

# Final Report

## **Haro**

Reinier Vladimir Santos

University of Florida  
Department of Electrical and Computer Engineering  
**EEL5666 Intelligent Machine Design Laboratory**  
Dr. A. Antonio Arroyo  
Dr. Eric M. Schwartz

Mike Pridgen  
Thomas Vermeer

## Table of Contents

Abstract	3
Introduction	3
Executive Summary	4
Integrated System	5
Software Flowchart	6
Hardware Block Diagram	7
Mobile Platform	8
Actuators	9
Sensors	10
Behaviors	11
Experimental Layout and Results	12
Conclusion	14
Appendix	15

## **Abstract**

The project involves the conceptualization, experimentation and design of an autonomous mobile robot within the constrained amount of time and resources. The robot will be called Haro and will contain intelligent behaviors akin to that of a reconnaissance companion. Haro is a replication of a fictional robot bearing the same name and similar behavioral aspects. The robot will be designed as to be as much as identical to the original predecessor in both physical and intellectual form, which sets the personal completion goal of the engineer.

## **Introduction**

During times of war, various events can occur in the life of a soldier. One of these events is when a soldier has to find his way back to his base. Another would be that when a soldier is ordered to do a reconnaissance mission to gather intelligence. An application of robotics would be to aid in these kinds of situations.

The original Haro is a fictional robot from the Japanese Animation Series called Gundam 00. Haro serves as a personal companion to one of the main characters and most of what he does is involved with assisting his owner in his missions. This is the basis to the design of my robot.

Inspired by its predecessor, Haro will serve as a personal companion of the owner on tactical events such as reconnaissance and withdrawal. The behaviors of the robot will revolve around these main points.

This paper will describe the design of my robot including the system diagram, choice of sensors and actuators, and detailed descriptions of proposed behaviors.

## **Executive Summary**

Haro is a robot that can aid in reconnaissance missions by applying wall following and line following algorithms. Basically, he can sneak and peak behind a corner by precisely following a wall and lead the user to a predefined path by following a line track.

Haro is able to accomplish these objectives using an 8-bit AVR processor called AT90USB646. The entire robot itself is in the shape of a sphere. The sensors used are a pair of Sharp Infrared Short Rangefinder on each side for wall following, a Digital Reflectance Sensor Array for line following, a Maxbotix LV-EZ1 sonar for general obstacle avoidance and inward turning for wall following, and a pair of wireless sensor gloves for manual control and command.

Haro is programmed using complex mathematics and algorithms in order to precisely control each actuation that needs to be performed in order to accomplish specific tasks. The software involves PID algorithms for both the wall following and line following, yaw compensation using Cartesian to Polar coordinate conversion for wall following, binary to X coordinate conversion for line following, and structures to handle filters, PID and polar coordinates.

## **Integrated System**

The system of Haro contains the following blocks:

Processor – The processor will handle all information processing locally.

Haro uses a main circuit board custom designed by myself. It makes use of a small form board called the Teensy++ that uses AT90USB646 AVR processor and expands it by including headers, debug leds, RC filter options for Analog to Digital conversion, and power supplies specifically positioned to fit the needs of both prototyping and ease.

Vision – Vision encompasses most of the sensory perception of the robot.

Haro uses two pairs of Short Range Infrared sensors on each side for wall following and an array of digital reflectance sensors for line following.

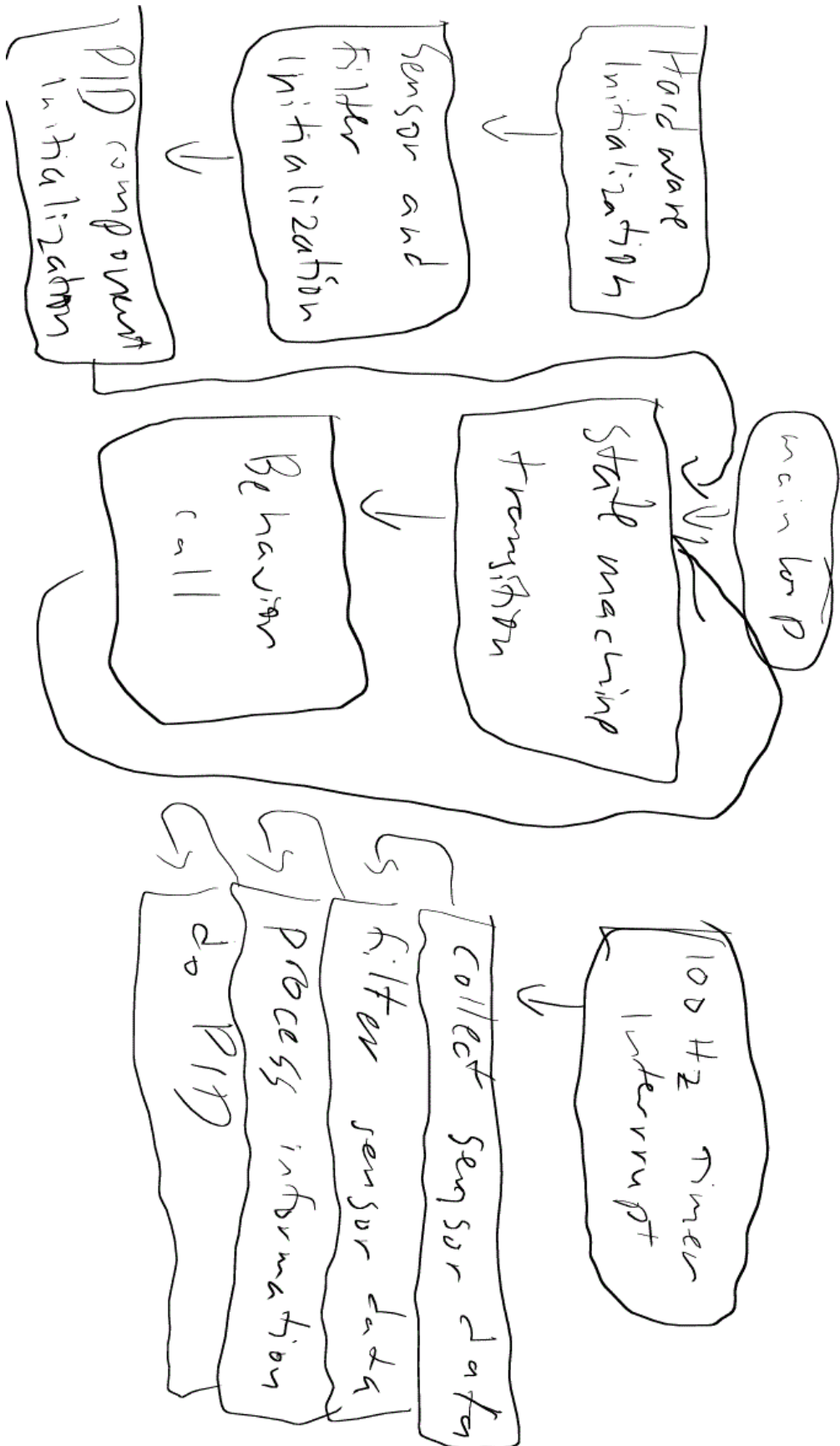
Actuation – Actuation refers to movement.

Actuation of Haro is handled using a custom optically isolated motor driver board designed by myself and controls a twin motor gear box that uses two DC motors and small truck wheels.

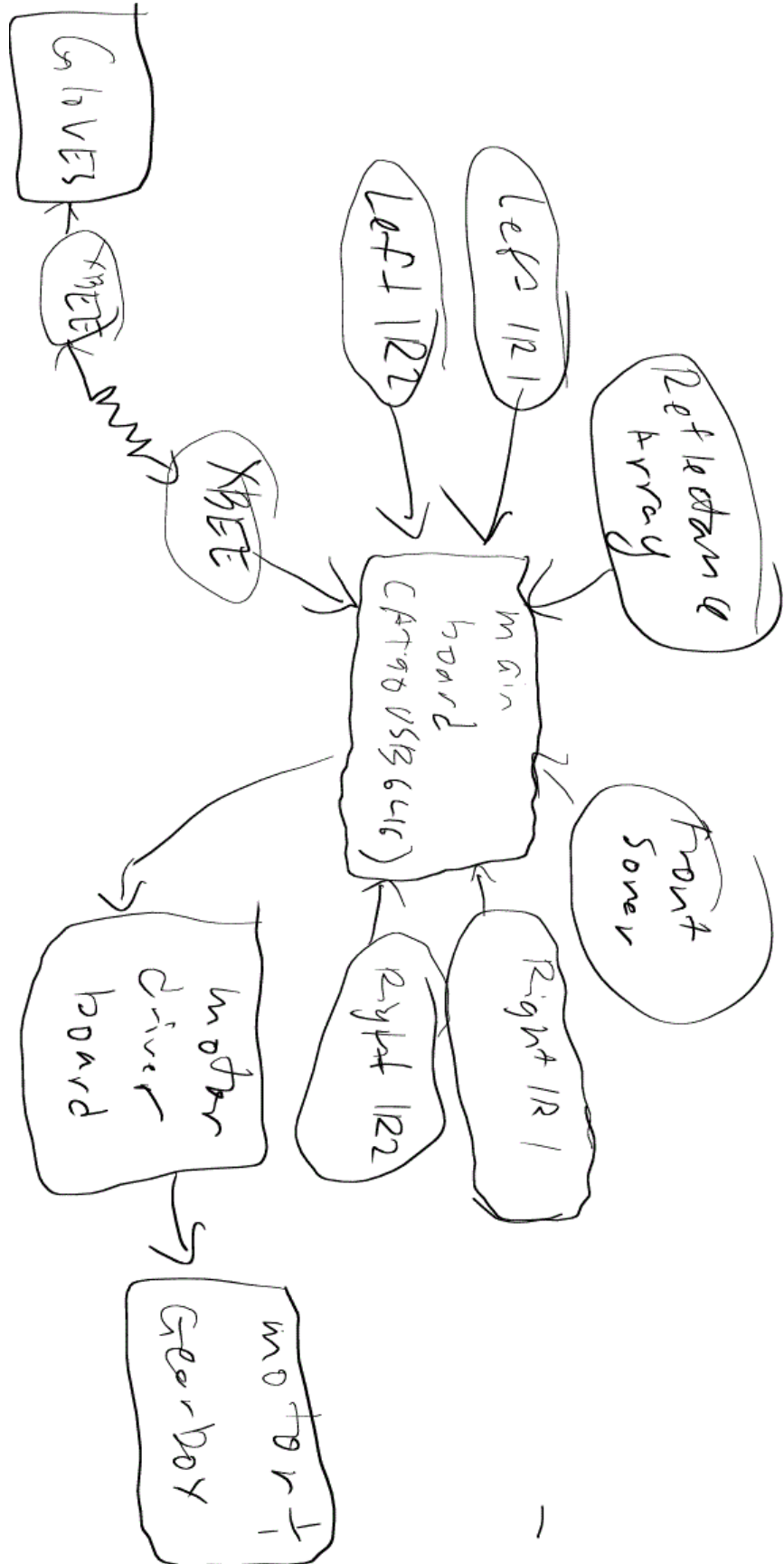
Manual Control – Haro can also be controlled using a pair of wireless motion and stress sensing gloves also custom designed by myself.

The system of Haro consists of these four blocks integrated into one embedded system. The vision consists of how the robot will take information from the real world. This information is processed within the main board. Using that processed information, control signals are sent to the motor driver board causing motion. The gloves on the other hand provide means of complete human interaction with the robot.

# Software Flow Chart

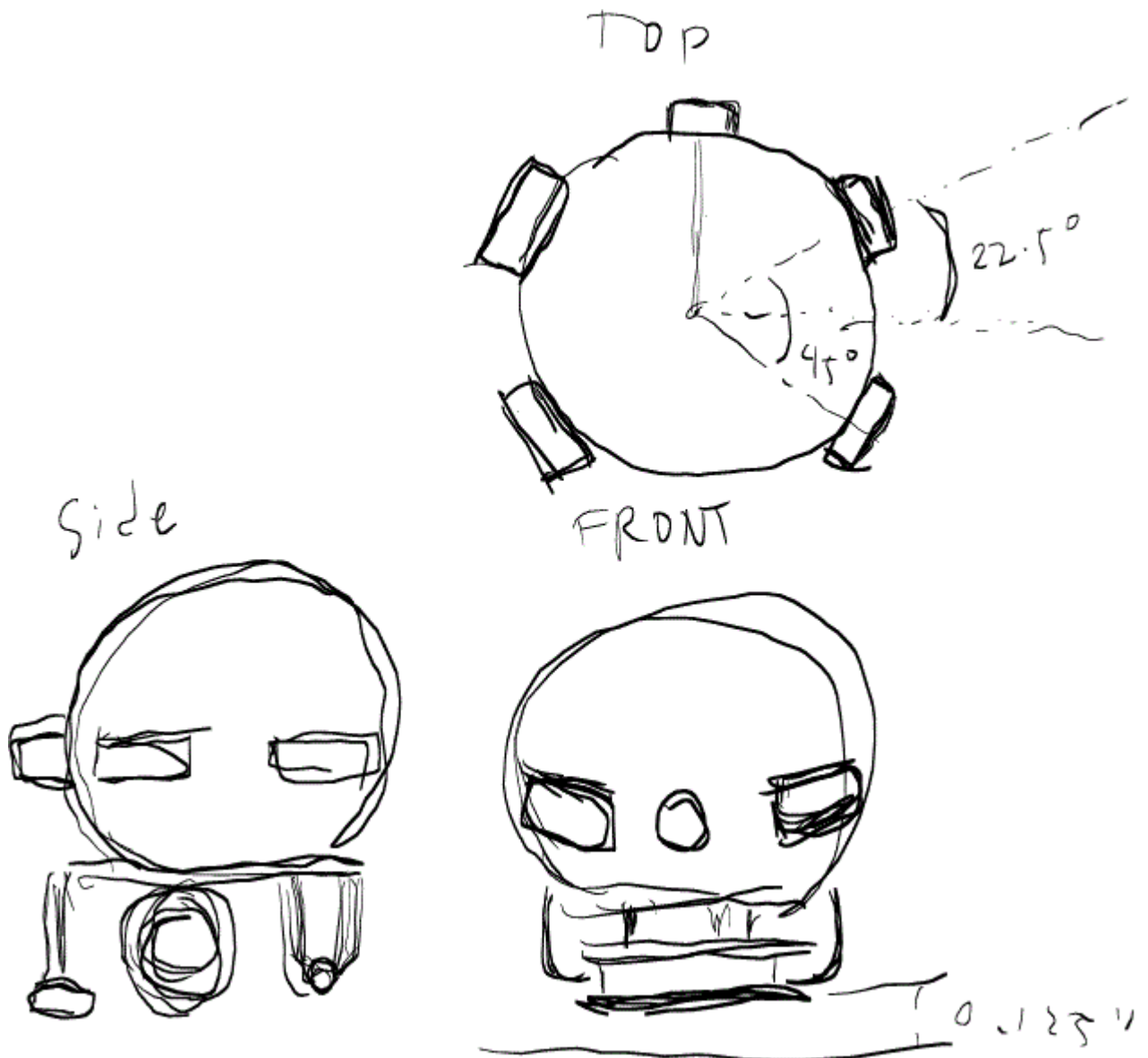


# Hardware Block Diagram



## Mobile Platform

The entire robot is in the shape of a sphere to accomplish the task of imitating the original animation character. The spherical body itself sits on top of an acrylic round platform that carries the whole weight of the robot. There are two pairs of Sharp Infrared Short Rangefinders on each side of the robot spaced 45 degrees. This set up makes it easy to find the yaw or the clockwise or counter clockwise rotation parallel to ground by finding the vector sum of one pair and finding its arc tangent. A sonar is placed right in front. The reflectance sensor array is placed in front and .125" above the ground to achieve optimum reflectance. All electronics and batteries are placed inside the spherical body. The twin gear box with truck wheels is on the bottom of the acrylic platform with a 1.5" caster as a third wheel.



## Actuators

Actuation includes two bi-directional, geared DC Motors inside a twin gearbox set that rotates two small truck wheels. The motors themselves are controlled using a custom optically isolated H Bridge motor driver board designed by myself. It contains three dual channel opto-isolators, two dual channel H Bridge drivers, a heat sink and separate power supply for the motors, opto-isolators and H Bridge drivers.

The opto-isolators consist of an LED-Phototransistor configuration. It uses the LED to turn on a phototransistor allowing current to flow inside the phototransistor, basically acting as a non contact switch. The H Bridge driver consists of an H Bridge transistor configuration that allows bidirectional flow of current into the motors with speed control using the enable lines that can take pulse width modulated signals.

The motor driver board was designed to completely electrically isolate the power of the motors and the power of the microprocessor. This eliminates power dip, microprocessor reset and electromagnetic interference. Also, since the motors themselves are rated for a peak current of 2.1A each and the H Bridge are rated to provide about 1A per channel; two channels are stacked for each H Bridge to provide a constant 2A of current per motor.

Coupled with the heat sink, this allows for continuous current reversals due to reversing the motors without thermal shutdown inside the H Bridge driver.

## Sensors

Haro uses the following sensors: Sharp Infrared Short Rangefinders, Maxbotix LV-EZ1 Sonar, QTR-8RC Digital Reflectance Sensor Array, and my custom wireless gloves, Ehrgeiz.

### Sharp Infrared Short Rangefinders used on Wall Following Algorithm

A pair of these is mounted on each side of the robot. They are used primarily for my wall following algorithm that uses both the Cartesian coordinate and polar coordinates of the robot in reference to the wall. What I find unique about my algorithm is that instead of looking at the actual proximity information, I look at the angle of rotation parallel to ground or more commonly known as the yaw. Basically, since I know the angle between two IR which is 45 degrees, half of that would be the angle between the line of sight of the IR and the relative X axis of the robot in reference to the wall or the normal.

### QTR-8RC Digital Reflectance Sensor Array used on Line Following Algorithm

This array of 8 reflectance sensors is mounted an eight of an inch above ground. This is used primarily for my line following algorithm that uses the X coordinate of the robot in reference to the line normal to any point in the line track. This sensor uses a small Infrared-Photoresistor configuration with an RC charge-discharge circuit to measure reflectance. In order to read a reflectance value, the phototransistor is turned on in software for 11 microseconds, turned off, and after 1 milliseconds, the signal is read off an RC circuit. How it works is the capacitor in the RC circuit is charged for 11 microseconds, and then discharged afterwards. The time it takes for it to discharge is based on how much reflectance the phototransistor is looking at. In plain words, if the sensor is on white surface, it takes only a few microseconds to discharge, while it takes more than a few milliseconds to discharge on black surface.

### Maxbotix LV-EZ1 Sonar

The sonar is mainly used for general obstacle avoidance and as fail-safe for wall following during cases where the robot would have to make an inward turn from a corner that curves in as opposed to a corner that curves out.

### Ehrgeiz Wireless Sensor Gloves

This is my special sensor that translates hand gestures into wireless data packets communicated using wireless RF modules called XBEES. It uses flex sensors as a transducer to finger stress or bend and accelerometers for hand motion on both gloves. Unfortunately they would not communicate with the robot due to timing issues but they will be integrated in the system in the near future.

## Behaviors

The following are the behaviors of the robot:

Wall following by calculating yaw using the Cartesian and Polar coordinates in reference to the wall.

Haro is able to follow a wall with precision by compensating the change in the angle in reference to the wall. This angle is processed inside a PID algorithm. Based on the output of the PID, the robot rotates either clockwise or counterclockwise to compensate the error.

Line following using the X coordinate in reference to the normal of the line track.

Haro is able to follow a line by converting the current position into an X coordinate in reference to the line normal to the line track. This X coordinate is processed inside a PID algorithm. Based on the output of the PID, the robot rotates either clockwise or counterclockwise to compensate the error.

Autonomous switching among right or left wall following and line following.

Using a finite state machine, the robot is able to autonomously switch among left wall following, right wall following and line following.

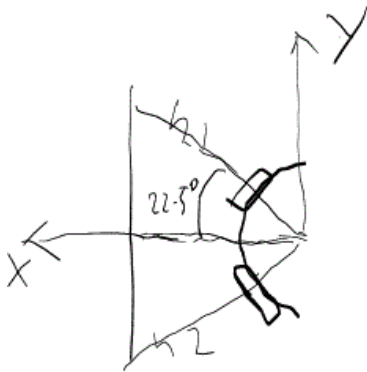
Manual control using Wireless Sensor Gloves.

Using a pair of wireless sensor gloves, the robot can be manually controlled or commanded to perform either wall following or line following.

## Experimental Layout and Results

The following figures show computations on how the wall following and line following algorithms are drawn out.

### Computation of Angle of Rotation using the Wall as a Reference



$$\begin{aligned} \text{PD: set point} &\Rightarrow \theta = 0 \\ K_P &= -1 \\ K_D &= \sim 0.008 \\ \Delta t &= \frac{1}{100} \text{ s} \end{aligned}$$

$$\sin(22.5^\circ) = 0.3826834323650897717284599840304$$

$$\cos(22.5^\circ) = 0.92387953251128675612818318939679$$

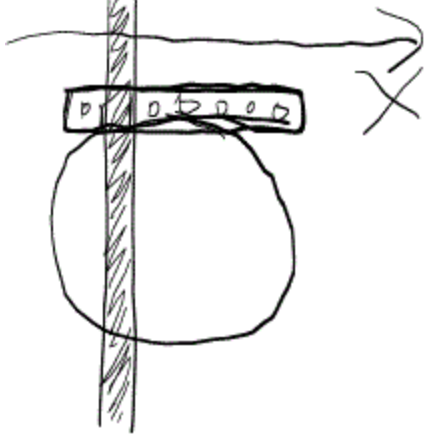
$$x_f = h_1 \cdot \cos(22.5) + h_2 \cdot \cos(22.5)$$

$$y_f = h_1 \cdot \sin(22.5) - h_2 \cdot \sin(22.5)$$

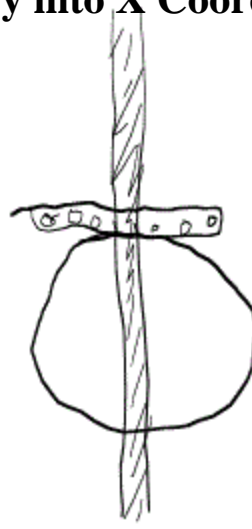
$$\theta = \tan^{-1} \left( \frac{y_f}{x_f} \right)$$

$$\text{vector sum} = \sqrt{(x_f)^2 + (y_f)^2}$$

# Linear Position into Binary into X Coordinate conversion



$$X < 0$$



$$X = 0$$



$$X > 0$$

PID = 5

set point  $\Rightarrow$   $x = 0$

$$K_P = -1$$

$$K_D = -0.008$$

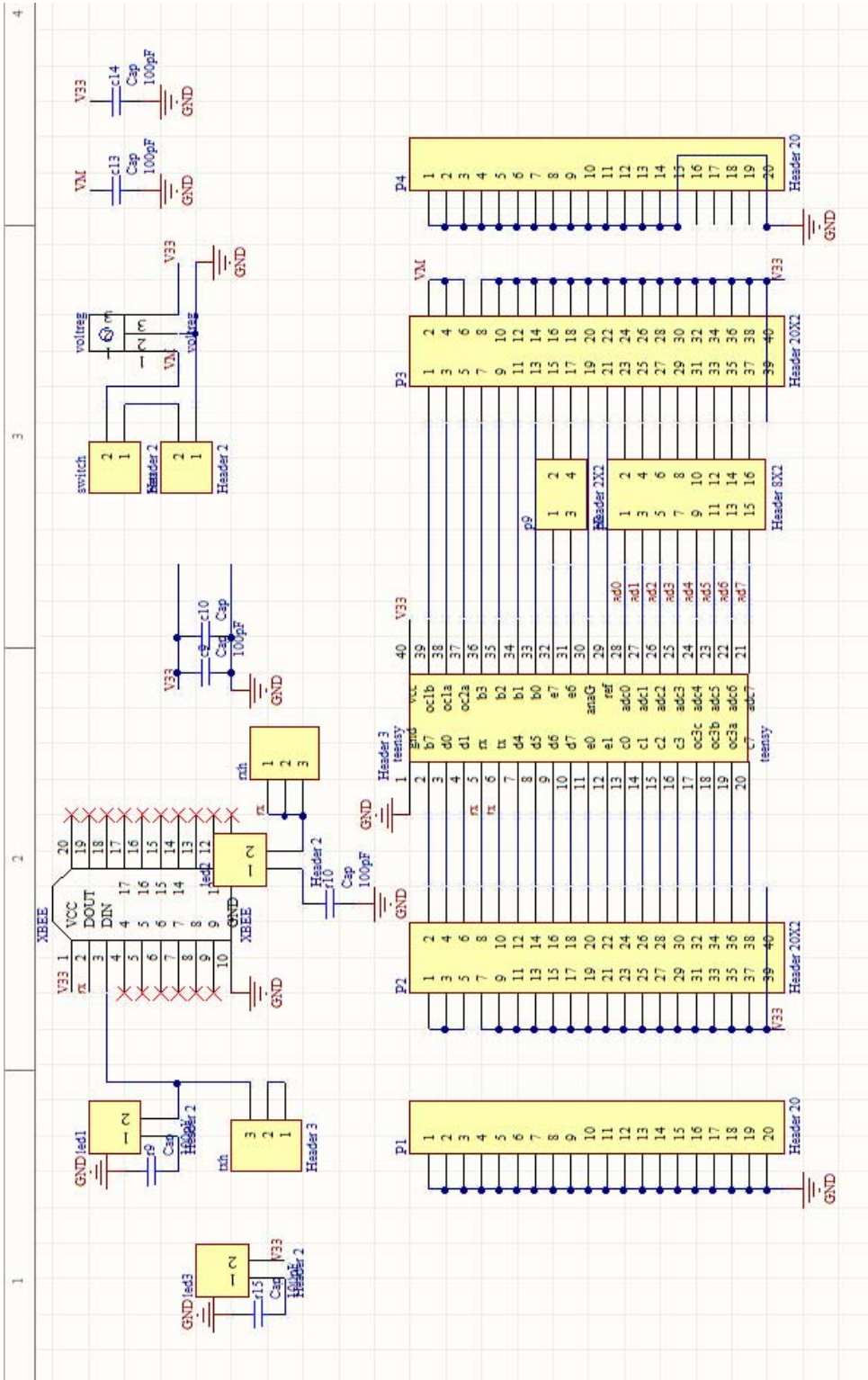
$$K_I = -0.8$$

$$dt = \frac{1}{100} \text{ s}$$

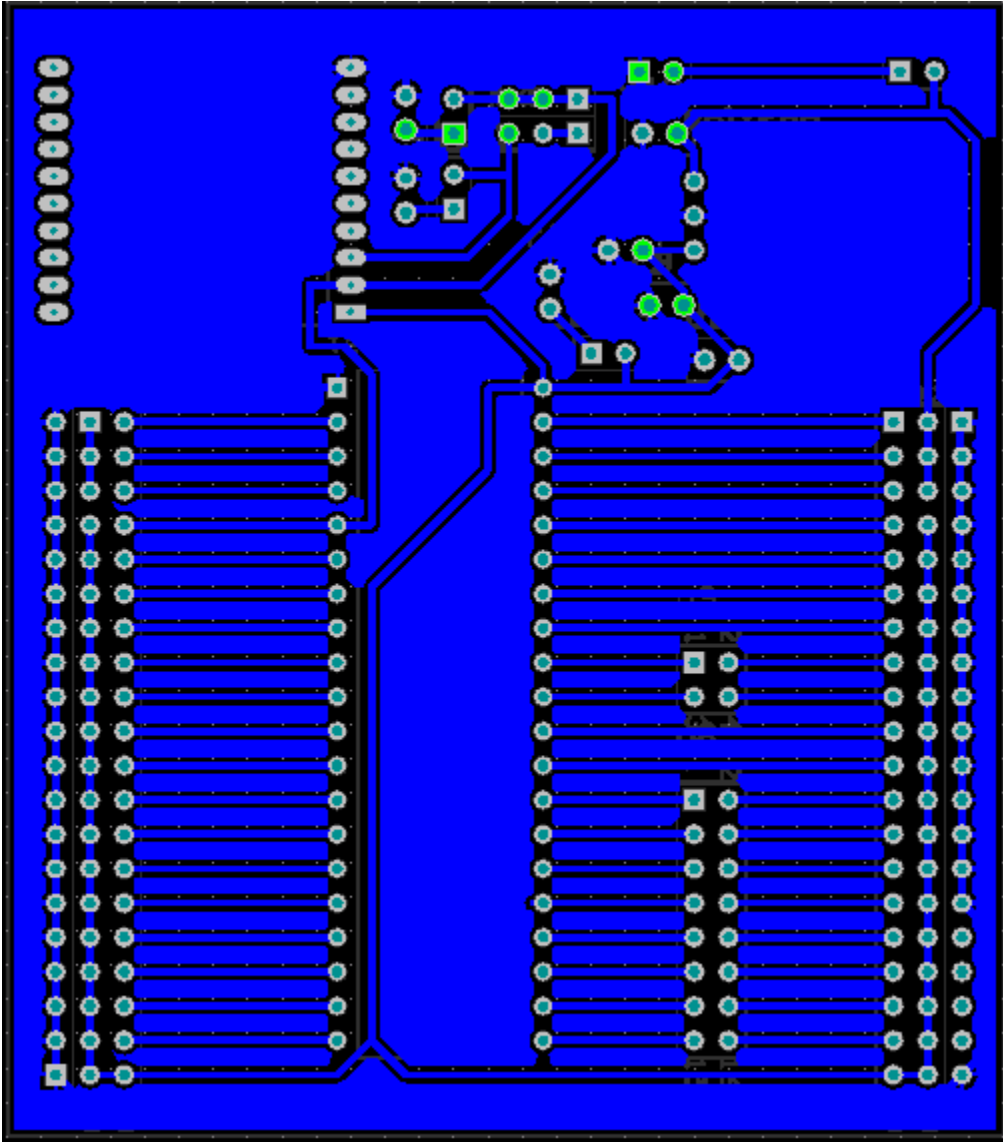
## **Conclusion**

Haro is able to accurately follow both a wall and a line. This project was more on researching and experimenting different algorithms to control behaviors of a robot. It was both a stressful and fun experience. The robot is still in its incomplete stage since the gloves were not interfaced in time for the final demonstration. However, I was very much contented with how the robot worked in the end. I'm thinking of adding some more components in the near future like a camera, text to speech, more actuation, and a lot more behaviors.

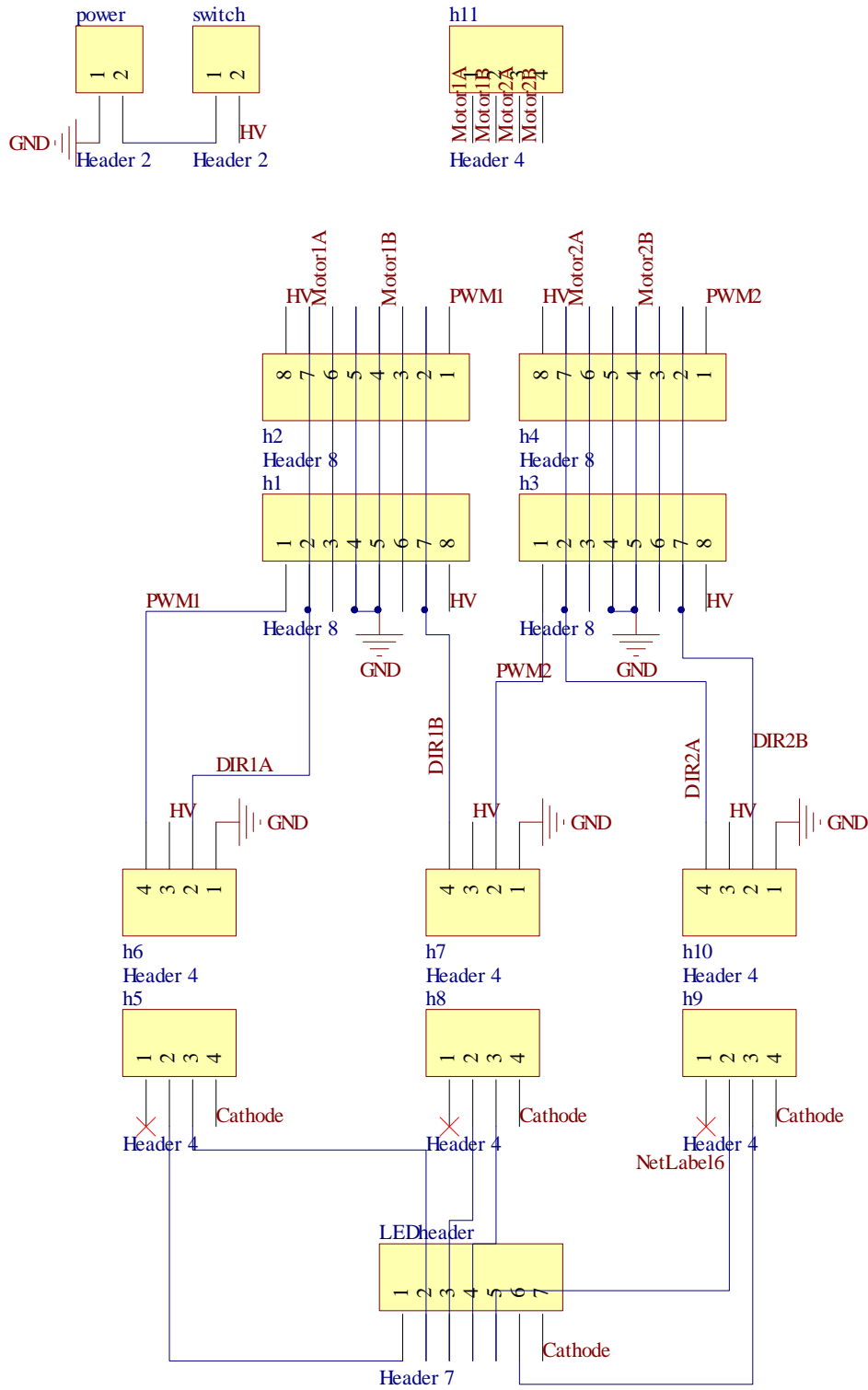
# Appendix A. Main Board Schematic



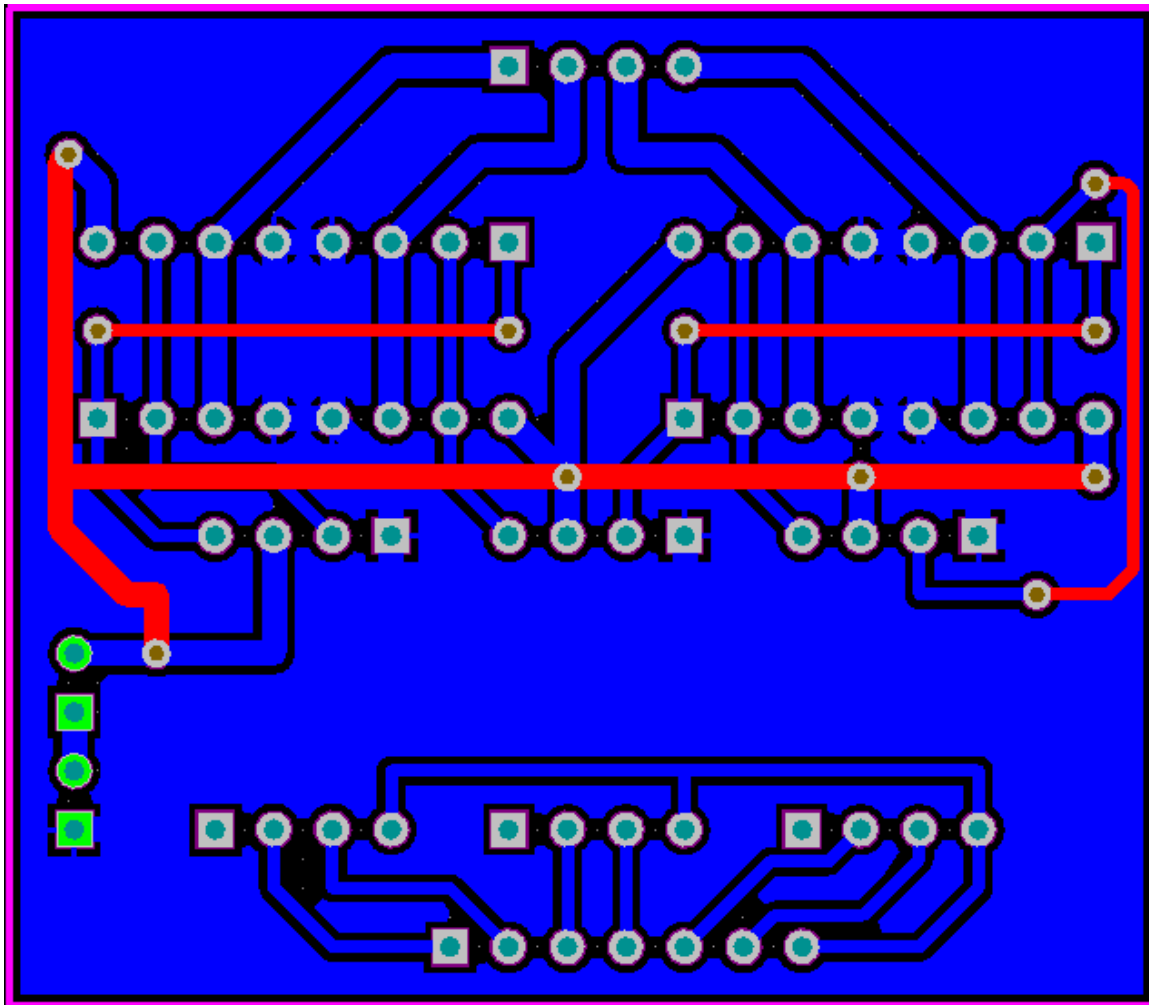
## Appendix B. Main Board PCB



# Appendix C. Motor Driver Board Schematic



## Appendix D. Motor Driver Board PCB



## Appendix E. Algorithms and Code Snippets

### 10 milliseconds interrupt routine.

```
ISR(TIMER1_COMPA_vect)
{
    rightir1.input = readADC(0);
    rightir2.input = readADC(1);
    leftir1.input = readADC(2);
    leftir2.input = readADC(3);
    obstacle.input = readADC(5);
    doFilter(&rightir1);
    doFilter(&rightir2);
    doFilter(&leftir1);
    doFilter(&leftir2);
    doFilter(&obstacle);
    findPolar(&rightPolar, rightir1.output, rightir2.output);
    findPolar(&leftPolar, leftir1.output, leftir2.output);
    rightDegree = (int)(ceil(rightPolar.degree));
    rightHypo = (int)(ceil(rightPolar.hypo));
    leftDegree = (int)(ceil(leftPolar.degree));
    leftHypo = (int)(ceil(leftPolar.hypo));
    findLineval();

    toggle;
}
```

### Sample Finite Auto switching State machine

```
switch(currentState)
{
    case RIGHTWALLFOLLOWSTATE:
        if(lineval == 0 && rightir1.output < 100 && rightir2.output < 100)
            currentState = LINEFOLLOWSTATE;
        else
            currentState = RIGHTWALLFOLLOWSTATE;
        rightwallfollow();
        break;
    case LINEFOLLOWSTATE:
        if(rightir1.output > 100 && rightir2.output > 100)
            currentState = RIGHTWALLFOLLOWSTATE;
        else
            currentState = LINEFOLLOWSTATE;
        linefollow();
        break;
    default: currentState = currentState; break;
}
```

## Structures

```
typedef struct filter
```

```
{  
    int input;          //input to filter  
    int filtarray[FILTLENGTH];  
    int index;  
    int oldest;  
    int sum;           //sum of inputs  
    int output; //output of filter  
}filter;              //filter structure
```

```
typedef struct pid
```

```
{  
    double setPoint;          //set point to PID  
    double processValue; //input to pid  
    int output;              //output of pid  
    double previousError; //error of previous cycle  
    double previousPreviousError;  
    double integral;         //sum of all errors  
    double derivative;      //rate of change of error  
    double kp;               //proportional constant  
    double ki;               //integral constant  
    double kd;               //derivative constant  
}pid;                        //pid structure
```

```
typedef struct polar
```

```
{  
    double Xcoord;  
    double Ycoord;  
    double radians;  
    double degree;  
    double hypo;  
}polar;
```

## Cartesian to Polar coordinate conversion

```
void findPolar(polar * p, int a, int b)
{
    p->Xcoord = (double)(a*0.92387953251128675612818318939679) +
                (double)(b*0.92387953251128675612818318939679);
    p->Ycoord = (double)(a*0.3826834323650897717284599840304) -
                (double)(b*0.3826834323650897717284599840304);
    p->radians = atan2(p->Ycoord,p->Xcoord);
    p->degree = p->radians*div180PI;
    p->hypo = hypot(p->Ycoord,p->Xcoord);
}
```

## 8 bit sample to X Coordinate conversion

```
void findLineval(void)
{
    switch(test)
    {
        case 0b00000000 : if(prevlineval<=0) lineval = -12; else lineval = 12;
break;
        case 0b11111111 : lineval = 0; break;
        case 0b11111000 : lineval = -11; break;
        case 0b11111100 : lineval = -10; break;
        case 0b11111110 : lineval = -9; break;
        case 0b10000000 : lineval = -8; break;
        case 0b11000000 : lineval = -7; break;
        case 0b11100000 : lineval = -6; break;
        case 0b11110000 : lineval = -5; break;
        case 0b01100000 : lineval = -4; break;
        case 0b01110000 : lineval = -3; break;
        case 0b01111000 : lineval = -2; break;
        case 0b01111100 : lineval = -1; break;
        case 0b00110000 : lineval = 0; break;
        case 0b00111000 : lineval = 0; break;
        case 0b00111100 : lineval = 0; break;
        case 0b00011000 : lineval = 0; break;
        case 0b00011100 : lineval = 0; break;
        case 0b00001100 : lineval = 0; break;
        case 0b00001110 : lineval = 0; break;
        case 0b00111110 : lineval = 1; break;
        case 0b00011110 : lineval = 2; break;
        case 0b00001110 : lineval = 3; break;
        case 0b00000110 : lineval = 4; break;
        case 0b00001111 : lineval = 5; break;
        case 0b00000111 : lineval = 6; break;
    }
```

```

        case 0b00000011 : lineval = 7; break;
        case 0b00000001 : lineval = 8; break;
        case 0b01111111 : lineval = 9; break;
        case 0b00111111 : lineval = 10; break;
        case 0b00011111 : lineval = 11; break;
        default: lineval = prevlineval; break;
    }
    prevlineval = lineval;
}

```

## Filter Codes

```

void initFilter(filter *f, int i)
{
    int localsum=0, localoutput=0;
    f->index = 0;
    for(int i = 0; i<FILTLENGTH; i++)
        f->filtarray[i] = 0;
    f->sum = 0;
    f->output = 0;
    for(int i = 0; i<16; i++)
    {
        _delay_ms(10);
        localsum += readADC(i);
    }
    localoutput = localsum >> 4;
    f->output = localoutput;
    f->sum = 0;
    for(int i = 0; i< FILTLENGTH; i++)
    {
        f->filtarray[i] = f->output;
        f->sum += f->output;
    }
    f->oldest = f->output;
}

void doFilter(filter *f)
{
    f->filtarray[f->index] = f->input;
    f->sum = f->sum - f->oldest + f->input;
    f->output = f->sum >> FILTPOWER;
    f->index++;
    if(f->index == FILTLENGTH)
        f->index = 0;
    f->oldest = f->filtarray[f->index];
}

```

## PID Algorithm

```
void initPid(pid *p, double sp, double pe, double i, double kp, double ki, double kd)
{
    p->setPoint = sp;
    p->previousError = pe;
    p->previousPreviousError = pe;
    p->integral = i;
    p->kp = kp;
    p->ki = ki;
    p->kd = kd;
}

int pidfreq = targetfreq;

int doPid(pid *p)
{
    //int output;
    double dt = 1/((double)pidfreq);
    double error = p->setPoint - p->processValue;
    p->integral = p->integral + error*dt;
    if(error == 0 && p->previousError == 0 && p->previousPreviousError == 0)
        p->integral = 0;
    p->derivative = ( error - p->previousError )/dt;
    double tempOutput = (p->kp*error) + (p->ki*p->integral) + (p->kd*p-
>derivative);
    p->output = (int)tempOutput;
    p->previousError = error;
    p->previousPreviousError = p->previousError;
    return p->output;
}
```