Service Animal GPS

Design Review

Chris Stoddard, Harrison Rose, Richard Lew

TA: Tom Galvin

Team # 27

Objectives:

The goal of this project is to create a device for service animals to wear that would enable them to guide people to various locations. The device will utilize a GPS receiver to determine the location of the animal and person, and will use two vibrating motors to give directions. The device will also be equipped with safety features such an ultrasonic range finder to detect curbs, so that the user can be prevented from walking into a busy intersection. The device will support pre-programmed and user recordable routes.

Benefits:

- Allows service animals to direct people to locations
- Gives disabled people the freedom to explore on their own
- Provides greater safety to the disabled
- Faster and more reliable than other methods

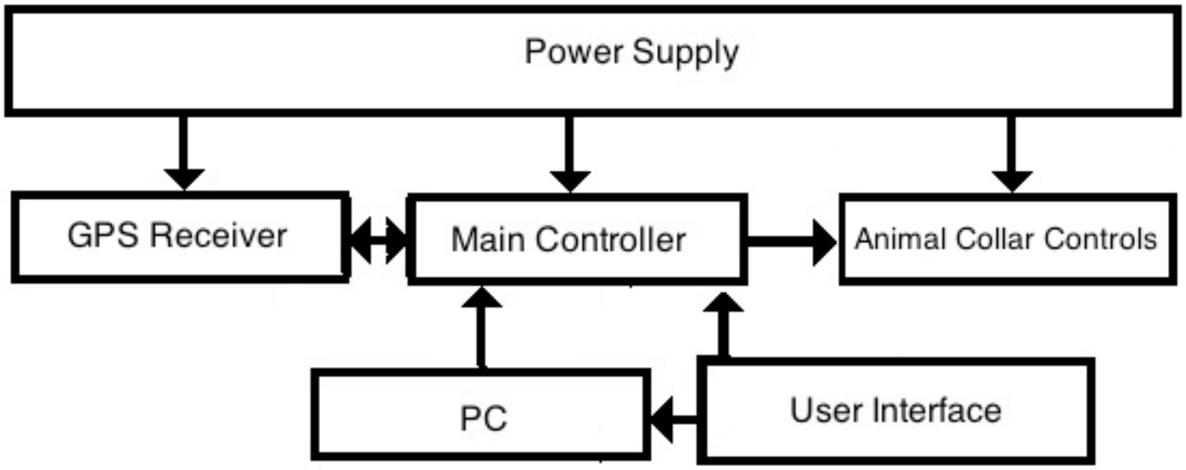
Features:

- Turn by turn directions for up to five locations can be stored
- Load directions from GPS software or record route
- Two vibrating motors for executing route instruction
- Adjustable collar

Performance Requirements:

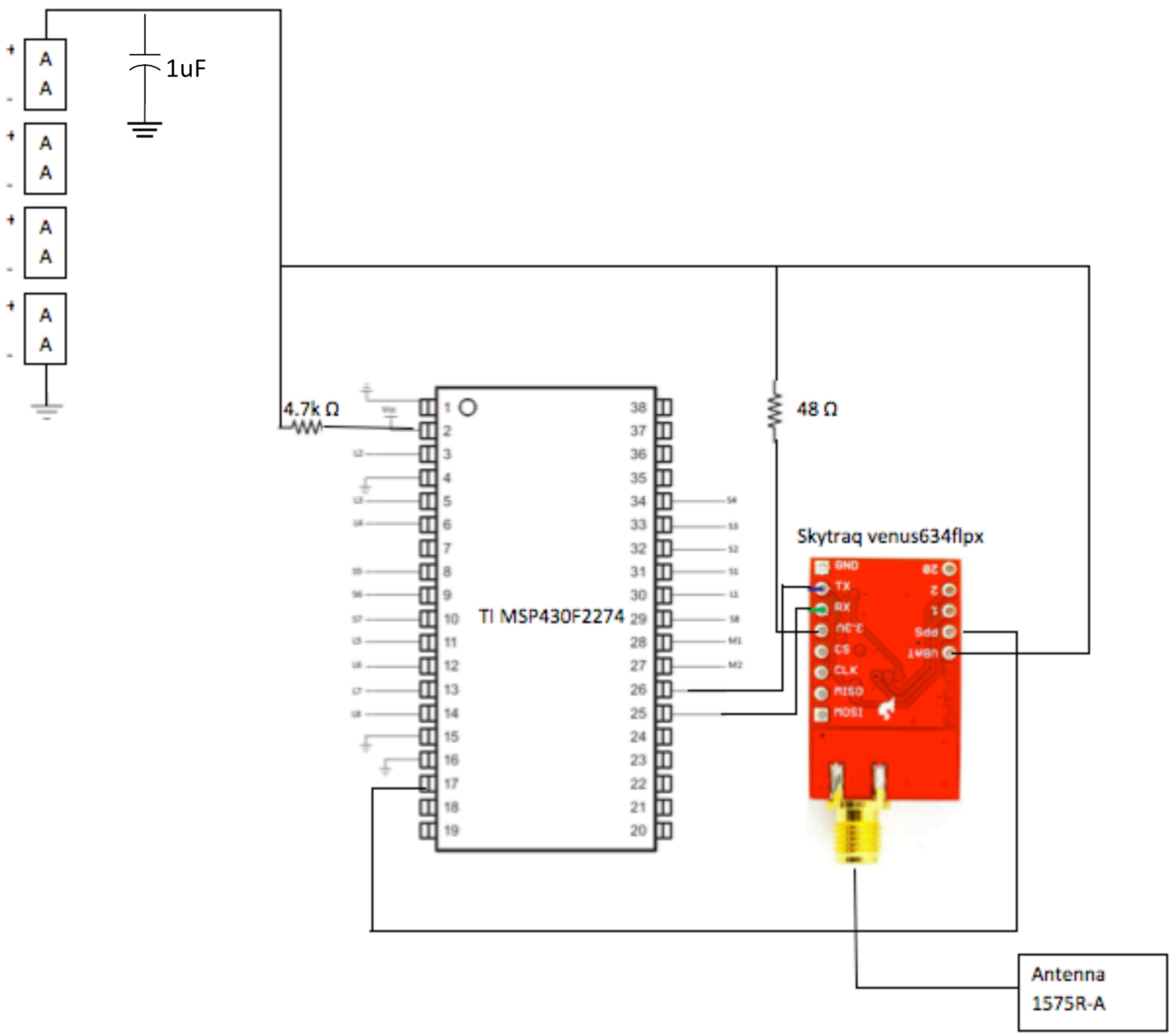
- Location accuracy within five meters
- Two and half meter precision turning
- Control the motors together and independently to give instructions
- Record routes so that they can be navigated without making errors
- GPS chip communicates with microcontroller using UART to send NMEA messages
- Can generate return route at any point during route
- Full charged battery yields a minimum of two hours of device use

Block Diagram:

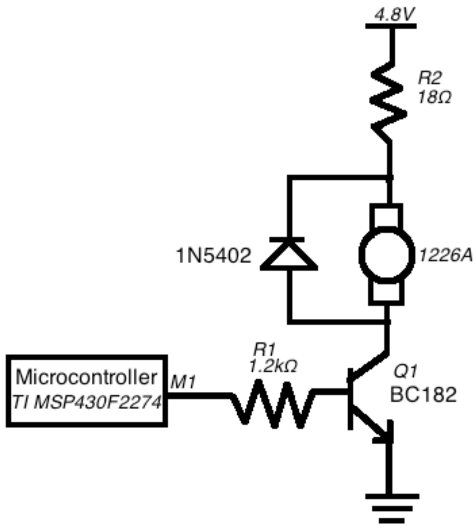


Schematics:

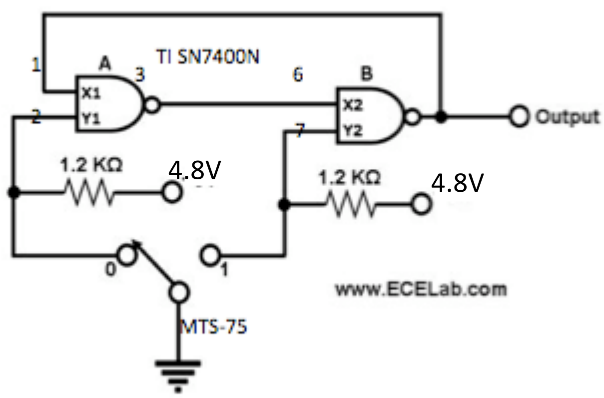
GPS and Microcontroller

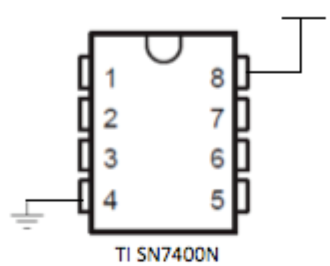


Motors (M1 and M2)

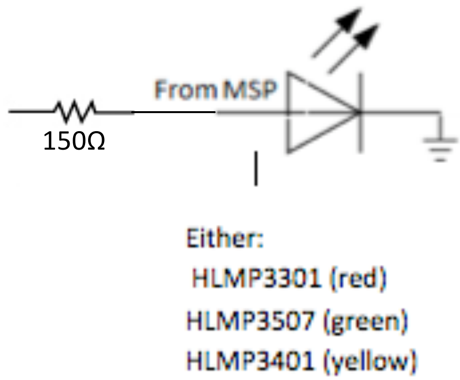


Switches (S1-S8)





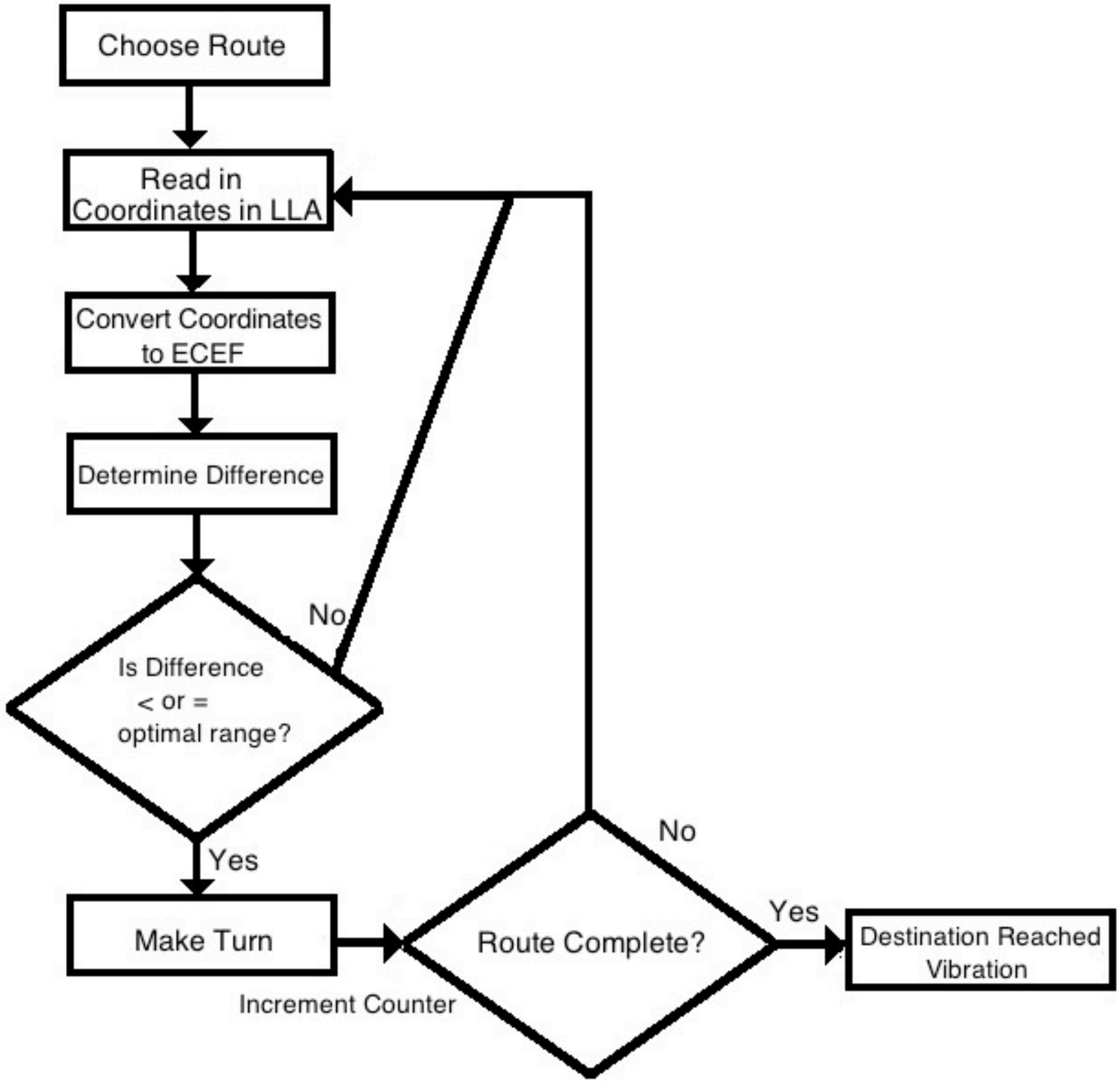
LEDs (L1-L8)



Descriptions:

Module	Description
Power Supply	Responsible for providing power to all necessary elements of the unit.
GPS Receiver	Will interface with satellites and the microcontroller to output the user's position.
Main Controller	Responsible processing data. It will read in data from the GPS receiver; which in turn will be processed so that output directional data can be sent to the two motors. The main controller also contains the memory of the unit, which will store the preprogrammed and user recorded routes.
PC	The user will employ his/her personal computer to configure the unit so that pertinent data such as routes, waypoints, and points of interest can be added to the device.
Animal Control Collar	Control collar will consist of motors that will be used to create vibrations in order to guide the service animal. The main controller will send the directional data into the motors. The physical collar will be adjustable so that it can comfortably fit a variety of different size and breed service animals.
User Interface	The user will be able to interact with the unit directly (so as to select destinations from memory or record a route) and with his/her personal computer to program routes onto the unit.

Software Interface:



Sample Code (Simulation):

```
%% Generate a vector that list all of the turn waypoints on the route
waypoints = [0 0; 100 0; 100 100; 200 100; 200 20; 340 20; 340 160];


%% Generate input vector, based on average walking speed and walking a
% direct path between each turn
```

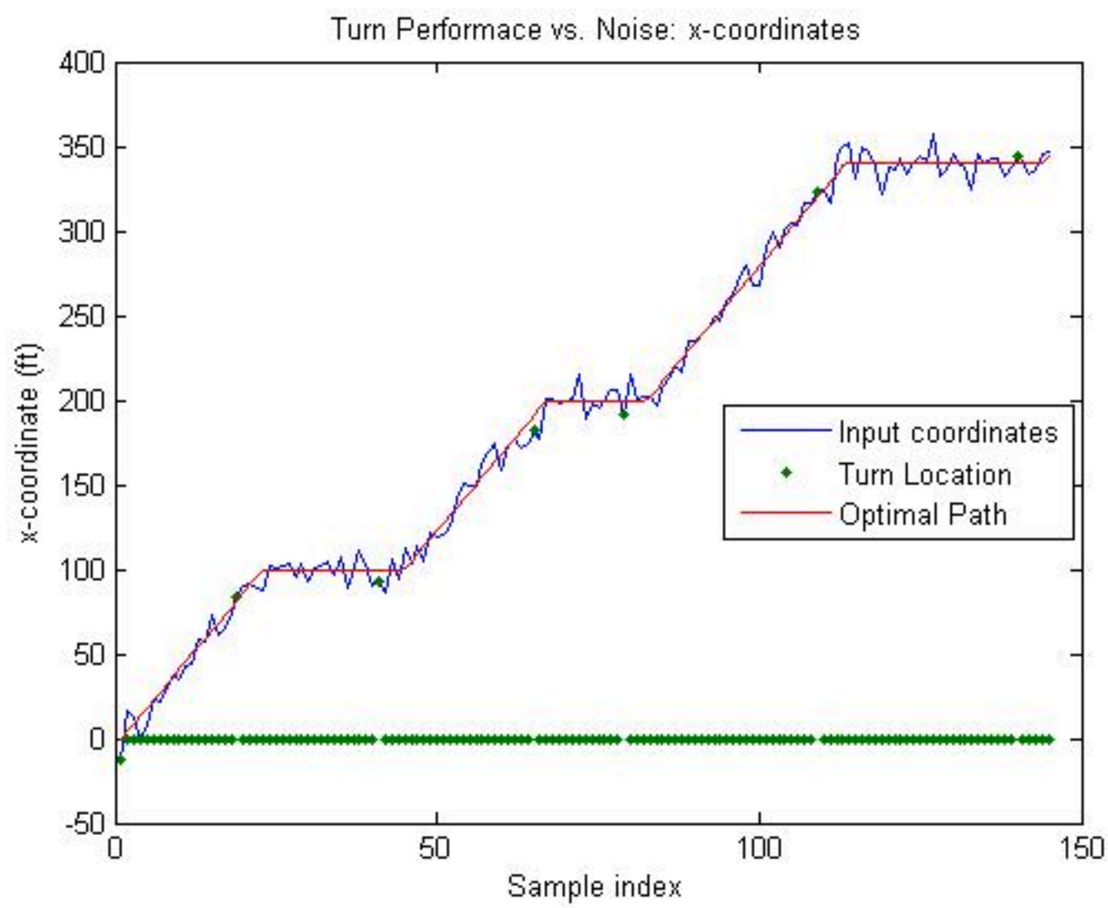
```
step = 4.546; % in ft/s
x_cor = zeros(1,145);
y_cor = zeros(1,145);
cnt = 2;

while x_cor(cnt-1) < (100-step)
   x_cor(cnt) = x_cor(cnt-1) + step;
   y_cor(cnt) = y_cor(cnt-1);
   cnt = cnt+1;
end

increment = 100 - x_cor(cnt-1);

x_cor(cnt) = x_cor(cnt-1) + increment;
y_cor(cnt) = step-increment;
cnt = cnt+1;


while y_cor(cnt-1) < 100-step
   x_cor(cnt) = x_cor(cnt-1);
   y_cor(cnt) = y_cor(cnt-1) + step;
   cnt = cnt+1;
end

increment = 100 - y_cor(cnt-1);

y_cor(cnt) = y_cor(cnt-1) + increment;
x_cor(cnt) = x_cor(cnt-1) + step-increment;
cnt = cnt+1;

while x_cor(cnt-1) < 200-step
   x_cor(cnt) = x_cor(cnt-1)+step;
   y_cor(cnt) = y_cor(cnt-1);
   cnt = cnt+1;
end

increment = 200 - x_cor(cnt-1);

x_cor(cnt) = x_cor(cnt-1) + increment;
y_cor(cnt) = y_cor(cnt-1) - step-increment;
cnt = cnt+1;


while y_cor(cnt-1) > 20+step
   x_cor(cnt) = x_cor(cnt-1);
   y_cor(cnt) = y_cor(cnt-1) - step;
```

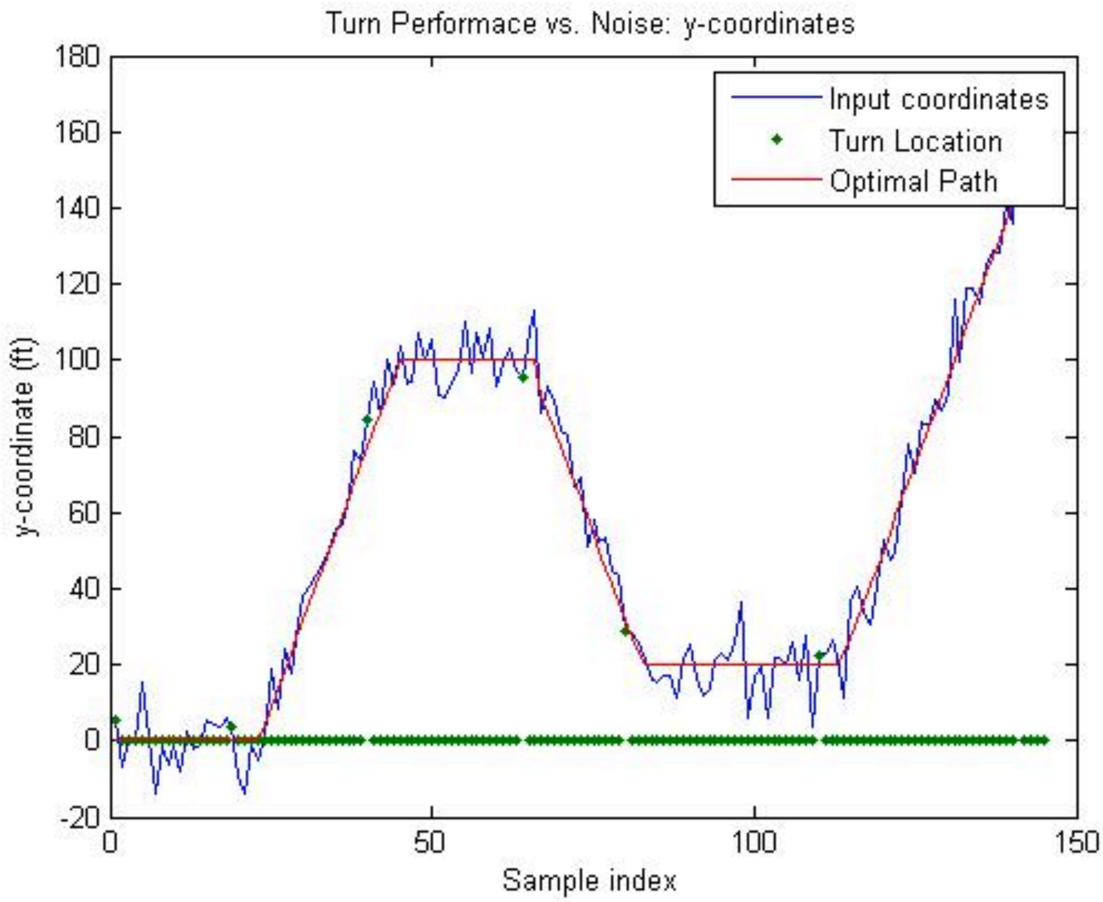
```
    cnt = cnt+1;
end

increment = y_cor(cnt-1)-20;

y_cor(cnt) = y_cor(cnt-1) - increment;
x_cor(cnt) = x_cor(cnt-1) + step-increment;
cnt = cnt+1;


while x_cor(cnt-1) < 340-step
    x_cor(cnt) = x_cor(cnt-1)+step;
    y_cor(cnt) = y_cor(cnt-1);
    cnt = cnt+1;
end

increment = 340 - x_cor(cnt-1);

x_cor(cnt) = x_cor(cnt-1) + increment;
y_cor(cnt) = y_cor(cnt-1) + step-increment;
cnt = cnt+1;


while y_cor(cnt-1) < 160-step
    x_cor(cnt) = x_cor(cnt-1);
    y_cor(cnt) = y_cor(cnt-1) + step;
    cnt = cnt+1;
end

increment = 160 - y_cor(cnt-1);

y_cor(cnt) = y_cor(cnt-1) + increment;
x_cor(cnt) = x_cor(cnt-1) + step-increment;
cnt = cnt+1;

%% Add noise to route data using normal distribution in m, then converted to feet.

x_cor_n = zeros(1,145);
y_cor_n = zeros(1,145);
dist_ave = 0;

for i = 1:1000;
    for i = 1:length(x_cor)
        x_cor_n(i) = x_cor(i) + 3.2808*normrnd(0,2);
        y_cor_n(i) = y_cor(i) + 3.2808*normrnd(0,2);
    end
```

```
    turn_range = 3.8208*5;

    turns = zeros(3,7);
    turn_dist = zeros(1,7);
    turn_cnt = 1;
    waypt_cnt = 1;


    % work through each data point and check if there should be a turn
    for x = 1:length(x_cor)
        if waypt_cnt == 8
            break
        end
        dist = sqrt((waypoints(waypt_cnt, 1)-x_cor_n(x))^2 + (waypoints(waypt_cnt,2)-
y_cor_n(x))^2);
        if dist < turn_range
            turn_dist(turn_cnt) = dist;
            turns(:,turn_cnt) = [x_cor_n(x) ; y_cor_n(x) ; x];
            turn_cnt = turn_cnt+1;
            waypt_cnt = waypt_cnt+1;
        end
    end

    dist_ave = dist_ave + sum(turn_dist)/length(turn_dist)/3.808;

end

dist_ave = dist_ave/1000

turn_cnt = 1;
turn_vect_x = zeros(1,145);
turn_vect_y = zeros(1,145);

for i = 1:145
    if turn_cnt == 8
        break
    end
    if turns(3, turn_cnt) == i
        turn_vect_x(i) = turns(1,turn_cnt);
        turn_vect_y(i) = turns(2,turn_cnt);
        turn_cnt = turn_cnt+1;
    else
        turn_vect_x(i) = 0;
        turn_vect_y(i) = 0;
    end
```

```
end

x = 1:145;
%plot(x,x_cor_n,x,turn_vect_x,'.', x, x_cor);
%xlabel('Sample index'), ylabel('x-coordinate (ft)'), title('Turn Performace vs. Noise: x-coordinates'), legend('Input coordinates', 'Turn Location', 'Optimal Path');


plot(x,y_cor_n,x,turn_vect_y,'.', x, y_cor);
xlabel('Sample index'), ylabel('y-coordinate (ft)'), title('Turn Performace vs. Noise: y-coordinates'), legend('Input coordinates', 'Turn Location', 'Optimal Path');
```

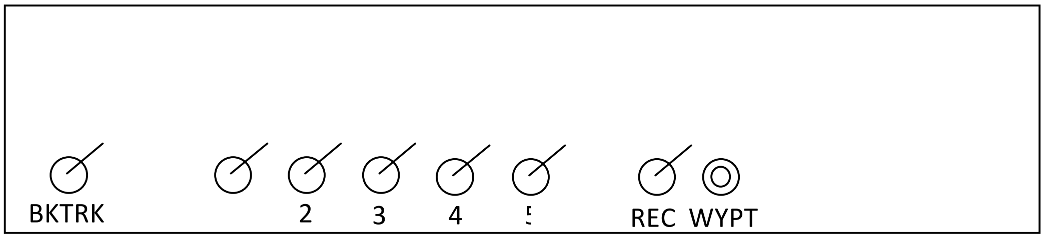
Results:





User Interface:

The user interface for our devices is as follows. There are eight switches to control the device, and eight LEDs that we will be using for debugging. To select a route, the user switches one of the route switches 1-5. This instructs the device that you want to be guided to destination indicated by the switch, one being far left and five being far right. After the destination is reached, the switch must be lowered before a new route can be activated. If, during the course of the route, the user decides they would like to go back to their original location, the back track switch can be toggled. This instructs the device to reverse the routing information and direct the user back home. With all of the switches in the off position, if backtrack is switched on and then a route is selected, the device will guide your along the route backwards (starting at the destination and terminating at the start point). To record routes, the user will flip the record switch, and then flip the route that they wish to record. As the user walks through the route, at every waypoint (turn) the user will depress the waypoint button to record that location as a waypoint. When the user has completed the route, turning the record switch off will save the current location as the end of the route. Below is an idea of what the user panel will look like on the device.



When this interface was designed, care was taken to ensure that users with vision impairments would be able to operate the device. To achieve this, tactile switches were used for almost all of the functions. This is nice because it allows the user to feel whether the switch is on or off, unlike simple push buttons or LCD menu based inputs. To input waypoints, a push button was used instead of a switch because on – off doesn't make sense for this function.

To load routes into the device, the user will need to create a text file the latitude, longitude, and direction of each waypoint along the route. The latitude and longitude must be in the format DDDMM.MMM, where D is a degree digit, and M is a minute digit. The direction, either left or right, should be stored as either 'l' or 'r'. For example, to create a waypoint at 79° 40.325' N, 52° 25.154' W, Left, the user should type on a single line 07940.325,05225.154,l. Once each waypoint is listed in a text file, the data will be loaded when the code is compiled and loaded onto the board.

Power Analysis:

The GPS unit and the microcontroller use 30 milliAmp hours (mAh) combined and so for two hours, it would take 60 [mAh]. The motors use 80 [mA/h] when they are active. If we are to vibrate the motors for three seconds for every turn, we need to calculate the amount of current used for during each turn.

Motor turn:

80 [mAh] / 3600 [sec] * 3 [sec] = 0.067 [mAh/turn]

We also want to vibrate both motors for 3 seconds when we want the person to stop. The person only needs to stop when the destination has been reached, therefore, only one stop will be needed.

Motor Stop:

80 [mAh] * 2 / 3600 [sec] * 3 [sec] = 0.133 [mAh]

The average walking speed of a person is 4.5466 feet per second. Looking at Google Maps, we determined that the average distance of one block is 500 feet. If the person makes an average of one turn every two blocks and waits at stoplights for an average of ten seconds, the person would make a turn every four minutes. This would result in a total of 30 turns.

Average:

500 [ft] * 2 / 4.5466 [ft/s] = 219.94 [sec] + 20 [sec] = 239.94 [sec]

0.067 [cmAh/turn] * 30 [turns] = 2.01 [mAh]

Total = 2.01 [mAh] + 0.133 [mAh] + 60 [mAh] = 62.143 [mAh]

For 2500 [mAh] batteries the lifetime would be

Lifetime = 2500 [mAh] / 62.143 [mAh] = 40.23 two hour trips

The worse case scenario would be if the person were to make a turn at every block. If we include the average of ten seconds wait at stoplights, the person would make one turn every two minutes. This would result in a total of 60 turns.

Worst Case:

500 [ft] / 4.5466 [ft/s] = 109.97 [sec] + 10 [sec] = 119.97 [sec]

0.067 [mAh/turn] * 60 [turns] = 4.02 [mAh]

Total = 4.02 [mAh] + 0.133 [mAh] + 60 [mAh] = 64.153 [mAh]

Lifetime = 2500 [mAh] / 64.153 [mAh] = 38.97 two hour trips

Test Plan and Verification:

Module	Requirement	Test	Verification
GPS unit	GPS must report location accurate to within five meters.	Choose ten locations outside; five in open areas and five close to buildings. The error in location should no more than five meters.	I. Using Google Maps, record the coordinates of five locations near buildings and five open-areas. Take the device to these places and measure the position using the GPS receiver. Compare the measured coordinates with the actual coordinates, and ensure that they differ by no more than five meters.
	The GPS unit must send standard NMEA messages via UART to the microcontroller.	Connect oscilloscope to the output of the UART pins on the GPS receiver, and observe the output.	Verify that the GPS receiver is sending NMEA messages on the UART transmit pin. Also verify that it starts sending one message every second.

Motors	Control the motors together and independently to give instructions	Have the microcontroller run through each possible vibration command combination to demonstrate the control.	I. Create a test program that cycles each motor independently using .25, .5, and 1 second pulses with 50% duty cycle. Use an oscilloscope to view the PWM signal from the transistor to ensure that the period and duty cycle of each wave is correct. III. Connect the motors directly to the power source and verify that the motors run.
	The microprocessor must be able to output .25, .5, and 1 second pulses with 50% duty cycle independently on two outputs	Connect an oscilloscope to each motor output of the microprocessor and view the signals.	Using the oscilloscope, view the output waveforms directly at the output of the microprocessor. Verify that the PWM waves are correct as listed in the requirement.
	The motors must run continuously when connected directly to a power source.	Connect each of the vibrating motors directly to the power supply in the lab.	Verify that the motors continuously vibrate when provided with power.

Microcontroller (Memory)	Record routes so they can be successfully navigated	Hit the record button, walk a route through the city, then hit stop. Return to the original location and select that route, and verify that the directions sent to the motors are correct.	Using the record feature, walk a route and save each turn by pressing the waypoint button. Look at the recorded waypoints in memory using the PC, and verify the coordinates using Google maps.
	Microcontroller must be able to store generated data to memory	Write a test program that writes a known sequence to memory.	Using the programming interface, verify that the sequence written to memory is found in the correct location.
Microcontroller	The microcontroller must receive NMEA messages sent to it by the GPS receiver. UART interface.	Develop a testing function on the microprocessor to implement only the UART interface and store the data to memory.	Run the test function and then view the stored data in the microprocessor memory. Ensure that the data is in the correct format according to the NMEA standard defined in the Skytraq datasheet.
	The device must not indicate a turn more than 2.5 m away from the turn location	Choose ten turns outside; five in open areas and five close to buildings. The device should not initiate a vibration for a turn further than a two and half meter radius prior	Walk towards each of the turns, and stop as soon as the turn command is issued. Measure the distance between where the person has stopped and the actual turn location. Verify that this distance is less than 2.5m.

		around the turn.	
	Can generate return route at any point during route.	Start walking a route, then hit the backtrack switch. Verify that the directions are correct on the return route.	When walking the route backwards, verify that the directions are correct by verifying that the turns on the return route are generated at the same locations as on the original route.
Power Supply	Full charged battery yields a minimum of two hours of device use.	Fully power on the device, and record the current and voltage being drawn from the battery pack with the motors off.	While the device is fully powered, connect a digital multimeter and verify that the output is 4.8V and 60 mA.
	Full charged battery yields a minimum of two hours of device use.	Fully power on the device, and configure the microprocessor to run the motors continuously. Record the current and voltage output from the battery pack.	Connect a digital multimeter to the battery pack and measure the change in current between the case when the motors are running and when the motors are off. Verify that the difference in current draw is 6 mA.

Curb Detection:

Originally it was intended for the device to have a curb detection feature implemented using an ultrasonic sensor. After further consideration, it was evident that the data we could collect with the sensor was not going to be enough. Using the average walking speed of a human and the maximum sampling frequency of the sensor we determined the maximum sampling frequency in linear feet.

3.1 [miles/hr] x 5280 [ft/mile] x 1/3600 [hr/s] x 1/2.28 [s/sample] = 1.99 [ft/sample]

Curb detection with such a large distance between samples is impractical. The sampling points were so far apart that there was no reliable way to detect curbs. Additionally, the sensor can only be tilted a maximum of 15°.

$\tan^{-1}(X/Y) = 15° \rightarrow Y = 2$ [ft]/ $\tan^{-1}(15°) = 7.4641$ [ft]

Thus, the sensor would have to be seven feet above the ground in order to see two feet in front of itself. We then found that the average height of a dog was 60 to 65 centimeters.

65 [inches]/2.54 [cm/inch] = 25.59 [inches]→ 25.59*tan (15°) = 6.8585 [inches] in front of the sensor.

Based on the average height of a dog, the sensor would have only been able to see seven inches in front of the dog, thus putting the dog and its owner at an extreme risk. Therefore, given these unrealistic parameters including a curb detection sensor on our device would be useless and unethical.

Tolerance Analysis:

The most important aspect of this device is the position and direction accuracy. The accuracy of a GPS receiver is depends on the various types of interference the signal undergoes on its path between the receiver and the satellite. The six most common forms of signal degradation include:

1) ***Ephemeris data***-- Errors in the transmitted location of the satellite (3 meters on average)
2) ***Satellite clock***--Errors in the transmitted clock of the satellite (2 meters on average)
3) ***Ionosphere***--Errors in the corrections of pseudorange due to ionospheric effects (5 meters on average)

4) ***Troposphere***--Errors in the corrections of pseudorange due to tropospheric effects (1 meters on average)
5) ***Multipath***--Errors caused by reflected signals entering the receiver antenna (1 meter on average)
6) ***Receiver***—Errors associated with the receiver such as noise, software, and biases. (1 meter on average)

In this case with averages values, there is an error of thirteen meters. The sum of the average errors associated with each facet of hindrance yields an overall position error of about thirteen meters. Therefore, we will ensure that our device will have an average positional accuracy of thirteen meters. To test this tolerance we will measure the user's physical distance from the GPS receiver's waypoints. For instance, if the receiver indicates that the destination has been reached, we will then measure how far off the user is from that destination. This tolerance test will be conducted during each of the "GPS Unit", "Route Recording", and "Memory" testing procedures.

Cost:

Item	**Make**	**Quantity**	**Cost per unit**	**Total Cost**	**Status**
GPS Chip	Skytraq venus634flpx	1	$100	$100	Posses
GPS antenna	1575R-A	1	$5	$5	Posses
Motor	1226A	2	$1.95	$3.90	Ordered
LEDs	HLMP3301 (red) HLMP3507	8	$0.50	$4	Readily-available

	(green) HLMP3401 (yellow)				
Switches	MTS-75	8	$0.85	$6.80	Need to order
Circuit board		1	$5	$5	Posses
Microcontroller	TI MSP430F2274	1	$20	$20	Need to order. Have development chip.
Circuit components				$10	Readily-available
Batteries	Duracell 1.2V Rechargeable AA			$10	Posses
Collar		1	$10	$10	Need to order
Total Cost				$174.70	

Labor:

($60.00/hr) x (2.5) x (150 hours) = $22,500

Total Cost:

Parts + Labor = $210 + $22,500 = <u>$22,614.70</u>

Schedule:

Week	Task	Member
2/6	Research motor implementation and power supply requirements	Richard
	Research sensor integration and vibration systems	Chris
	Research direction implementation and GPS receiver capabilities	Harrison
2/13	Acquire Parts	Chris
	Design Review	Harrison
2/20	Power on and program microcontroller	Chris
	Power up GPS and verify location	Harrison
	Plan motor/microcontroller interface	Richard
2/27	Control motors using microcontroller	Richard
	Code vibration system	Chris
	Create preprogrammed routes using GPS	Harrison
3/5	Load vibration system onto microcontroller	Richard
	Load routes to GPS via PC	Harrison
	Interface GPS and microcontroller	Chris
3/19 (Spring Break)	Read data from rangefinder with microcontroller	Chris
	Start on interfacing microcontroller for route recording	Richard
	Write code for route recording	Harrison
3/26	Begin route recording test	Chris
	Write code for backtracking	Harrison
4/2	Implement backtracking onto microcontroller	Richard
	Finish route recording	Chris
4/9	Test and debug entire recorded route and backtrack	Harrison
4/16	Full unit test and debug	Richard
4/23	Demo	Chris
4/30	Presentation	Harrison
	Final Paper	Chris

Ethical Issues:

There are several ethical issues to consider when designing and implementing our idea. Most notably, it is imperative that all of our claims and feature precise and accurate. The intention of our device is to aid visually impaired people, and therefore the slightest miscalculation in the navigation solution or misdirection could result in serious injury or death to the service animal and/or the user. For instance, one-meter difference could cause someone to walk into oncoming traffic. This realization was greatly influenced by IEEE code of ethics bylaws 1, 3, 8, and 9. Additionally, we originally wanted to implement a curb detection sensor, however, upon calculating the minimum effective range of the sensor, we discovered that it would only be able to detect for a distance of six inches ahead of the service animal. We felt that this was not nearly a large enough range and would put the user and service animal in danger when approaching an intersection. The decision to ultimately exclude a sensor was made to adhere to IEEE code of ethics bylaws 1 and 9.

There are other less conventional ethical issues associated with our unit. For instance, someone could preprogram a route, attach an explosive, and use the service animal as a weapon. Using our device for such purposes would be in direct conflict of IEEE code of ethics by laws 1, 5, and 9.

References:

"Blind people safety project". Faith Degirmenci, Abdessettar Ibourki, Morad Oumina. Proposal, Spring 2006

"GPS Tracking Device with DGPS". Alex Stezskal, Algirdas Navickas, Vivek Thyagarajan. Proposal, Spring 2010

Misra, Pratap, and Per Enge. *Global Positioning System: Signals, Measurements, and Performance*. Lincoln, MA: Ganga-Jamuna, 2011. Print.

"Sam Wormley's GPS Errors & Estimating Your Receiver's Accuracy." Educational Observatory Institute. Web. 08 Feb. 2012. <http://edu-observatory.org/gps/gps_accuracy.html>.

Skytraq datasheet for Skytraq Venus634FLPx. <http://www.sparkfun.com/datasheets/GPS/Modules/Skytraq-Venus634FLPx_DS_v051.pdf>

"Switch Debouncing (Bounce-free Switches) Using NAND Gates - Www.ECELab.com." Electronics and Communications Engineering. Web. 21 Feb. 2012. <http://www.ecelab.com/switch-debounce-NAND.htm>.

TI datasheet for MSP430F2274. < http://www.ti.com/product/msp430f2274>.

TI datasheet for SN7400N. <http://www.ti.com/lit/ds/symlink/sn7400.pdf>.