

# **ECE 456/556 Project**

## **Unmanned Vehicle Design Project**



Group UAV05  
Dana Baumann (ECE456)  
Frederick Livingston (ECE556)  
Nathan Pater (ECE556)

December 12, 2005

## Table of Contents

1. Introduction.....	Page 3
2. PID Control System.....	Page 4
3. PID Controller Algorithm.....	Page 7
4. PID Controller Block Diagram & Mathematical Model .....	Page 9
5. Ultrasonic Receiver Design.....	Page 10
6. Results & Discussion.....	Page 13
7. References.....	Page 14

**Ensure pages are correct**

## 1. Introduction

Unmanned vehicle (UV) Mechatronics project requires the design of PID controllers to control each wheel of the UV allowing it to move along a predefined path. This can be accomplished by the use of sensors and control implementation skills introduced in this course. Each group will be evaluated and graded based on the following criteria:

- 1) Time required for the UV to move along the predefined path – see figure 1A
- 2) Error ( $e$ ) tracking while moving along the predefined path and its accuracy – see figure 1B
- 3) Parking the UV ( $y$ ) as close as possible to the block ( $x$ ) at the end of the predefined path without touching the block – see figure A. This requires us to build our own receiver in lieu of using the one already mounted on the UV.

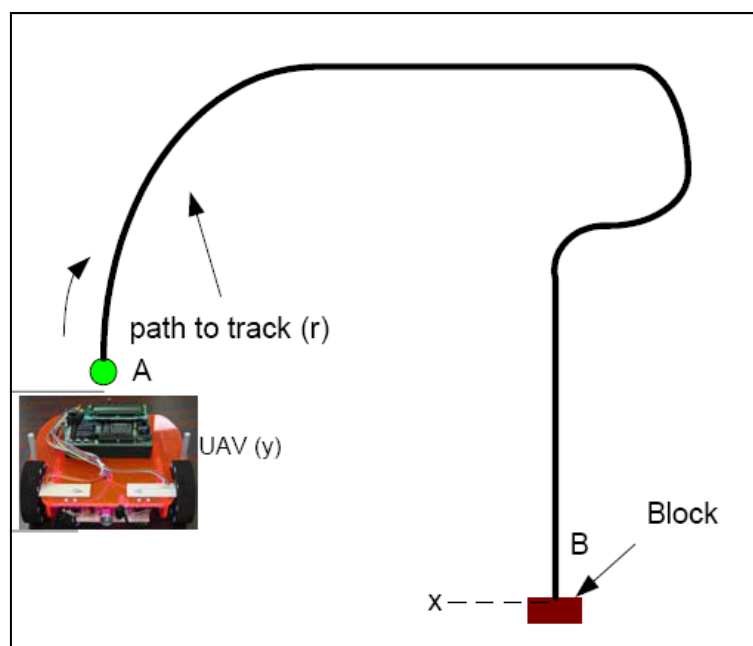


Figure 1A

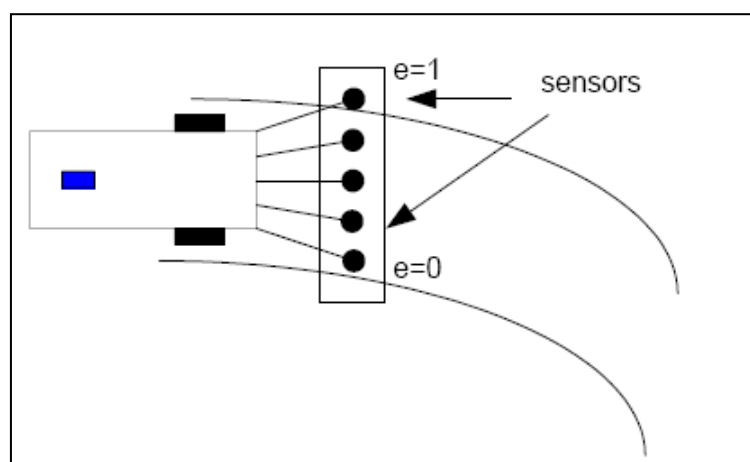
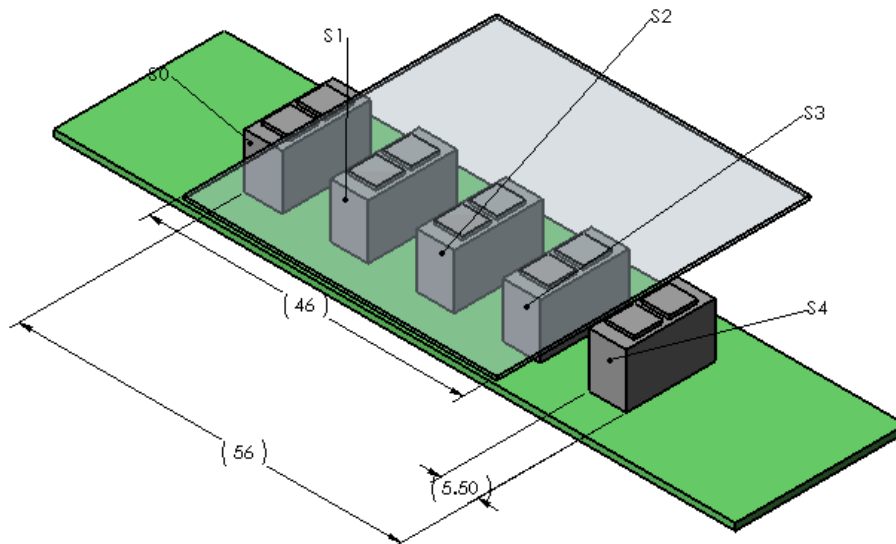


Figure 1B

## 2. PID Control System

The unmanned vehicle uses 5 parallel infrareds emitters and detectors to track a line that lies on a flat surface. The line is constructed by using black 46 millimeter diameter tape and contains curvature and straight paths. The path contains no intersections.

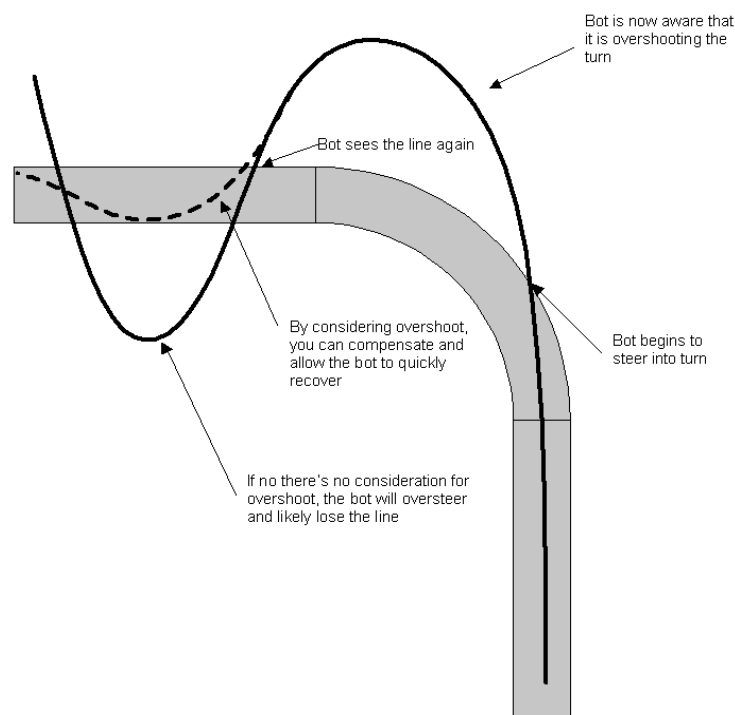
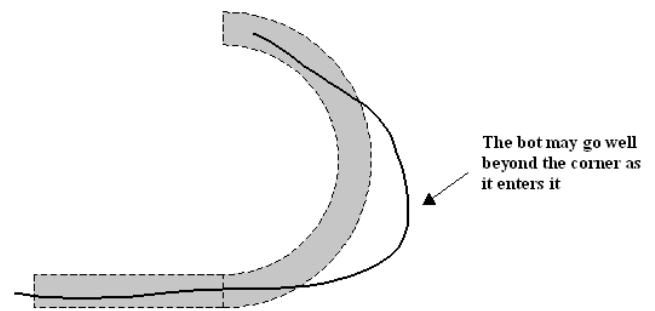


It is possible to use a single sensor for navigation that would follow one edge of the line. Using a 2 motor design, one motor is active when the line is seen and the other motor is inactive. This design will only work when the robot is in close proximity of the line. If the position error is greater than the radius of the sensor, the robot will spin a circle and never regain sight of the line. Another algorithm is to use the two outer most sensors (S0 and S4) to detect the line. These sensors would then straddle the line. If the sensor does not detect the line it would output a zero, if it detects a line it would output one. If the signal detects the line it would adjust the motor parameters to move away from the line.



S0	S1	S2	S3	S4	
0	1	1	1	0	Straddling the line or lost the line
1	1	1	1	0	Found left side of the line
0	1	1	1	1	Found right side of the line

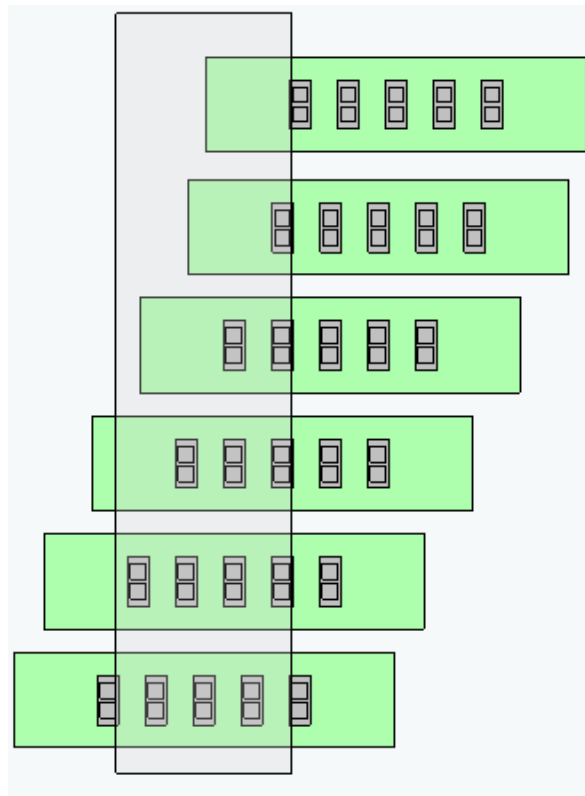
Using all 5 sensors is believed to increase the system performance. Using the additional sensors S1, S2, and S4 increases sensitivity resolution that the sensors have of sensing the line's location. As the speed of the system increases the robot will tend to go well beyond the corners. This over steering can be reduced by either reducing robotics speed or with proper planning.



In the diagram above, the robot loses the line as it entered the turn. Since it began to turn left as it entered the turn, it knows to continue to turn left until it finds the line. The far left sensor will find the line (at a rather acute angle) before any other sensor. The normal logic for only the left sensor seeing the line would be to steer hard left. However, in this case, that would cause us to cross the line and not turn into it. By recognizing the over steer condition, we can reverse the left only sensor logic to turn hard right instead of left. This steers the robot back onto the track instead of across it. This scenario is why the use of 5 sensors is important. The use of 5 sensors provide for a finer level of control and quicker response to overshoot/over steer.

The sensors output would be as following:

00000	Lost line from overshoot or break in line
10000	Almost off the line, steer hard right and reduce speed
11000	Near the left edge of the line, steer right
11100	Left of the center of the line, steer right
11110	Slightly to the left of the center of -the line, slight correction to the right
01110	Centered over line, increase speed for straight runs.
01111	Slightly to the right of the center of the line, slight correction to the left
00111	Right of the center of the line, steer left
00011	Near the right edge of the line, steer left
00001	Almost off the line, steer hard left and reduce speed
11111	Line intersection



Instead of using a binary approach, using an analog sensor value from 0-255 will gain the system even more sensitivity.

### 3. PID Controller Algorithm

It is desired to obtain a linear relationship between line position and sensor output. The proposed algorithm uses the outer 2 sensors to determine the direction of error, and the middle sensors control the strength of the error. The first step is retrieving the analog signal from the individual sensors.

// snippet 1: Capture sensor values

S0 = analog(0); // S0 is an integer value between 0 and 255

S4 = analog(4); // S4 is an integer value between 0 and 255

If sensor0 has a value greater than sensor4 then the robot is off center to the left. If sensor4 has a value greater than sensor0 then the robot is off center to the right. The direction variable is used to note the direction of error.

// snippet 2: Determine the direction of error

If (S0 > S4)

direction = -1;

Else

direction = 1;

The next section of code concatenates the sensors values into a single output value of 0 to 2550.

// snippet 3: Compute output value

S0 = (int) (1 - S0/255.0)\*255\*direction;

S1 = (int) (1 - S1/255.0)\*255\*direction;

S2 = (int) (1 - S2/255.0)\*255\*direction;

S3 = (int) (1 - S3/255.0)\*255\*direction;

S4 = (int) (1 - S4/255.0)\*255\*direction;

Output = S0 + S1 + S2 + S3 + S4 + 1275;

For example if the robot is located far right of center, the sensor values are: S0 = S1=S2=S3=0 and S4 =1. Since S0 is not greater than S4 the direction is set to one. The output is calculated to be 2549.

Position	S0	S1	S2	S3	S4	Alg1	Alg2	Alg2-Adj.	
Far Right 23 mm	0	0	0	0	0	-1275	-1275	0	
	1	0	0	0	0	-1274	-1274	1	
	127	0	0	0	0	-1148	-1148	127	
	255	0	0	0	0	-1020	-1,020	255	
	255	127	0	0	0	-893	-893	382	
	255	255	0	0	0	-765	-765	510	
	255	255	127	0	0	-638	-638	637	
	255	255	255	0	0	-510	-510	765	
Center	255	255	255	127	0	-383	-383	892	
	255	255	255	255	0	-255	-255	1020	
	127	255	255	255	0	-383	127	1402	
	0	255	255	255	0	510	0	1275	
	0	255	255	255	127	383	127	1402	
	0	255	255	255	255	255	255	1530	
			127	255	255	255	383	383	1658
		0	0	255	255	255	510	510	1785
-23mm Far Left			127	255	255	638	638	1913	
		0	0	0	255	765	765	2040	
				127	255	893	893	2168	
		0	0	0	255	1020	1,020	2295	
		0	0	0	127	1148	1,148	2423	
		0	0	0	1	1274	1,274	2549	
		0	0	0	0	1275	1,275	2550	

The above table shows example calculation of output given position. Notice with algorithm one the output is non-linear. The output appears linear except when the robot is in the center; therefore, this is treated as a special case. Alg2 deals with this case by eliminating data from the center sensors. Alg2-Adj simply offsets the value to give a linear response starting at 0. Below is the modified code:

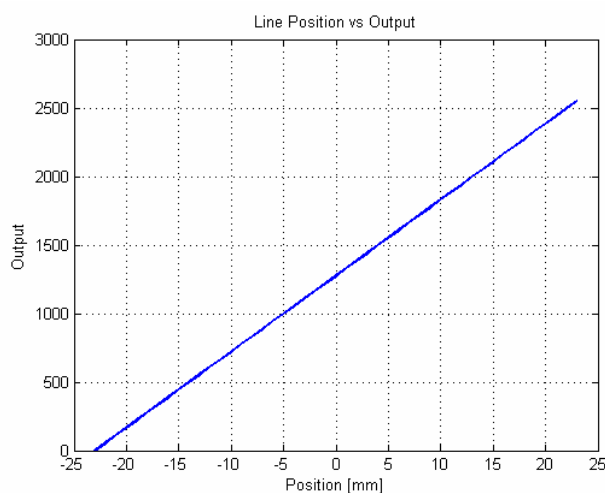
```

S0 = analog(0);
S1 = analog(1);
S2 = analog(2);
S3 = analog(3);
S4 = analog(4);

If (S0 > S4)
    direction = -1;
Else
    direction = 1;

// if the center sensor are all with in 90% of max
// fix center special case
If (S1 > 230 && S2 > 230 && S3 > 230){
    If (direction == -1){
        S0 = S0*direction;
        S1 = 0; S2 = 0; S3 = 0; S4 = 0;
    }
    Else{
        S4 = S4*direction;
        S0 = 0; S1 = 0; S2 = 0; S3 = 0;
    }
}
Else{
    S0 = (int) (1 - S0/255.0)*255*direction;
    S1 = (int) (1 - S1/255.0)*255*direction;
    S2 = (int) (1 - S2/255.0)*255*direction;
    S3 = (int) (1 - S3/255.0)*255*direction;
    S4 = (int) (1 - S4/255.0)*255*direction;
}
Output = S0 + S1 + S2 + S3 + S4 + 1275;

```

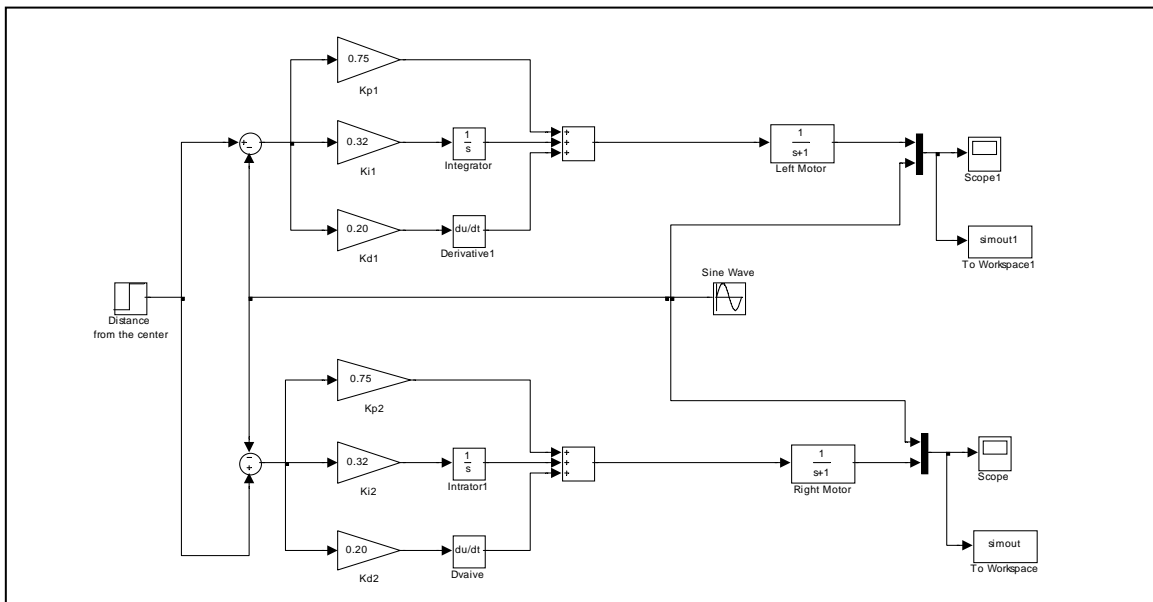
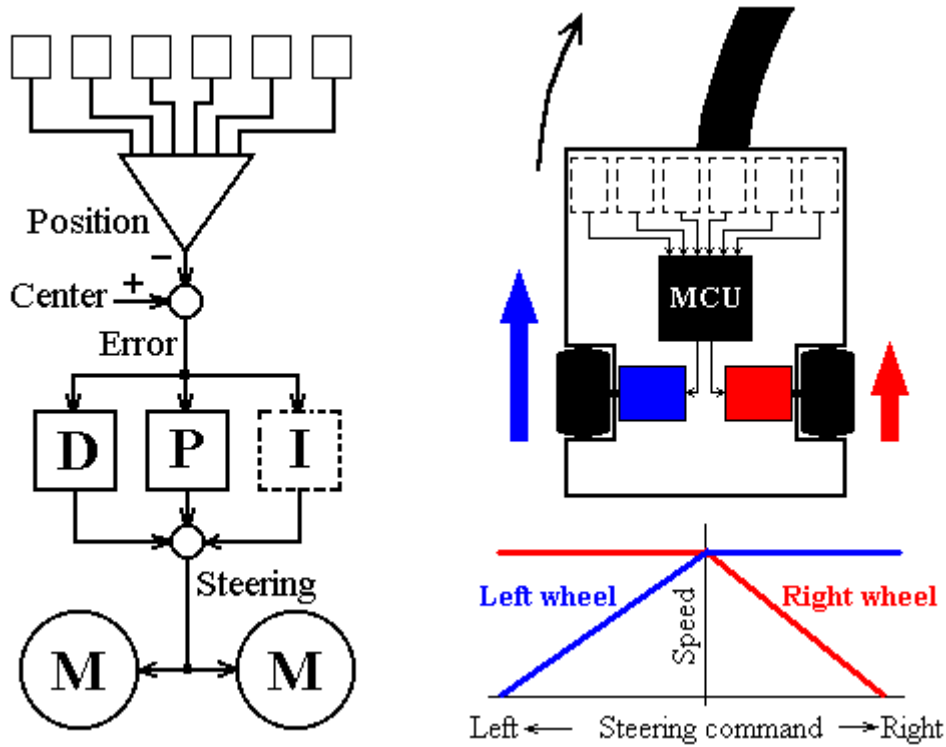


The image shows the actual line position vs. detected line position in the center value of 1275.

Where the Output Equation is:

$$Output(pos) = \frac{1275}{23}(pos) + 1275$$

### 4. PID Control Block Diagram & Mathematical Model



Mathematical Model

## 5. Ultrasonic Receiver Design

As listed in the introduction, one of the evaluation criteria of this project is how close we can park the UV to the end of the predefined path. To accomplish this task, we were required to design and build our own ultrasonic receiver by using the information provided to us as shown in figures 5A and 5B, and by implementing the circuit shown in figure 5C.

Figure 5A shows how the receiver is connected to the handy board

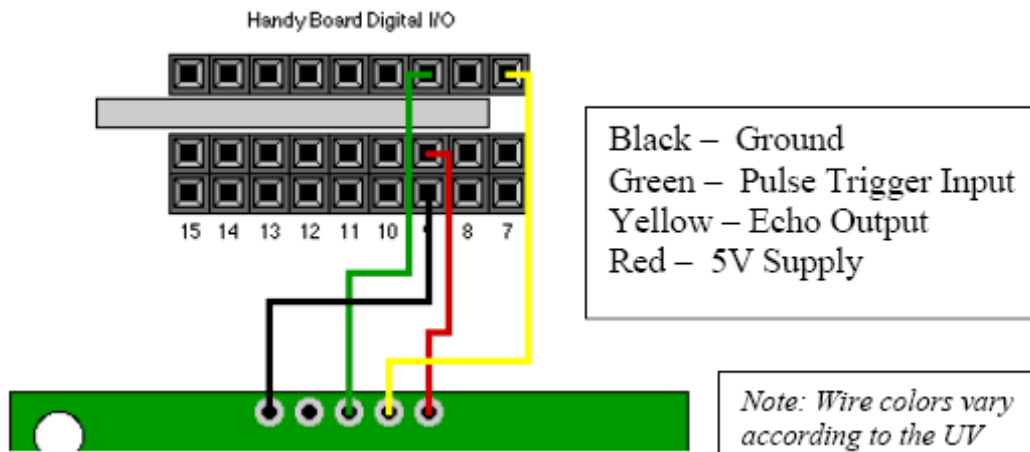


Figure 5A

Figure 5B shows the timing diagram which describes the operational characteristics of the ultrasonic range finder.

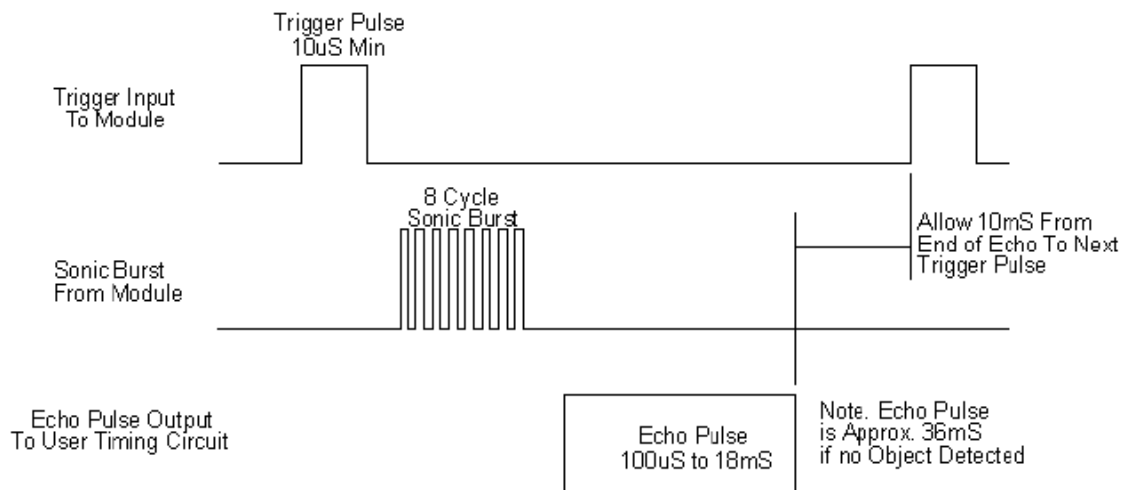


Figure 5B

Using the following basic schematic, we designed and built our own power converter circuit and receiver circuit to be mounted on our UV – see figure 5D. We basically implemented the A & B blocks shown below:

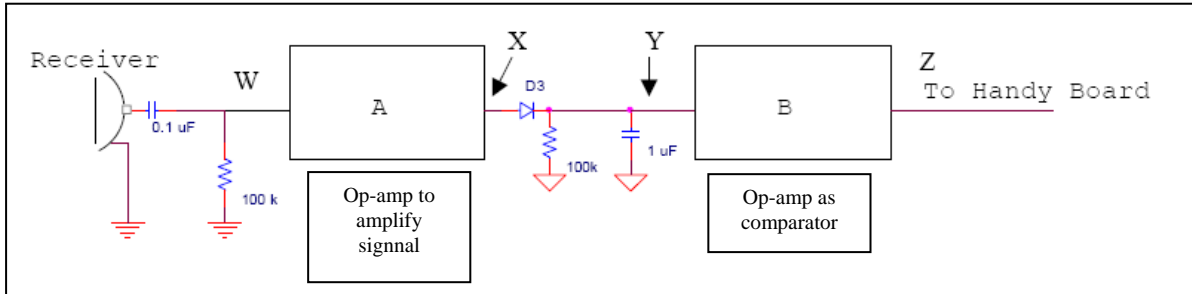
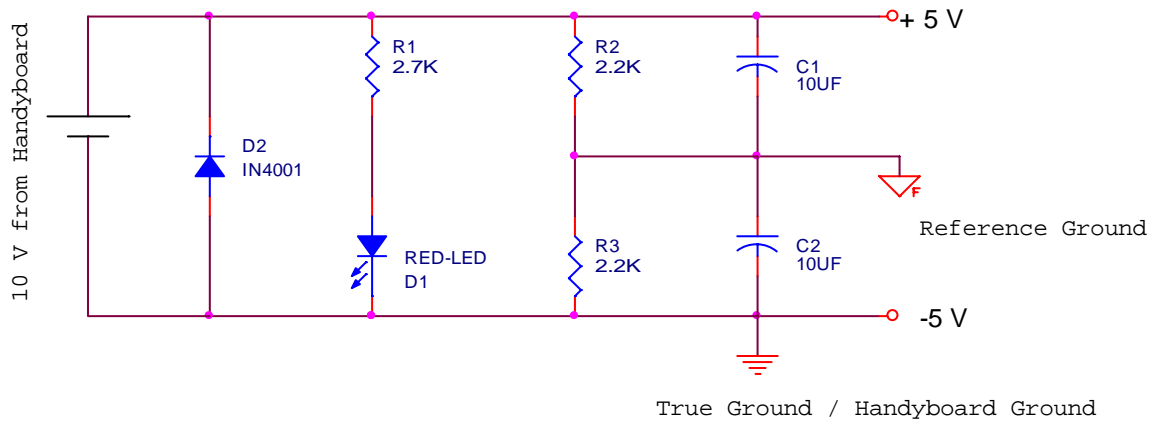
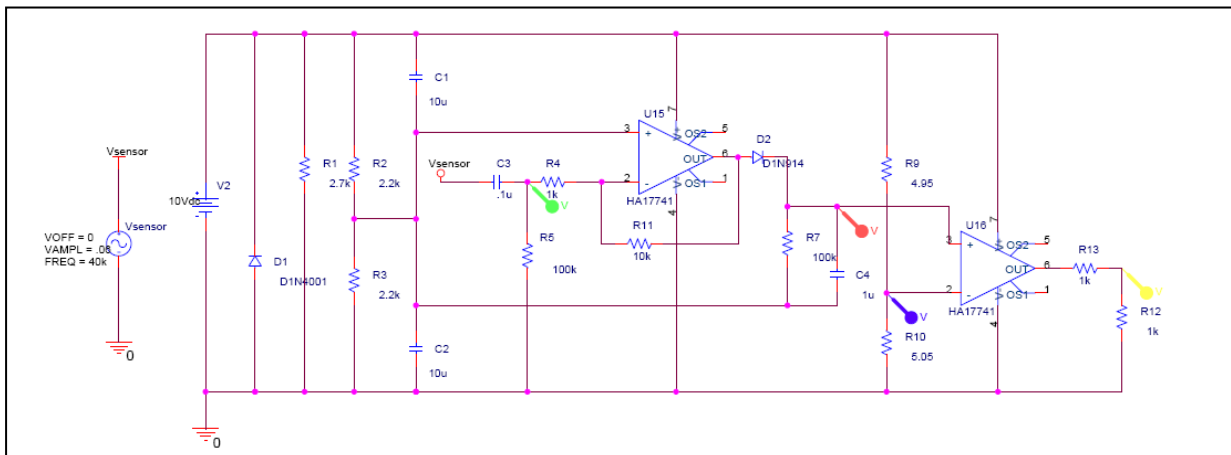


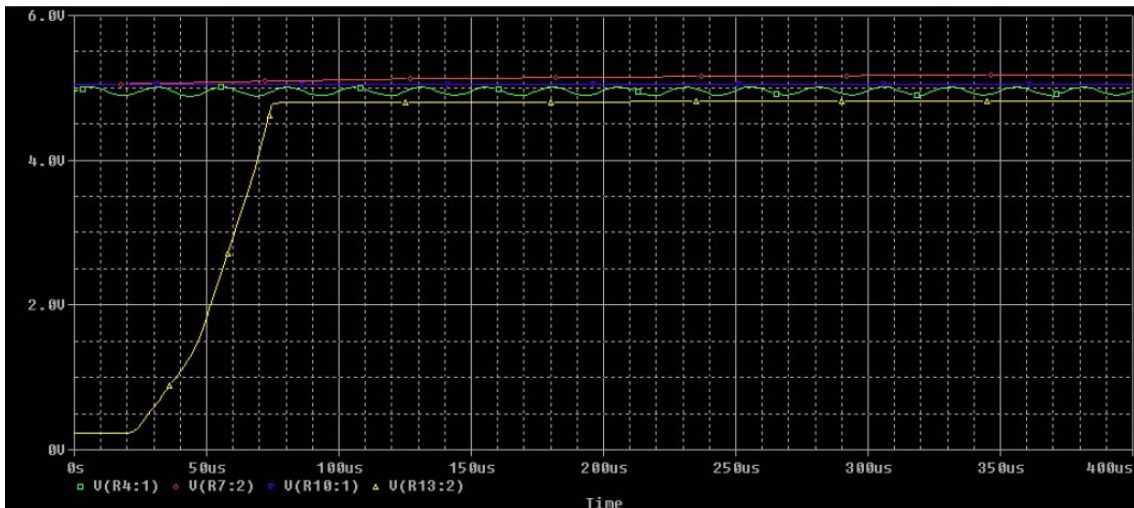
Figure 5C



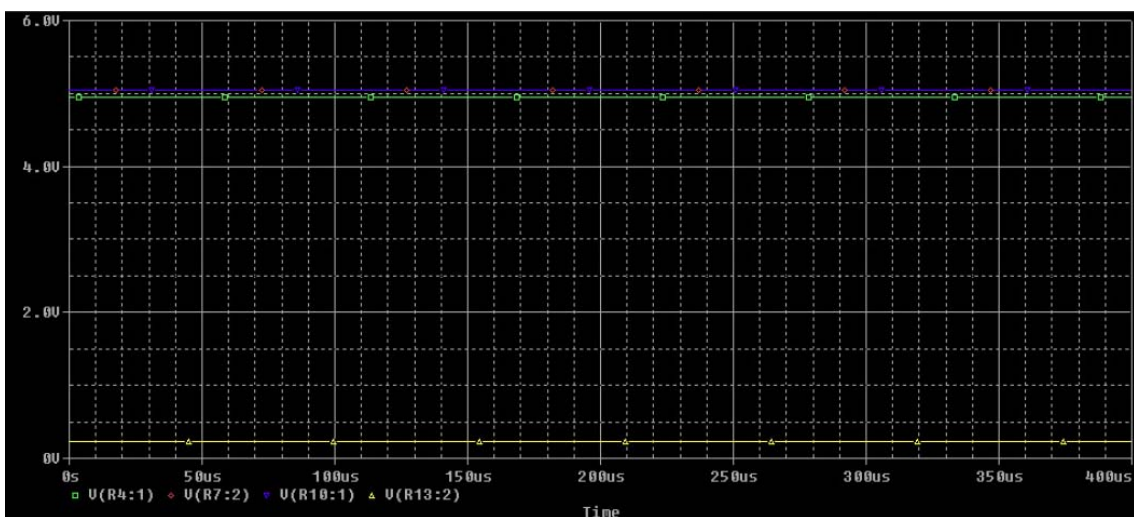
True Ground / Handyboard Ground



Implemented Sonar Circuit



Sonar Circuit Pulse Received



Sonar Circuit No Pulse Received

After constructing the sonar circuit, the final task was to tune the second op-amp functioning as a comparator. The input voltage from the first op-amp was measured while receiving a sonar signal and while not receiving a signal to determine where the comparator voltage should be set. If the input with the sonar signal is greater than the comparator value the output of the circuit will be set to  $+V_{cc}$ . If the input voltage (without the sonar) is less than the comparator value, the circuit output will be zero volts. Since  $V_{cc}$  of the Handyboard is 10V, a simple voltage divider was used to cut the circuit output to the standard +5V for the digital interface.

Now that the Handyboard is able to detect incoming sonar pulses, the next step is to determine how far to travel before stopping just short of the finish line object. In testing, our UAV was consistently able to detect the object at a distance of roughly 10 inches. To improve the performance of our UAV, while maintaining a fast algorithm, a delay was added to the PID loop to continue forward for a set amount time to cover the remaining 10 inches to the finish line. This value was tweaked, with testing, and a 4.5 second delay was needed to consistently stop without hitting the object.

## 6. Results and Discussion

After completing our entire PID adjustments, IR sensing, error calculation, and Sonar sensing code the entire program was assembled into one continuous loop. The newly added code did slow our PID algorithm so adjustments were needed to tune the circuit for maximum speed and minimum error. The following table was constructed as various values were inserted for  $T_i$ ,  $T_d$  and  $K_p$  and a test run completed. Observations were also made of the UAVs performance in straight sections of the track and through the curves. This insight was important as each PID parameter would affect the performance of UAV differently depending on the curvature of the path. Note: the error values in the table were recorded using an outdated algorithm and the new error value is much higher, but should be proportional to the table value.

PID Tuning Run #	$T_i$	$T_d$	$K_p$	Time	Error	Curve1	Curves2	Straight1	Straight2
1	2	0.25	0.84	78	216	average	overshot	1st straight good	overcorrect slightly
2	2.5	0.25	0.84	64	78	good	good	average	very smooth
3	3	0.25	0.84	65	92	not as good as prev.	average	wobbles	not as smooth
4	2.75	0.25	0.84	72	170	average	average	wobbles	wobbles
5	2.75	0.25	0.84	65	104	average	average	wobbles	average
6	2.75	0.25	0.84	67	109	average	average	slight wobble	average
7	2.625	0.25	0.84	65	98	average	average	slight wobble	very smooth
8	2.25	0.25	0.84	73	132	average	overshot	wobbles	finished poorly
9	2.375	0.25	0.84	68	113	good	overshot	average	average
10	2.5	0.25	0.84	77	?200	average	average	average	finished poorly
11	2.5	0.25	0.84	67	100	average	average	average	average
12	2.5	0.25	0.84	66	94	average	average	average	average
13	2.5	0.5	0.84	84	69	average	average	wobbles	wobbles
14	2.5	0.375	0.84	85	112	not as good	not as good	average	wobbles
15	2.5	0.375	0.84	77	88	average	not as good	average	slight wobbles
16	2.5	0.75	0.84	93	136	average	average	huge wobble	huge wobble
17	2.5	0.2	0.84	64	109	smooth, but loose	smooth, but loose	good	average
18	2.5	0.3	0.84	68	70	good	big corrections	wobbles	average
19	2.5	0.3	0.84	68	77	good	big corrections	slight wobble	average
20	2.5	0.275	0.84	74	170	average	big corrections	slight wobble	slight wobbles
21	2.6	0.3	0.84	74	114	average	average	average	wobbles
22	2.53	0.3	0.84	73	103	average	average	average	average
23	2.5	0.3	0.84	67	84	good	good	good	good

Experimental distance measured and final PID parameter values are as follows:

$$K_i = 1/T_i = 1/3.125 = 0.32$$

$$K_d = T_d = 0.20$$

$$K_p = 0.75$$

## 7. References

1. ECE456/556 Project Description
2. ECE456/556 Lecture Slides
3. Line Following – A Guide to Using Sensors  
<http://www.wrighthobbies.net/guides/linefollower.htm>
4. Path Tracking Control of Mobile Robots Using a Quadratic Curve - IEEE
5. Gain Adaptation of Networked DC Motor Controllers Based on QOS Variations – IEEE
6. ECE456/592R Ultrasonic Ranger Finder
7. Sonar Drivers for the Handy Board  
<http://handyboard.com/software/sonar.html>