsonar1 = ADCA_getVal(1);
		_delay_ms(15);

	//check right sonar
	ADCA_request(2,2);
	_delay_ms(15);

	while(!ADCA_CH2_ConvComplete);
		rsonar0 = ADCA_getVal(2);
		_delay_ms(15);

	ADCA_request(2,2);
	_delay_ms(15);

	while(!ADCA_CH2_ConvComplete);
		rsonar1 = ADCA_getVal(2);
		_delay_ms(15);
```

```
	fprintf(&USB_str, "L: (%d, %d); R: (%d, %d)\r\n", lsonar0, lsonar1, rsonar0,
rsonar1);

	//sonar ranges
	int verynear = 250;
	int near = 300;
	int far = 350;
	int veryfar = 400;

	//duty cycle scaler
	int D = 375;

	//duty cycles
	//int veryslow = verynear+D; //675
	//int slow = near+D; //725
	//int fast = far+D; //775
	//int veryfast = veryfar+D; //825

	int speed;
	//int K = 1;
	//speed = (K/(K+1))*oldvalue + (1/(K+1))*newvalue // k>=0

	//obstacle very near to both sonars; back up
	if( lsonar1 < verynear && rsonar1 < verynear ){
		speed = (0.5)*(lsonar0+D) + (0.5)*(lsonar1+D);
		setMotorDuty(1, speed, MOTOR_DIR_BACKWARD_gc);
		setMotorDuty(3, speed, MOTOR_DIR_BACKWARD_gc);

		//rear left bumper, pin 2
		if (PORTD.IN & 0x04){
			//fprintf(&USB_str, "nothing\r\n");
		}
		else{
			_delay_ms(20);
			setMotorDuty(1, 700, MOTOR_DIR_FORWARD_gc);
			setMotorDuty(3, 700, MOTOR_DIR_FORWARD_gc);
			_delay_ms(500);
			setMotorDuty(1, 0, MOTOR_DIR_FORWARD_gc);
			setMotorDuty(3, 0, MOTOR_DIR_FORWARD_gc);
			fprintf(&USB_str, "Obstacle Avoid:\rRL bumper\r\n");
		}

		//rear right bumper, pin 3
		if (PORTD.IN & 0x08){
			//fprintf(&USB_str, "nothing\r\n");
		}
		else{
			_delay_ms(20);
			setMotorDuty(1, 700, MOTOR_DIR_FORWARD_gc);
			setMotorDuty(3, 700, MOTOR_DIR_FORWARD_gc);
			_delay_ms(500);
			setMotorDuty(1, 0, MOTOR_DIR_FORWARD_gc);
			setMotorDuty(3, 0, MOTOR_DIR_FORWARD_gc);
			fprintf(&USB_str, "Obstacle Avoid:\rRR bumper\r\n");
		}

		_delay_ms(500);
		fprintf(&USB_str, "Obstacle Avoid:\rBack up\r\n");
```

```
}

//obstacle very near to left sonar; turn hard right
else if( lsonar1 < verynear && rsonar1 > verynear ){
	speed = (0.5)*(lsonar0+D) + (0.5)*(lsonar1+D);
	setMotorDuty(1, speed, MOTOR_DIR_FORWARD_gc);
	setMotorDuty(3, 0, MOTOR_DIR_FORWARD_gc);

		//front left bumper, pin 0
		if (PORTD.IN & 0x01){
			//fprintf(&USB_str, "nothing\r\n");
		}
		else{
			_delay_ms(20);
			setMotorDuty(1, 700, MOTOR_DIR_BACKWARD_gc);
			setMotorDuty(3, 700, MOTOR_DIR_BACKWARD_gc);
			_delay_ms(500);
			setMotorDuty(1, 0, MOTOR_DIR_BACKWARD_gc);
			setMotorDuty(3, 0, MOTOR_DIR_BACKWARD_gc);
			fprintf(&USB_str, "Obstacle Avoid:\rFL bumper\r\n");
		}

		//front right bumper, pin 1
		if (PORTD.IN & 0x02){
			//fprintf(&USB_str, "nothing\r\n");
		}
		else{
			_delay_ms(20);
			setMotorDuty(1, 700, MOTOR_DIR_BACKWARD_gc);
			setMotorDuty(3, 700, MOTOR_DIR_BACKWARD_gc);
			_delay_ms(500);
			setMotorDuty(1, 0, MOTOR_DIR_BACKWARD_gc);
			setMotorDuty(3, 0, MOTOR_DIR_BACKWARD_gc);
			fprintf(&USB_str, "Obstacle Avoid:\rFR bumper\r\n");
		}

	_delay_ms(250);
	fprintf(&USB_str, "Obstacle Avoid:\rHard right\r\n");
	}

//obstacle very near to right sonar; turn hard left
else if( rsonar1 < verynear && lsonar1 > verynear ){
	speed = (0.5)*(rsonar0+D) + (0.5)*(rsonar1+D);
	setMotorDuty(1, 0, MOTOR_DIR_FORWARD_gc);
	setMotorDuty(3, speed, MOTOR_DIR_FORWARD_gc);

		//front left bumper, pin 0
		if (PORTD.IN & 0x01){
			//fprintf(&USB_str, "nothing\r\n");
		}
		else{
			_delay_ms(20);
			setMotorDuty(1, 700, MOTOR_DIR_BACKWARD_gc);
			setMotorDuty(3, 700, MOTOR_DIR_BACKWARD_gc);
			_delay_ms(500);
			setMotorDuty(1, 0, MOTOR_DIR_BACKWARD_gc);
			setMotorDuty(3, 0, MOTOR_DIR_BACKWARD_gc);
			fprintf(&USB_str, "Obstacle Avoid:\rFL bumper\r\n");
```

```
                }

                //front right bumper, pin 1
                if (PORTD.IN & 0x02){
                        //fprintf(&USB_str, "nothing\r\n");
                }
                else{
                        _delay_ms(20);
                        setMotorDuty(1, 700, MOTOR_DIR_BACKWARD_gc);
                        setMotorDuty(3, 700, MOTOR_DIR_BACKWARD_gc);
                        _delay_ms(500);
                        setMotorDuty(1, 0, MOTOR_DIR_BACKWARD_gc);
                        setMotorDuty(3, 0, MOTOR_DIR_BACKWARD_gc);
                        fprintf(&USB_str, "Obstacle Avoid:\rFR bumper\r\n");
                }

        _delay_ms(250);
        fprintf(&USB_str, "Obstacle Avoid:\rHard left\r\n");
    }

    //obstacle is near both sensors; turn left or right
    else if( (lsonar1 > verynear && lsonar1 < near) && (rsonar1 > verynear && rsonar1
< near) ){
        speed = (0.5)*(lsonar0+D) + (0.5)*(lsonar1+D);

        //turn left
        if (lsonar1 > rsonar1){
                setMotorDuty(1, speed, MOTOR_DIR_FORWARD_gc);
                setMotorDuty(3, speed*0.75, MOTOR_DIR_FORWARD_gc);

                //front left bumper, pin 0
                if (PORTD.IN & 0x01){
                        //fprintf(&USB_str, "nothing\r\n");
                }
                else{
                        _delay_ms(20);
                        setMotorDuty(1, 700, MOTOR_DIR_BACKWARD_gc);
                        setMotorDuty(3, 700, MOTOR_DIR_BACKWARD_gc);
                        _delay_ms(500);
                        setMotorDuty(1, 0, MOTOR_DIR_BACKWARD_gc);
                        setMotorDuty(3, 0, MOTOR_DIR_BACKWARD_gc);
                        fprintf(&USB_str, "Obstacle Avoid:\rFL bumper\r\n");
                }

                //front right bumper, pin 1
                if (PORTD.IN & 0x02){
                        //fprintf(&USB_str, "nothing\r\n");
                }
                else{
                        _delay_ms(20);
                        setMotorDuty(1, 700, MOTOR_DIR_BACKWARD_gc);
                        setMotorDuty(3, 700, MOTOR_DIR_BACKWARD_gc);
                        _delay_ms(500);
                        setMotorDuty(1, 0, MOTOR_DIR_BACKWARD_gc);
                        setMotorDuty(3, 0, MOTOR_DIR_BACKWARD_gc);
                        fprintf(&USB_str, "Obstacle Avoid:\rFR bumper\r\n");
                }
```

```
                _delay_ms(100);
                fprintf(&USB_str, "Obstacle Avoid:\rTurn left\r\n");
        }

        //turn right
        else{
        setMotorDuty(1, speed*0.75, MOTOR_DIR_FORWARD_gc);
        setMotorDuty(3, speed, MOTOR_DIR_FORWARD_gc);

                //front left bumper, pin 0
                if (PORTD.IN & 0x01){
                        //fprintf(&USB_str, "nothing\r\n");
                }
                else{
                        _delay_ms(20);
                        setMotorDuty(1, 700, MOTOR_DIR_BACKWARD_gc);
                        setMotorDuty(3, 700, MOTOR_DIR_BACKWARD_gc);
                        _delay_ms(500);
                        setMotorDuty(1, 0, MOTOR_DIR_BACKWARD_gc);
                        setMotorDuty(3, 0, MOTOR_DIR_BACKWARD_gc);
                        fprintf(&USB_str, "Obstacle Avoid:\rFL bumper\r\n");
                }

                //front right bumper, pin 1
                if (PORTD.IN & 0x02){
                        //fprintf(&USB_str, "nothing\r\n");
                }
                else{
                        _delay_ms(20);
                        setMotorDuty(1, 700, MOTOR_DIR_BACKWARD_gc);
                        setMotorDuty(3, 700, MOTOR_DIR_BACKWARD_gc);
                        _delay_ms(500);
                        setMotorDuty(1, 0, MOTOR_DIR_BACKWARD_gc);
                        setMotorDuty(3, 0, MOTOR_DIR_BACKWARD_gc);
                        fprintf(&USB_str, "Obstacle Avoid:\rFR bumper\r\n");
                }

        _delay_ms(100);
        fprintf(&USB_str, "Obstacle Avoid:\rTurn right\r\n");
        }

}

//no obstacle in sight; go forward very fast
else if( lsonar1 > veryfar && rsonar1 > veryfar ){
        speed = 800;
        setMotorDuty(1, speed, MOTOR_DIR_FORWARD_gc);
        setMotorDuty(3, speed, MOTOR_DIR_FORWARD_gc);
        _delay_ms(50);

        //front left bumper, pin 0
        if (PORTD.IN & 0x01){
                //fprintf(&USB_str, "nothing\r\n");
        }
        else{
                _delay_ms(20);
                setMotorDuty(1, 700, MOTOR_DIR_BACKWARD_gc);
                setMotorDuty(3, 700, MOTOR_DIR_BACKWARD_gc);
```

```
			_delay_ms(500);
			setMotorDuty(1, 0, MOTOR_DIR_BACKWARD_gc);
			setMotorDuty(3, 0, MOTOR_DIR_BACKWARD_gc);
			fprintf(&USB_str, "Obstacle Avoid:\rFL bumper\r\n");
		}

		//front right bumper, pin 1
		if (PORTD.IN & 0x02){
			//fprintf(&USB_str, "nothing\r\n");
		}
		else{
			_delay_ms(20);
			setMotorDuty(1, 700, MOTOR_DIR_BACKWARD_gc);
			setMotorDuty(3, 700, MOTOR_DIR_BACKWARD_gc);
			_delay_ms(500);
			setMotorDuty(1, 0, MOTOR_DIR_BACKWARD_gc);
			setMotorDuty(3, 0, MOTOR_DIR_BACKWARD_gc);
			fprintf(&USB_str, "FR bumper\r\n");
		}

		fprintf(&USB_str, "Obstacle Avoid:\rForward fast\r\n");
	}

	//obstacle somewhere in between; go forward
	else{
		speed = 750;
		setMotorDuty(1, speed, MOTOR_DIR_FORWARD_gc);
		setMotorDuty(3, speed, MOTOR_DIR_FORWARD_gc);
		_delay_ms(50);

		//front left bumper, pin 0
		if (PORTD.IN & 0x01){
			//fprintf(&USB_str, "nothing\r\n");
		}
		else{
			_delay_ms(20);
			setMotorDuty(1, 700, MOTOR_DIR_BACKWARD_gc);
			setMotorDuty(3, 700, MOTOR_DIR_BACKWARD_gc);
			_delay_ms(500);
			setMotorDuty(1, 0, MOTOR_DIR_BACKWARD_gc);
			setMotorDuty(3, 0, MOTOR_DIR_BACKWARD_gc);
			fprintf(&USB_str, "Obstacle Avoid:\rFL bumper\r\n");
		}

		//front right bumper, pin 1
		if (PORTD.IN & 0x02){
			//fprintf(&USB_str, "nothing\r\n");
		}
		else{
			_delay_ms(20);
			setMotorDuty(1, 700, MOTOR_DIR_BACKWARD_gc);
			setMotorDuty(3, 700, MOTOR_DIR_BACKWARD_gc);
			_delay_ms(500);
			setMotorDuty(1, 0, MOTOR_DIR_BACKWARD_gc);
			setMotorDuty(3, 0, MOTOR_DIR_BACKWARD_gc);
			fprintf(&USB_str, "Obstacle Avoid:\rFR bumper\r\n");
		}
```

```
                fprintf(&USB_str, "Obstacle Avoid:\rForward\r\n");
        }

                //grab data from buffer
                if (dataInBufE1())
                        fscanf(&uartE1_str,"%c",&position);
                //else position = 0;
                _delay_ms(20);

}// end while

//////////servo code//////////

//left: 50 deg, right: 130 deg, center: 90 deg
//up: 80 deg, down: 100 deg, center: 70 deg

//////////servo pan test code//////////
//
// int x;
//
// for (x=50; x<130; x++){
//     setServoAngle(x, 1); //pan servo
//     _delay_ms(50);
//     //setServoAngle(x, 2); //tilt servo
// }
//
// setMotorDuty(2, 700, MOTOR_DIR_BACKWARD_gc); //missile launcher
// _delay_ms(3700);
// setMotorDuty(2, 0, MOTOR_DIR_BACKWARD_gc);
//
// int y;
//
// for (y=130; y>50; y--){
//     setServoAngle(y, 1); //pan servo
//     _delay_ms(50);
//     //setServoAngle(y, 2); //tilt servo
// }
//
// setMotorDuty(2, 700, MOTOR_DIR_BACKWARD_gc); //missile launcher
// _delay_ms(3700);
// setMotorDuty(2, 0, MOTOR_DIR_BACKWARD_gc);

//////////Image processing routine//////////

//grab data from buffer
if (dataInBufE1()){
        fscanf(&uartE1_str,"%c",&position);
}
_delay_ms(20);

if (position > 0){
        setMotorDuty(1, 0, MOTOR_DIR_FORWARD_gc);
        setMotorDuty(3, 0, MOTOR_DIR_FORWARD_gc);
}

//target somewhere on the screen
while ( position > 0 && position < 255 ){
```

```
	//////////check sonar values/////////
	int lsonar0;
	int rsonar0;
	int lsonar1;
	int rsonar1;

	//check left sonar
	ADCA_request(1,1);
	_delay_ms(15);

	while(!ADCA_CH1_ConvComplete);
		lsonar0 = ADCA_getVal(1);
		_delay_ms(15);

	ADCA_request(1,1);
	_delay_ms(15);

	while(!ADCA_CH1_ConvComplete);
		lsonar1 = ADCA_getVal(1);
		_delay_ms(15);

	//check right sonar
	ADCA_request(2,2);
	_delay_ms(15);

	while(!ADCA_CH2_ConvComplete);
		rsonar0 = ADCA_getVal(2);
		_delay_ms(15);

	ADCA_request(2,2);
	_delay_ms(15);

	while(!ADCA_CH2_ConvComplete);
		rsonar1 = ADCA_getVal(2);
		_delay_ms(15);

	if (lsonar0 < 250 || lsonar1 < 250 || rsonar0 < 250 || rsonar1 < 250){
		position = 0;
	}

	//fprintf(&USB_str, "L: (%d, %d); R: (%d, %d)\r\n", lsonar0, lsonar1, rsonar0,
rsonar1);

	// left region: 0-85
	// center region: 86-170
	// right region: 171-254
	// dead center: 127

	int dx1 = (85-position)*0.6;
	int dx2 = (170-position)*0.6;
	double dx = abs(127-position);
	double dy = (lsonar0+rsonar0)/2;
	double theta = tan(dx/dy);
	int pan_angle = getServoAngle(1);
	int tilt_angle = getServoAngle(2);
	int j;

	setMotorDuty(1, 0, MOTOR_DIR_FORWARD_gc);
```

```
        setMotorDuty(3, 0, MOTOR_DIR_FORWARD_gc);

        //left region
        if ( position > 0 && position <= 85 ){
                fprintf(&USB_str,"Aligning:\rTurn Left: %d\r\n",position);
                setMotorDuty(1, 0, MOTOR_DIR_FORWARD_gc);
                setMotorDuty(3, 600+dx1, MOTOR_DIR_FORWARD_gc);
                _delay_ms(50);
                setMotorDuty(1, 0, MOTOR_DIR_FORWARD_gc);
                setMotorDuty(3, 0, MOTOR_DIR_FORWARD_gc);
                setServoAngle(50, 1);
        }

        //center region
        if ( position > 85 && position <= 170 ){
                fprintf(&USB_str,"Aligning:\rIn Center: %d\r\n",position);
                setMotorDuty(1, 0, MOTOR_DIR_FORWARD_gc);
                setMotorDuty(3, 0, MOTOR_DIR_FORWARD_gc);

                setServoAngle(90, 1);
                _delay_ms(50);
        }

        //right region
        if ( position > 170 && position < 255 ){
                fprintf(&USB_str,"Aligning:\rTurn Right: %d\r\n",position);
                setMotorDuty(1, 600+dx2, MOTOR_DIR_FORWARD_gc);
                setMotorDuty(3, 0, MOTOR_DIR_FORWARD_gc);
                _delay_ms(50);
                setMotorDuty(1, 0, MOTOR_DIR_FORWARD_gc);
                setMotorDuty(3, 0, MOTOR_DIR_FORWARD_gc);

                setServoAngle(130, 1);

        }

        //grab data from buffer again
        if (dataInBufE1()){
                fscanf(&uartE1_str,"%c",&position);
        }
        else{
                position = 0;
        }
        _delay_ms(20);

} //end while

//position within the crosshairs
while (position == 255){

        fprintf(&USB_str,"Targeting:\rIn Sight: &d\r\n", position);
        setServoAngle(90, 1);
        _delay_ms(100);
        setMotorDuty(2, 700, MOTOR_DIR_FORWARD_gc); //missile launcher
        _delay_ms(3000);
        setMotorDuty(2, 0, MOTOR_DIR_FORWARD_gc);

        //grab data from buffer again
```

```
        if (dataInBufE1()){
                fscanf(&uartE1_str,"%c",&position);
        }
        _delay_ms(20);
 }

} //end while(1)

} //end main loop
```

Visual Studio Code:

```
// ObjectTracking.cpp : Define the entry point for console app.
#include "stdio.h"
#include "stdlib.h"
#include "cvaux.h"
#include "cxcore.h"

#include "cv.h"
#include "highgui.h"
#include "BlobResult.h"
#include "math.h"
#include <vector>
#include "COMfunct.h"

// Get thresholded image in HSV format
IplImage* GetThresholdedImageHSV( IplImage* img){
    // Create an HSV format image from image passed
    IplImage* imgHSV = cvCreateImage(cvGetSize(img),
                                     8,
                                     3);

    cvCvtColor(img, imgHSV, CV_BGR2HSV);

    // Create binary thresholded image acc. to max/min HSV rang5es
    // For detecting blue gloves in "MOV.MPG - HSV mode
    IplImage* imgThresh = cvCreateImage(cvGetSize(img),
                                        8,
                                        1);

       //detect blue
    cvInRangeS(imgHSV,
               cvScalar(105, 105, 60),
               cvScalar(135, 255, 172),
               imgThresh);

    // Tidy up and return thresholded image
    cvReleaseImage(&imgHSV);
    return imgThresh;
}
int main()
{

       //data sent over Xbee serial is in hex values
       //processor can interpret either hex value or decimal values
       //display data over XCTU using USB port
       COM_Index INDEX;
```

```
        HANDLE COMPORT;
        COMPORT = OpenCom(5);

        int posX = 0;
        int posY = 0;

        CBlobResult blobs;
    CBlob *currentBlob;
    CvPoint pt1, pt2;
    CvRect cvRect;
    int key = 0;
    IplImage* frame = 0;

    // Initialize capturing live feed from video file or camera
    //CvCapture* capture = cvCaptureFromCAM(0);
        CvCapture* capture =
cvCreateFileCapture("http://192.168.1.103/videostream.cgi?user=phongt12&pwd=Hondaj30.&res
olution=8&rate=0&a=.mjpg");  // For IP cam

    // Get the frames per second
    int fps = (int)cvGetCaptureProperty(capture,
                                        CV_CAP_PROP_FPS);

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

    // Windows used to display input video with bounding rectangles
    // and the thresholded video
    cvNamedWindow("video");
    cvNamedWindow("thresh");

    // An infinite loop
    while(key != 'x')
    {
        // If we couldn't grab a frame... quit
        if(!(frame = cvQueryFrame(capture)))
            break;

        // Get object's thresholded image (blue = white, rest = black)
        IplImage* imgThresh = GetThresholdedImageHSV(frame);

        // Detect the white blobs from the black background
        blobs = CBlobResult(imgThresh, NULL, 0);

        // Exclude white blobs smaller than the given value (10)
        // The bigger the last parameter, the bigger the blobs need to be for inclusion
        blobs.Filter(blobs,
                     B_EXCLUDE,
                     CBlobGetArea(),
                     B_LESS,
                     50);

        // Attach a bounding rectangle for each blob discovered
        int num_blobs = blobs.GetNumBlobs();
              int *vec_x = new int[num_blobs];
```

```
      int *vec_y = new int[num_blobs];

      int x_values = 0;
      int y_values = 0;

for (int i = 0; i < num_blobs; i++){
    currentBlob = blobs.GetBlob(i);
    cvRect = currentBlob->GetBoundingBox();

    pt1.x = cvRect.x;
    pt1.y = cvRect.y;
    pt2.x = cvRect.x + cvRect.width;
    pt2.y = cvRect.y + cvRect.height;

             //find center of rectanges
             posX = (pt2.x + pt1.x)/2;
             posY = (pt2.y + pt1.y)/2;

             //store center position in a vector
             vec_x[i] = posX;
             vec_y[i] = posY;

             x_values = vec_x[i] + x_values;
             y_values = vec_y[i] + y_values;


    // Attach bounding rect to blob in orginal video input
    cvRectangle(frame,
                pt1,
                pt2,
                cvScalar(255, 255, 255, 0),
                1,
                8,
                0);
}

             int x_avg = 0;
             int y_avg = 0;

             if (num_blobs > 0){
                    x_avg = x_values/num_blobs*244/320;
                    y_avg = y_values/num_blobs*254/240;
             }
             else{
                    x_avg = 0;
                    y_avg = 0;
             }

             // Print it out for debugging purposes
             //printf("position (%d,%d)\n", posX, posY);

             char Xout = x_avg;
             char Yout = y_avg;

             //check if object is within crosshairs
             //x: dead center: 122, range: 107-137
             //y: dead center: 127, range: 112-142
             if ((x_avg > 107 && x_avg <= 137) && (y_avg > 112 && y_avg <= 142)){
```

```
                Xout = 255;
                WriteCom(COMPORT, &Xout, 1);//out to comport
        }

        //output pan angle
        WriteCom(COMPORT, &Xout, 1);//out to comport

        //output tilt angle
        if (y_avg > 0 && y_avg <= 25){
                Yout = 245;
                WriteCom(COMPORT, &Yout, 1);//out to comport
        }
        else if (y_avg > 25 && y_avg <= 49){
                Yout = 246;
                WriteCom(COMPORT, &Yout, 1);//out to comport
        }
        else if (y_avg > 49 && y_avg <= 73){
                Yout = 247;
                WriteCom(COMPORT, &Yout, 1);//out to comport
        }
        else if (y_avg > 73 && y_avg <= 98){
                Yout = 248;
                WriteCom(COMPORT, &Yout, 1);//out to comport
        }
        else if (y_avg > 98 && y_avg <= 122){
                Yout = 249;
                WriteCom(COMPORT, &Yout, 1);//out to comport
        }
        else if (y_avg > 122 && y_avg <= 147){
                Yout = 250;
                WriteCom(COMPORT, &Yout, 1);//out to comport
        }
        else if (y_avg > 147 && y_avg <= 171){
                Yout = 251;
                WriteCom(COMPORT, &Yout, 1);//out to comport
        }
        else if (y_avg > 171 && y_avg <= 195){
                Yout = 252;
                WriteCom(COMPORT, &Yout, 1);//out to comport
        }
        else if (y_avg > 195 && y_avg <= 220){
                Yout = 253;
                WriteCom(COMPORT, &Yout, 1);//out to comport
        }
        else if (y_avg > 220 && y_avg <= 244){
                Yout = 254;
                WriteCom(COMPORT, &Yout, 1);//out to comport
        }
        else{}

        //printf("position: %d\n", Xout);
        printf("position (%d,%d), Xout: %d\n", x_avg, y_avg, Xout);

// Add the black and white and original images
cvShowImage("thresh", imgThresh);
cvShowImage("video", frame);

// Optional - used to slow up the display of frames
```

```
        key = cvWaitKey(2000/640);

        // Prevent memory leaks by releasing thresholded image
        cvReleaseImage(&imgThresh);
    }

    // We're through with using camera.
    cvReleaseCapture(&capture);

    return 0;
}
```