Intelligent Machine Design Lab

EML 5666

Final Report

Luther Lloyd III

Mechanical Engineer

Table of Contents

1. Abstract 3

2. Executive Summary 3

3. Introduction 4

4. Integrated System 5

5. Mobile Platform 6

6. Actuation 7

7. Sensors 8

8. Behaviors 9

9. Experimental Layout and Results 12

10. Conclusion 13

11. Documentation 13

12. Appendices
   - A-1: Drive Sub-functions 14
   - A-2: Obstacle Avoidance Functions 16
   - A-3: Search Function main call 18
   - A-4: Acquire Function main 19

1. Abstract

The purpose of the following report is to describe the design challenge of building an intelligent machine and then testing it to validate the design. The intelligent machine described below is a search and acquire robot. Relying on random turn sequences the robot will search the room pausing often to use the vision system to “look” for certain objects and then acquire them based on the position of the object within the screenshot. Once the robot is in the correct position to acquire the object a lifting mechanism will place it into the storage container. As an intelligent machine it will be capable of obstacle avoidance while searching such that it will be able to preemptively avoid walls in its immediate path and avoid running through objects.

2. Executive Summary

After spending the semester working on this project I have learned a lot more about intelligent machines that I thought I was going to. With a greater mechanical knowledge of machines than electrical, there were many learning opportunities available with a search and acquire robot. The plan was to create a robot which would be a fun project and something I could use after it was completed. To get the most out of this project I decided to leverage my mechanical design and manufacturing skill to allow for a more aggressive electrical control setup including wireless communication between the robot, camera and image processor. The steep learning curve with programming the robot and developing the image processing was much steeper than I had anticipated. This made the project schedule tight; though working diligently I was able to overcome it while still learning a lot.

As a teaching assistant for the mechanical engineering student shop I was able to use the machines to fabricate the single piece chassis and machine all of the mounts. The aluminum chassis is made from right angle extrusions welded together, providing a sturdy platform with great form factor and reliability. Sheet metal was utilized to mount the sensors and buttons because of its ease of manufacturing and simplicity of assembly. The mounts for the gear-motors were machined from billet to hold the motors and front obstacle avoidance sensors to the chassis.

The Epiphany DIY control board was simple to program using the Atmel Studio environment and the interface subroutines written by Tim Martin. The biggest benefit of using this control board is that it had all of the inputs, outputs, motor and servo drivers built in reducing the its footprint on the chassis. This all-in-one control board also made it much easier to debug, allowing me to focus on programming the behaviors. Programming the obstacle avoidance and acquire subroutines went smoothly. However, when piecing all of the subroutines together I ran into issues where the robot did not perform as expected. Tweaking of the code and function continued up till the end of the project to get it working as desired.

The image processing work took the most effort to get running, though since it was completed it has worked stunningly. The difficult aspect of the image processing was deciding how the robot would “see” the wrench and then what information the robot would need to proceed. OpenCV had many useful libraries to choose from when determining how to find the wrench. I went with color detection using an HSV mask; this worked very well with the wrenches painted orange and having a blue floor. Using the rotated rectangle bounding box function the position and orientation of the wrench was very easy to calculate. The calculations had to be sent over a wireless serial connection (Xbee radio) which required converting the calculated numbers from type double to type char. At first I was worried about the potential loss of integrity of the data though that turned out to be a non-issue after testing.

Although the class is over and the robot mostly working; I plan to continue developing this as prototype to make it more reliable. I would like to develop it further and potentially market the concept to iRobot as a garage cleaning device. This is the end goal which is easily a few months or more away from being a reality.

3. Introduction

As a mechanical engineer I enjoy working on my car and motorcycle and I don’t enjoy having to pick up the wrenches all the time. To avoid this I am creating Wrench Dog, a mechanic’s best friend. Wrench Dog will search for wrenches on the floor, identify them and then pick them up. Since there will likely be other items on the floor Wrench Dog will need to be able to sense these random objects and avoid them while “sniffing” out the wrenches. Wrench Dog will also need to be durable and robust to be able to consistently pick up wrenches everyday work is being done to a car or motorcycle. The main goal in conjunction with picking up the wrenches is to provide a learning environment for vision and electrical controls while building on my previous mechanical design experience. Gaining experience in dealing with vision and electrical controls will aid me in my future job as high-speed manufacturing equipment designer.

4. Integrated System

The intelligent machine 'Wrench Dog' utilizes the Epiphany DIY board for "intelligence" and functional control. Sensors used for object avoidance, IR range and bump/feeler switches are fed directly to the Epiphany DIY for quick reaction in adjusting the robots direction. An IP camera will be integrated to the control board by first having the raw video transmitted from the camera to a laptop over WiFi for processing and then sending the resultant command signals to the Epiphany DIY control board via RF with XBee radios. The video processing is done on an external laptop due to the processing power required for color detection and potentially pattern recognition. The choice to use an external video processor instead of using a more advanced processor for control board was due to my insufficient experience in dealing with such hardware/software interactions. Utilizing the Epiphany DIY board allows simpler programming for motion control allowing more focus to be spent learning the vision system controls. The Epiphany DIY board is an integrated system by itself incorporating motor and servo drivers along with XBee communications port and a large I/O selection making it a simple package that nestles nicely within the robot frame. There are three major components to the function of the robot: Search, Acquire and Avoid. The Search and Acquire functions are interconnected through the drive functions. The drive functions control the motion of the robot while checking the four bump sensors. If a bump sensor is hit the drive function is ended and Avoid function determines the avoidance maneuver required and then calls on the appropriate drive function. Once the avoidance maneuver is over the Search function regains control of the drive functions. While in Search the robot will pause between drive commands to analyze the image taken by the IP camera. If a wrench is in the image the Acquire function takes over using the IP camera images and drive functions to move the robot to the wrench so the magnet can grab it and it can then be dropped into the storage bin.

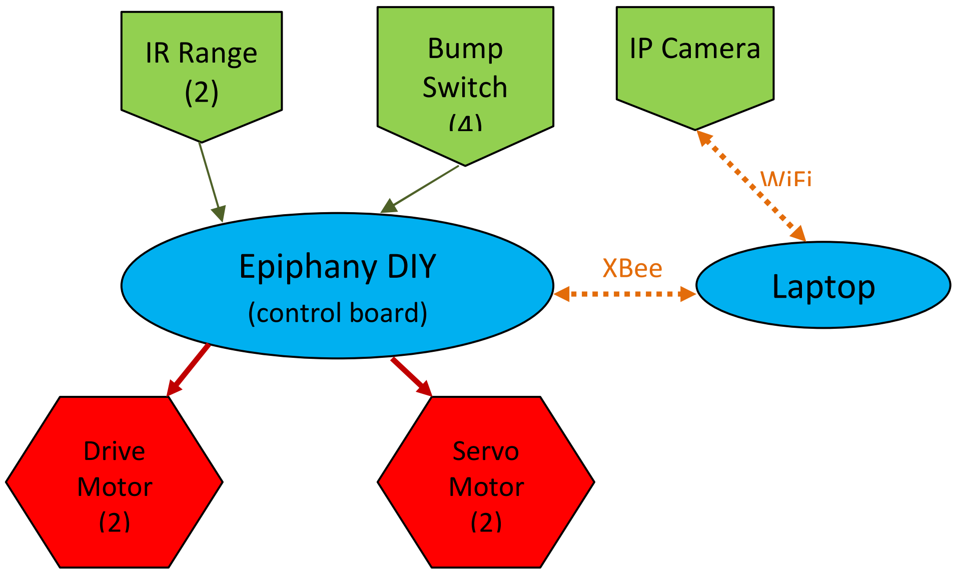


Figure 4.1 Block diagram of controls integration between inputs, controllers and outputs.

5. Mobile Platform

The mobile platform is used to hold all of the sensors, actuators, motors and controller in an appropriate size to allow the full functionality of the design. The base of the mobile platform is manufactured from various pieces of aluminum angle extrusions due to their strength to weight ratio and ease of machining and welding. The main piece is a 2”x2”x3/16” aluminum angle which will be the bottom of the storage bin, mounting point of the drive motors, front bump and IR sensors, high torque servo and lifting arm. The purpose of mounting all of these components to the angle is to reduce the weight by using a single piece for multiple tasks. Light weight is goal because the robot is battery operated; the less power needed to move the robot around the longer the robot can go between charges. The control board and rear caster will be held by smaller pieces of angle since less strength is needed for that part of the frame. The platform will be driven via a 25mm diameter 75:1 gearmotors with 56mm rubber wheels directly mounted the output shaft via custom machined hubs. This allows for differential steering which has the advantage of being able to turn the robot within its footprint. To avoid shorting the control board it will be electrically isolated via plastic mounting and covers. The single piece construction protects the important electronics and allows many mounting points to be made in the frame.

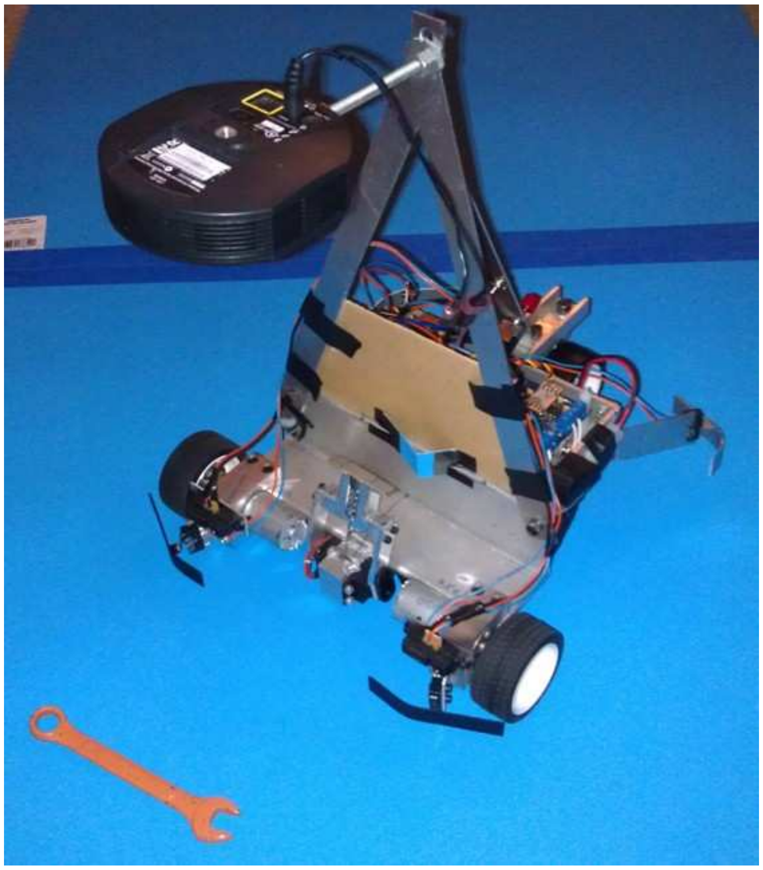
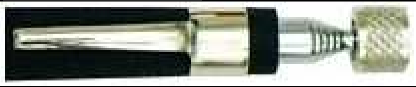


Figure 5.1 Picture of Wrench Dog, Highlighting aluminum extrusion as chassis and multiple task mounting brackets. All screws are easily accessible allowing for quick serviceability.

6. Actuation

Wrenches weigh around 0.25 lbs so a high torque servo (1501MG) was selected to allow for a 2.75” long lifting arm. The large factor of safety in lifting force allows for adjustments to the arm and mechanism without needing the change the servo. The gearmotors are small though provide plenty of torque through the 75:1 gear reduction. With these motors mounted to the 56mm tires the robot will be capable of a speed of around 8 in/sec. After testing it was found that this speed was much greater than desired causing jerky and inconsistent motions. Choosing a motor that would produce a maximum speed of around 1 in/sec would have been a better choice. Because the motors are too fast for what I need I have to run them at the lowest PWM signal that still moves the robot limiting the speed tuning that can be done. A magnet is utilized on the end of the lifting arm to “grab” the wrench. The high torque servo then rotates the arm upward to the release position. Once in the release position the secondary servo pulls a string connected to the magnet to reduce the magnet field at the tip of the arm enough to drop the wrench into the storage area. Unfortunately while lifting the wrench the single magnet does not provide enough strength to prevent the wrench from twisting. This requires more precise alignment to actually pick up the wrench where the use of a pair of magnets would have made it easier to acquire the wrenches.

Table 6.1 Actuator List

Picture	Name	Description
	75:1 Metal Gearmotor 25Dx54L mm	Gearmotor used to drive the robot. Two motors total each directly tied to the tires.
	HiTec HS-645MG	This medium torque servo is used to pull the release wire of the magnet. This wire pulls the magnet away from the arm end release the wrench
	Power HD High-Torque Servo 1501MG	This high torque (240 oz-in) servo is used to lift the shovel. With the 4-bar link used to move the shovel this servo can lift nearly 4lbs at the shovel end.
	Tamiya 70111 Sports Tire Set	Cool race looking tires mounted directly to drive motors for looks and performance.
	Magnetic Pick-Up Tool	Used to pick up the wrench. Only the top of the pick-up tool was needed

7. Sensors

Sensors are an important part of this robots functionality. The combination of infra-red (IR) range finders at the front of the robot and bump switches at each corner allow for robust obstacle avoidance. The IR sensors allow for the robot to avoid walls and objects over 2 in tall before the robot contacts them. If either of the two IR sensors detects a wall that is closer than five inches it will turn instead of drive forward. The bump switches at each corner are for avoiding walls and other objects that the robot may come in contact with. With low contact force required for detection and a 100 Hz sampling rate, even the slightest of bumps are detected and avoided. The IP camera was chosen to provide an opportunity for advanced learning since there are many uses for vision systems within the manufacturing industry which I will be working in. The IP camera is utilized to take a snapshot of the ground and send it to the laptop to determine if a wrench is in view. To find the wrenches OpenCV 2.3.1 will be employed for color detection and tracking. For reliability on multiple surfaces and different lighting scenarios the wrenches will be painted in a high visibility matte orange. Utilizing example code for HSV color detection from Josh Weaver I was able to create a robust program which calculates the orientation of the wrench, distance from wrench center to robot and the relative angle between wrench center and robot heading. Originally this data was going to be used for open-loop motions to acquire the wrenches. This scheme did not function reliably so a modification of acquire function will be implemented to use the same information and more screenshots to create closed-loop motion to increase the robustness of the actuation.

Table 7.1 Sensor List

Picture	Name	Description
	IP Camera, WCV80N	Wireless camera used to look for and identify wrenches to be picked up via HSV color detection
	Sharp GP2Y0A21YK0F Analog Distance Sensor 10-80cm	IR range finder used for long range object avoidance
	Snap-Action Switch with 16.7mm Lever: 3-Pin, SPDT, 5A	Bump switch at all four corners to check for undesired environment contact

8. Behaviors

As mentioned in the Integrated System section, there are three major components to the function of the robot: Search, Acquire and Avoid. These behaviors are separate in nature but interconnected in operation to allow seamless transitions between the functions. Each of these functions make use of basic sub-functions for operation to reduce code complexity and simplify debugging.

Motion Sub-functions

Four motor driving sub-functions; driveForward, driveBackward, turnLeft, turnRight have either a time input or angle input and return the state of the digital inputs of the bump switches. If the bump switches are hit these functions exit returning the “bump” state. The code for these functions can be found in Appendix A-1.

Search

The search function works by choosing a random number (between 3-10) forward drives. The distance of the forward drive is approximately 1-2 inches. After this series of forward drives a random turn from 0-360^0 will occur. After each drive forward and turn the robot will pause to scan the camera image for a wrench. If no wrench is found this group will repeat, otherwise the Acquire function will be called. After the Acquire function completes the Search function will resume.

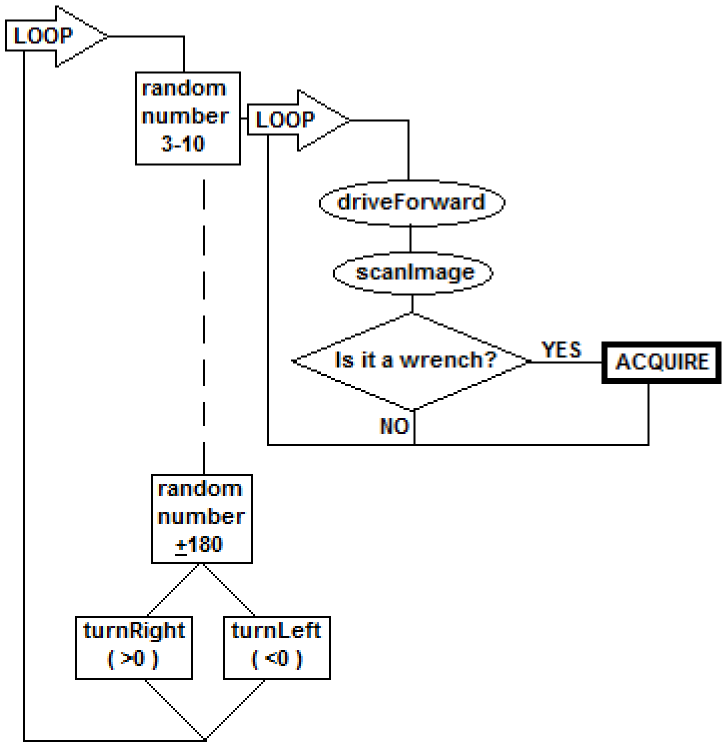


Figure 8.1 Search function block diagram

Acquire

The purpose of the Acquire function is to group the commands necessary to detecting, tracking and picking up a wrench. The detection program will be running in Visual Studio/OpenCV on a remote laptop and use HSV color detection. The detection will be done by comparing the image against a known range of HSV values for the wrench color. With the mask image created (white pixel for in HSV range, black dot for outside, Figure 8.4) the rotated rectangle bounding box function will be used to determine distance from robot and orientation of the wrench (Figure 8.3). These values will be read from the Xbee radio serial comm port. If the wrench is farther away than an inch from the arm the robot will drive forward a small amount and then request a new image scan. This process repeats until the wrench is close to arm. At this point the arm is lowered to the pick-up position, the robot drives forward into the wrench and then raises the arm and wrench to the drop location; releasing the wrench completes the Acquire function. The small movements and multiple image captures create a "visual servoing" to help reduce the error between the commanded motion of the robot and its actual motion.

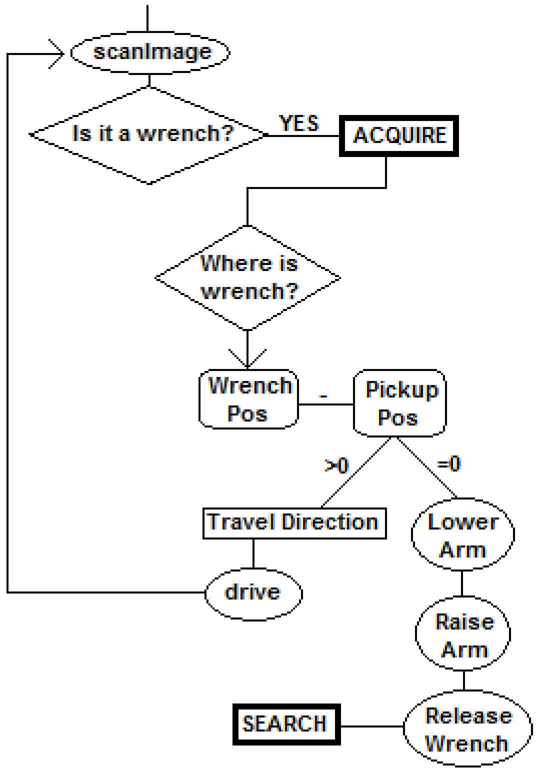


Figure 8.2 Acquire function block diagram

Avoid

The Avoid function is used to determine which sensors were hit or if there is an object directly in front of the robot. Bump detection is running every 10 milliseconds, if a bump is detected then a sub-routine is used to determine which senor or sensors were hit to determine which direction to move in to avoid the wall, table 8.1 details out these reactions. Each avoidance maneuver has some random turn angle associated with it to reduce the likelihood it will get stuck in a corner.

The bump detection continues to run while in an avoidance maneuver. The IR sensors are apart of the avoid function group though are only checked as the robot moves forward since they can only detect if an object is in front of the robot.

Table 8.1 Bump switch avoidance table

Bump Switches Hit	**Avoidance Maneuver**
Front Left	Drive Backward 0.25sec, Turn Right random angle
Front Right	Drive Backward 0.25sec, Turn Left random angle
Rear Left	Drive Forward 0.25sec, Turn Right random angle
Rear Right	Drive Forward 0.25sec, Turn Left random angle
Both Front	Drive Backward 0.25sec, Random Turn (angle, direction)
Both Rear	Drive Forward 0.25sec, Random Turn (angle, direction)

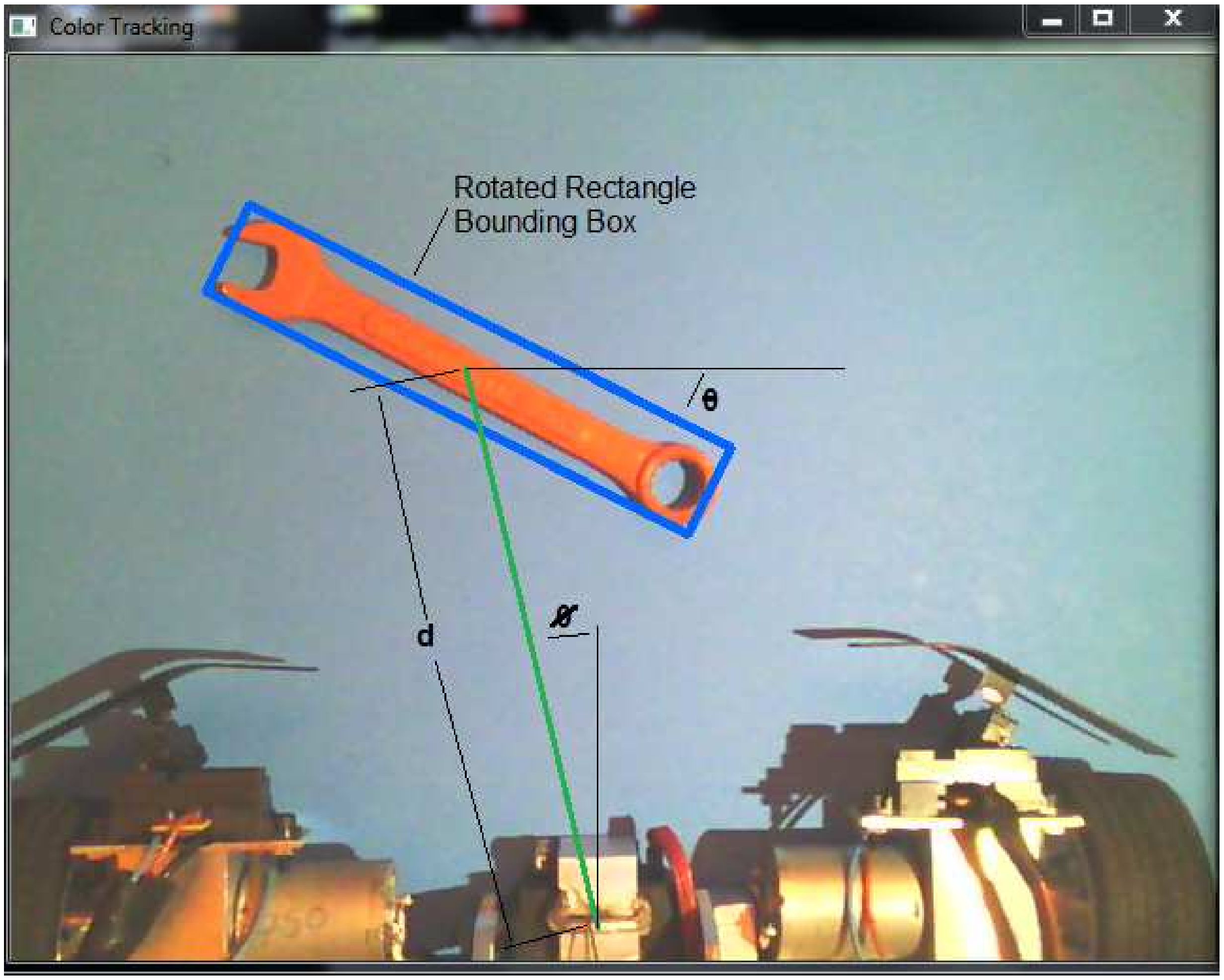
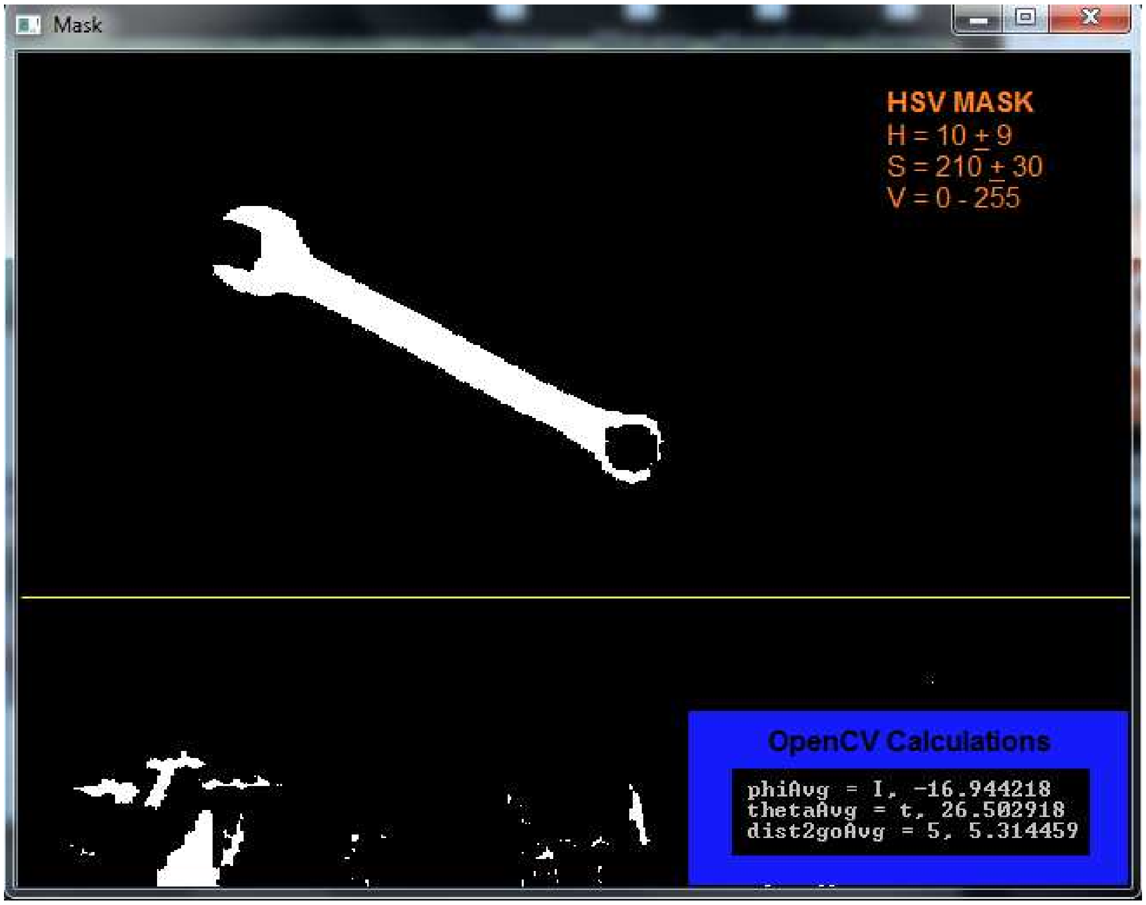


Figure 8.3 Color Image processed with rotated bounding box



```
phiAvg = I, -16.944218
thetaAvg = t, 26.502918
dist2goAvg = 5, 5.314459
```

Figure 8.4 Mask Image showcasing how the HSV difference for wrench vs background

9. Experimental Layout and Results

The robot has been fabricated and the sub-routines tested. The IR sensors were calibrated using a measuring tape to obtain the distance between the IR sensor and a notebook to compare to the analog value. The measurements and values are in Table 9.1. The obstacle avoidance program was written and tested independently of the rest of the program. This was tested by having the robot run in a loop of random movements while in a room and periodically hitting the various switches multiple times in order to confirm code robustness. The robot responds quickly and with the random turn angles does a good job of keeping off of walls and out of corners.

Table 9.1 IR Range Calibration

Wall Distance	Analog Value
< 3”	4095
5”	3200
7”	2100
9”	1700
> 11”	1100

With the camera and a sample color tracking code paint colors and finishes has been tested for their repeatable detection against various backgrounds and lighting conditions; the chosen color was a high visibility matte orange on a blue background. Once the image processing program was working I tested the acquire sub-routine to make sure the communications between the robot and laptop. When testing the sub-routine with a wrench in the screenshot the robot reacted well and accurately more than 80% of the time. Unfortunately when running this sub-routine with the search function in the arena the robot was not able to achieve these same results in testing or in demo day. It was determined that this issue was caused by using open-loop or dead reckoning to control the robot; which, when multiple motions were required compounded the error with the motor control. To prevent this I have re-worked the acquire sub-routine to move the robot in small increments and take more screenshots to reduce the error compounding.

10. Conclusion

The robot design and fabrication took longer than anticipated due to a few design changes that had to be made while fabricating. The changes were mostly related to serviceability or material available when it was being built. Since being fabricated the major focus has been on creating the program code to control the robot. This also seemed to take longer than I thought it would though I had tried to plan for this since I have less programming experience than I have mechanical design and fabrication experience. I had the most to learn from working on the vision system and feel I gained some useful experience with how it can be used and what methods are available. Testing the robot revealed many ideas I had not thought about before working on this project such as planning the program when purchasing hardware instead of just programming around the variability of hardware. I have gained a lot of useful experience from this project; to think of projects from a combined mindset of both an electrical and mechanical engineer. As this project wraps up it should be something that is very useful for my garage project.

11. Documentation

Epiphany DIY control board: http://ootbrobotics.pixelgeko.com/

Hardware Integration Software for ATXMega – Tim Martin

Motors, Tires, Sensors from Pololu: http://www.pololu.com/

Linksys IP Camera: http://homestore.cisco.com/en-us/cameras/linksys-WVC80N_stcVVproductId84737621VVviewprod.htm

HSV Sample Program (used for basis of OpenCV program) – Josh Weaver

12. Appendices

A-1: Drive Sub-functions

```
// Motor 2 = Right Motor      Motor 4 = Left Motor
// travel speed ~ 2.24" / 1 sec
// 1 wheel turn 14.01* / 1 sec, 2 wheel turn 28.97* / 1 sec

double turnTime;
int noBump = 0x0F, i;
int mtr2spd = 560, mtr4spd = 570; //minimum PMW for motion
uint8_t bumpDetected = 0x0F;

uint8_t driveForward(double driveTime){
       i=0;
       bumpDetected = 0x0F;
       while ((bumpDetected == noBump) && (i <= driveTime)){
              setMotorEffort(4,mtr4spd,MOTOR_DIR_FORWARD);
              setMotorEffort(2,mtr2spd,MOTOR_DIR_FORWARD);
              _delay_ms(10);
              i+=10;
              bumpDetected = PORTD.IN;
       }
       setMotorEffort(4,0,MOTOR_DIR_NEUTRAL);
       setMotorEffort(2,0,MOTOR_DIR_NEUTRAL);
       return bumpDetected;
}

uint8_t driveBackward(double driveTime){
       i=0;
       bumpDetected = 0x0F;
       while ((bumpDetected == noBump) && (i <= driveTime)){
              setMotorEffort(4,mtr4spd,MOTOR_DIR_BACKWARD);
              setMotorEffort(2,mtr2spd,MOTOR_DIR_BACKWARD);
              _delay_ms(10);
              i+=10;
              bumpDetected = PORTD.IN;
       }
       setMotorEffort(4,0,MOTOR_DIR_NEUTRAL);
       setMotorEffort(2,0,MOTOR_DIR_NEUTRAL);
       return bumpDetected;
}

uint8_t turnRight(uint8_t turnMode, uint16_t turnAngle){
       bumpDetected = 0x0F;
       i=0;
       switch(turnMode){
              case(1):      // turnMode = 1 : Right Wheel Forward
                     turnTime = (turnAngle / 14.01) * 1000;
                     while ((bumpDetected == noBump) && (i <= turnTime)){
                            setMotorEffort(2,mtr2spd,MOTOR_DIR_FORWARD);
                            _delay_ms(10);
                            i+=10;
                            bumpDetected = PORTD.IN;
                     }
                     setMotorEffort(2,0,MOTOR_DIR_NEUTRAL);
              break;
```

```
		case(2):	// turnMode = 2 : Left Wheel Backward
			turnTime = (turnAngle / 14.01) * 1000;
			while ((bumpDetected == noBump) && (i <= turnTime)){
				setMotorEffort(4,mtr4spd,MOTOR_DIR_BACKWARD);
				_delay_ms(10);
				i+=10;
				bumpDetected = PORTD.IN;
			}
			setMotorEffort(4,0,MOTOR_DIR_NEUTRAL);
		break;

		case(3):	// turnMode = 3 : Right Wheel Forward and Left Wheel Backward
			turnTime = ((double)turnAngle / 28.97) * 1000;
			while ((bumpDetected == noBump) && (i <= turnTime)){
				setMotorEffort(2,mtr2spd,MOTOR_DIR_FORWARD);
				setMotorEffort(4,mtr4spd,MOTOR_DIR_BACKWARD);
				_delay_ms(10);
				i+=10;
				bumpDetected = PORTD.IN;
			}
			setMotorEffort(2,0,MOTOR_DIR_NEUTRAL);
			setMotorEffort(4,0,MOTOR_DIR_NEUTRAL);
		break;
	}
	return bumpDetected;
}

uint8_t turnLeft(uint8_t turnMode, uint16_t turnAngle){
	bumpDetected = 0x0F;
	i=0;
	switch(turnMode){
		case(1):	// turnMode = 1 : Left Wheel Forward
			turnTime = (turnAngle / 14.01) * 1000;
			while ((bumpDetected == noBump) && (i <= turnTime)){
				setMotorEffort(4,mtr4spd,MOTOR_DIR_FORWARD);
				_delay_ms(10);
				i+=10;
				bumpDetected = PORTD.IN;
			}
			setMotorEffort(4,0,MOTOR_DIR_NEUTRAL);
		break;

		case(2):	// turnMode = 2 : Right Wheel Backward
			turnTime = (turnAngle / 14.01) * 1000;
			while ((bumpDetected == noBump) && (i <= turnTime)){
				setMotorEffort(2,mtr2spd,MOTOR_DIR_BACKWARD);
				_delay_ms(10);
				i+=10;
				bumpDetected = PORTD.IN;
			}
			setMotorEffort(2,0,MOTOR_DIR_NEUTRAL);
		break;

		case(3):	// turnMode = 3 : Left Wheel Forward and Right Wheel Backward
			turnTime = ((double)turnAngle / 28.97) * 1000;
			while ((bumpDetected == noBump) && (i <= turnTime)){
				setMotorEffort(4,mtr4spd,MOTOR_DIR_FORWARD);
```

```
                        setMotorEffort(2,mtr2spd,MOTOR_DIR_BACKWARD);
                        _delay_ms(10);
                        i+=10;
                        bumpDetected = PORTD.IN;
                    }
                    setMotorEffort(4,0,MOTOR_DIR_NEUTRAL);
                    setMotorEffort(2,0,MOTOR_DIR_NEUTRAL);
                break;
        }
        return bumpDetected;
}

uint8_t turnLorR(uint8_t turnMode, int turnAngle){
        //purpose is to use one function to decided between left or right functions
        // (-) => Left   (+) => Right
        if(turnAngle < 0){bumpDetected = turnLeft(turnMode,-turnAngle);}
        else{bumpDetected = turnRight(turnMode,turnAngle);}
        return bumpDetected;
}
```

A-2: Obstacle Avoidance Functions

```
uint8_t bumpCheckDelay(double delayTime){
        int i=0;
        bumpDetected = 0x0F;
        while ((bumpDetected == 0x0F) && (i <= delayTime)){
                _delay_ms(10);
                i+=10;
                bumpDetected = PORTD.IN;
        }
        return bumpDetected;
}


void whichBump(uint8_t bumpDetected){
//case statement to determine which sensor combination was hit
//calls appropriate drive function to get away from it
        switch(bumpDetected){
                case(0x0E):   //Front Left, pin 1 closed, mask = 0000 1110
                        bumpHit = driveBackward(250);
                        if (bumpHit == noBumpObs){
                                turnAngle = rand() % 180;
                                bumpHit = turnRight(2,turnAngle);
                                if (bumpHit != noBumpObs) { whichBump(bumpHit); }
                        }else { whichBump(bumpHit); }
                break;

                case(0x0D):   //Front Right, pin 2 closed, mask = 0000 1101
                        bumpHit = driveBackward(250);
                        if (bumpHit == noBumpObs){
                                turnAngle = rand() % 180;
                                bumpHit = turnLeft(2,turnAngle);
                                if (bumpHit != noBumpObs) { whichBump(bumpHit); }
                        }else { whichBump(bumpHit); }
                break;
```

```
            case(0x0B):   //Rear Left, pin 3 closed, mask = 0000 1011
                   bumpHit = driveForward(250);
                   if (bumpHit == noBumpObs){
                          turnAngle = rand() % 180;
                          bumpHit = turnRight(2,turnAngle);
                          if (bumpHit != noBumpObs) { whichBump(bumpHit); }
                   }else { whichBump(bumpHit); }
            break;

            case(0x07):   //Rear Right, pin 4 closed, mask = 0000 0111
                   bumpHit = driveForward(250);
                   if (bumpHit == noBumpObs){
                          turnAngle = rand() % 180;
                          bumpHit = turnLeft(2,turnAngle);
                          if (bumpHit != noBumpObs) { whichBump(bumpHit); }
                   }else { whichBump(bumpHit); }

            break;

            case(0x0C):   //Front, pin 1 & 2  closed, mask = 0000 1100
                          bumpHit = driveBackward(250);
                          if (bumpHit == noBumpObs){
                                 turnAngle = rand() % 360 - 180;
                                 bumpHit = turnLorR(2,turnAngle);
                                 if (bumpHit != noBumpObs) { whichBump(bumpHit); }
                          }else { whichBump(bumpHit); }
            break;

            case(0x03):   //Back, pin 3 & 4  closed,  mask = 0000 0011
                          bumpHit = driveForward(250);
                          if (bumpHit == noBumpObs){
                                 turnAngle = rand() % 360 - 180;
                                 bumpHit = turnLorR(2,turnAngle);
                                 if (bumpHit != noBumpObs) { whichBump(bumpHit); }
                          }else { whichBump(bumpHit); }
            break;
       }
}

int objAhead(){
//reads IR Sensors to avoid objects, returns 1 if object is there, 0 otherwise
       int leftIR = 0, rightIR = 0;
       adcChannelMux(&ADCA,1,0);
       adcChannelMux(&ADCA,2,2);
       _delay_ms(50);
       for(int ii=0; ii<2; ii++){
              leftIR += analogRead(&ADCA,1);
              rightIR += analogRead(&ADCA,2);
              _delay_ms(100); //read delay
       }
       leftIR /= 2;
       rightIR /= 2;

       if ((leftIR >= 2100)||(rightIR >= 2100)){ //bigger number = closer
              return 1;
       }else{return 0;}
}
```

A-3: Search Function main

```
void searchDrive(){
	int numDrive, turnAng, i=0, start = 0, wall =0;

	callCamera();
	int exit = 0;

	while (exit == 0){   //run switch break
		numDrive = rand() % 8 + 3; //3-10
		isWrench = 0;
		i = 0;
		//straight drives
		while ((isWrench == 0)&&(i < numDrive )){
			i++;
			wall = objAhead();
			if ((wall == 0)&&(PORTF.IN && 0x10)){
				bumpDet = driveFwSlow(drvTfrw);
				if (bumpDet == noBumpSrch){
					if(PORTF.IN && 0x10){bumpDet1 = bumpCheckDelay(3000);}
//wait for camera image to stabilize
					if ((bumpDet1 == noBumpSrch)&&(PORTF.IN && 0x10)){
						if(PORTF.IN && 0x10){callCamera();}
					}else{whichBump(bumpDet1);}

				}else{whichBump(bumpDet);}
			}else{break;}
			if(PORTF.IN&& 0x10){break;}
		}

		//turn
		bumpDet = 0x0F;
		if ((objAhead())&&(PORTF.IN && 0x10)){bumpDet = driveBackward(800);} //if
in front of wall back up before turn
		if (bumpDet == noBumpSrch){
			if(PORTF.IN && 0x10){turnAng = rand() % 360 - 180;
			bumpDet1 = turnLorR(3,turnAng);}
			if (bumpDet1 == noBumpSrch){
				if(PORTF.IN && 0x10){bumpDet2 = bumpCheckDelay(3000);} //wait
for camera image to stabilize
				if (bumpDet2 == noBumpSrch){
					if(PORTF.IN && 0x10){callCamera();}
				}else{whichBump(bumpDet2);}
			}else{whichBump(bumpDet1);}
		}else{whichBump(bumpDet);}

		if(!(PORTF.IN && 0x10)){exit = 1;} // run switch off
	}
}


void callCamera(){//call for image processing data
	int ii;
	attempts = 0;
	while (attempts < 3){
		ph =0; th =0; di =0; ii = 0;
		attempts++;
		fprintf(&Xbee_str,"t"); //send command to OpenCV
```

```
            while(!(dataInBufE1())){_delay_ms(10);} // wait for image processing data
            //read image processing data
            while(dataInBufE1()){
                  fscanf(&Xbee_str,"%c",&readChar);
                  tempChar[ii] = readChar;
                  ii++;
            }
            if (ii == 3){
                  //convert image processing data
                  ph = (int) tempChar[0];
                  th = (int) tempChar[1];
                  di = (int) tempChar[2];
                  if (((ph != 0)&&(th != 0))&&(di != 0)){
                        acquireDrive(ph-89,th-89,di);
                  }
                  attempts = 4; // exit loop
            }
      }
}
```

A-4: Acquire Function main

```
void acquireDrive(int phi, int theta, int dist2go){
      //-phi = left of center
      //+theta = cw from horizontal
      //returns 1 for good grab and 0 for no grab
      int theta2 = theta-phi; //phi-theta sign conv follows theta sign
      //no realignment needed
      if(abs(theta2) <= 30){grabWrench(phi,theta,dist2go);}

      else{  //wrench at bad angle, attempt to re-align then find wrench again
            bumpDet1 = driveBackward(700);
            if(bumpDet1 == noBumps){
                  angle = 45*(-theta2/abs(theta2));
                  bumpDet2 = turnLorR(3,angle);
                  _delay_ms(250);
                  if(bumpDet2 == noBumps){
                        bumpDet3 = driveForward(700)
                        _delay_ms(250);
                        if(bumpDet3 == noBumps){
                              angle = 90*(theta2/abs(theta2));
                              bumpDet4 = turnLorR(1,angle);
                              _delay_ms(250);
                              if(bumpDet4 == noBumps){
                                    callCamera(); //call camera again
                              }else{whichBump(bumpDet4);}
                        }else{whichBump(bumpDet3);}
                  }else{whichBump(bumpDet2);}
            }else{whichBump(bumpDet1);}
      }
}
```

```
void grabWrench(int phi, int theta, int dist2go){ //subroutine to execute motions to pick
up wrench
	//which is wrenchDist away straight ahead
	int pickupPos = 167, dropPos = 30;
	int addTime = 0;
	if(wrenchDist > 4.0){
		bumpDet = turnLorR(3,phi);
		if(bumpDet == noBumps){
			bumpDet1 = driveForward(450);
			if (bumpDet1 == noBumps){callCamera();}
			else{whichBump(bumpDet1);}
		}else{whichBump(bumpDet);}
	}else{
		setServoAngle(1,pickupPos);
		_delay_ms(750); // make sure arm is all the way down
		bumpDet = driveFwSlow(500);
		servoPickup(pickupPos,dropPos);
		servoRelease();
		if(bumpDet != noBumps){whichBump(bumpDet);}
	}
}

void servoPickup(int lowPos, int upPos){ //subroutine to raise arm at a slow speed
	for(int i=lowPos;i >=upPos;i-=1){
		setServoAngle(1,i);
		_delay_ms(25);
	}
}

void servoRelease(){ //subroutine to release wrench
	setServoAngle(2,130);
	_delay_ms(500);
	setServoAngle(2,60); //reset release
}
```