µBot

Written Report

University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Laboratory

Instructors: Dr. A. Antonio Arroyo
Dr. Eric Schwartz

TAs: Mike Pridgen
Adam Barnett
Sara Keen

Orlando Misas
4/22/08
UFID:8219-3161

Table of Contents

List of Figures

List of Tables

Abstract

The purpose of this project is to create an autonomous robot that is able to follow verbal commands from humans. The command alternatives will be stored in the robot and then compared against an input. The person controlling the robot will speak into a separate unit that will decode and translate the instruction. The unit will then send a short signal to the robot which will follow the command as long it is within "logic." For example the robot will ignore any instruction that goes against its sensor readings, in order to avoid foreseeable damages.

The microphone unit will use fast Fourier transform to translate voice input into frequency domain. The prevalent frequencies will then be compared to stored quantities to decode the instruction. Instead of sending large data packets, small instructions can be sent, which will reduce the time the robot has to sacrifice to interpret the command. The possible instructions will be consist of only one word and they will allow the user to cycle through the robot's behaviors. The robot will have two main behaviors, wall-following and obstacle avoidance, as well as an auxiliary behavior. The third mode will allow the user to give simple instructions to the robot, and the robot will respond if the environment permit's the command.

Executive Summary

The μBot is an autonomous agent that is able to respond to verbal commands from the user. The commands are used strictly to switch between behaviors not to command the robot step by step. The robot is able to perform obstacle avoidance, wall following and some basic behaviors to show functionality. The extra commands include moving forward , rotating right, rotating left and stopping; the first three of these is performed for a random amount of time between one and six seconds.

The system itself is separated into two independent blocks, the microphone unit and the platform. The microphone unit is responsible for acquiring and decoding of the audio signal. It also performs a Fourier transform on the audio data in order to match the analyzed signal with saved values. The unit then sends a simple command to the platform specifying which behavior to use.

The platform is a simple design with two wheels and a caster. Movement is achieved with two low voltage DC motors, which is controlled with a motor controller that is rated to provide the power required by the motors. The platform carries multiple IR and Sonar sensors to assist it with its different behaviors. There is also a LCD screen mounted on the platform to allow the debugging of the unit. The platform has two power supplies in order to provide low, 2.4 Volts of power to the motors and a considerably higher 5 Volts to the digital circuits.

Communication between the two units is done by sending a fifty percent duty cycle square wave from the microphone unit to the platform and depending on its period or frequency the platform identifies the command and changes the

behavior of the platform. Behaviors are programmed to work for the duration of one time step, not considering past values this allows easy switching between behaviors. The main program of the platform includes an infinite loop that checks the communication line and depending on a global value initiates one step of the desired behaviors.

Introduction

Objective

The objective of this project is to create an autonomous robot capable of following different behaviors of moderate difficulty. The robot must have the ability of accepting verbal commands and follow previously programmed instructions. The commands must be used to illustrate both complex and simple functions from changing behavior to changing the primary direction of movement. In order to provide autonomous functions the robot will be equipped with various sensors such as sonar and IR. The robot will also have to be able to override instructions depending on sensor readings. The greatest challenge will be successfully implementing the algorithm to distinguish different verbal commands.

Theory

The principal idea behind differentiating the commands is the Fourier transform. As Lathi describes it, the Fourier analysis is the frequency domain representation of a signal in the time domain. Furthermore FFT (fast Fourier transform) will be used to carry out the conversion in a reasonable time. The FFT is a variation of DFT (discrete Fourier transform) developed by J.W. Cooley and John Tukey in 1965. Just as the DFT, it allows the calculation of a frequency domain representation of a signal from a discrete time domain form but it does so using less computations. In the worst case the DFT performs N^2 computations, or $O(N^2)$, whereas the FFT will perform the task in N log N computations or O(N log N). FFT breaks down a DFT using recursion and uses the divide-and-conquer algorithm to solve the problem.

Content

The paper will present the system design of the platform and the microphone unit. It will discuss the physical design of the platform, at this point the actuation will be explained in detail. Next are the sensor description and its particular uses. The “Sensors” section also includes a detailed view of how the microphone unit works. The paper then discusses the programmed behaviors. The experimental data is explained after the behaviors. And finally the paper presents the conclusions and appendices.

Integrated System

The robot will have two components the microphone unit which will serve to decode verbal instructions and the platform which will serve as an autonomous robot. Each unit has a microcontroller and they are connected via RF link. Two way communication through the RF link is not essential because I will assume that the user is in range to see visual signals coming from the platform.

The microphone unit is composed of a microphone, a filter, an amplifier, the microcontroller and a transmitter. The microphone receives voice input and converts it into an electrical wave which then goes through a band pass filter which filters frequencies below 300 Hz and above 3 kHz which is approximately the range of human voice. The signal is amplified after leaving the filter in order to produce a more reliable set of A/D conversions. The microcontroller then samples and processes the incoming signal and transmits an instruction via the RF transmitter.

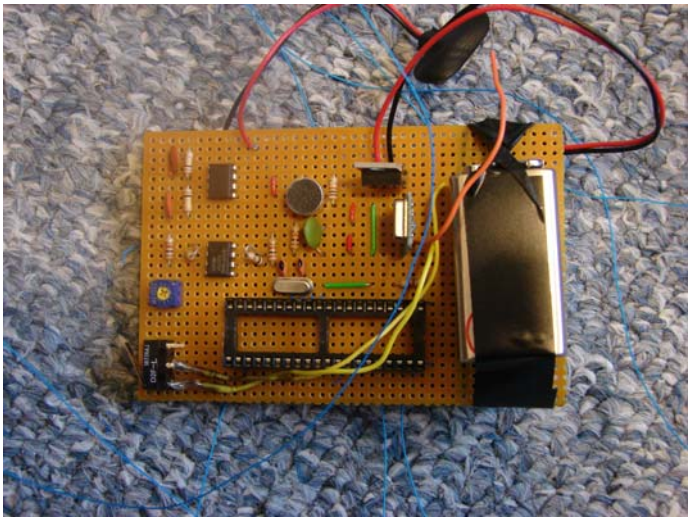

Figure 1: Microphone Unit

The following is a block diagram of the platform.

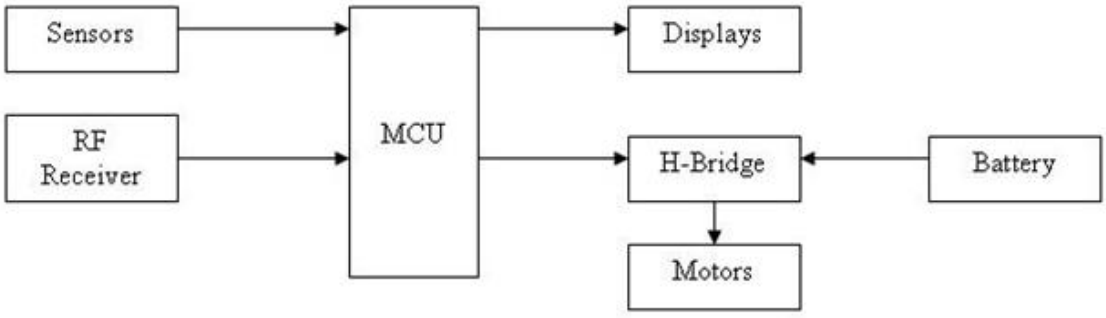

Figure 2: Block Diagram for Robot

The platform is composed of a receiver, multiple sensors, displays, a motor controller, a motor and a microcontroller. The robot itself will be controlled through the microcontroller which will send signals to the motor controller in order to control the motors. The microcontroller will respond the sensor readings to follow set behaviors. The microcontroller will also use a LCD along with an array of LED's to warn the user about its current mode of operation, errors, and direction. The robot will also be connected to a receiver which will allow the user to switch between behaviors or between primary directions if the robot is in the correct mode of operation.

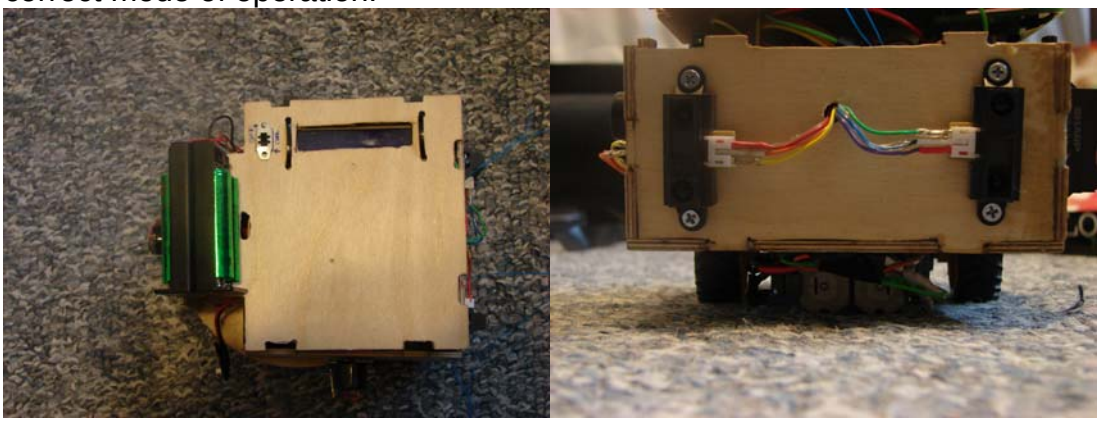

Figure 3: Front and Top views of platform

The following is the block diagram of the microphone unit. For a detailed explanation of each component refer to the "Appendix, Microphone Unit's Blocks" section. For overall description of the microphone unit refer to the "Experimental Layout and Results" section.

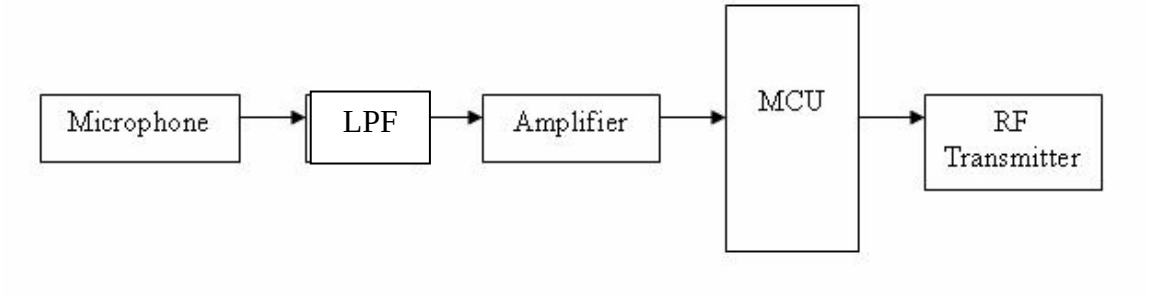

Figure 4: Block Diagram for Microphone Unit

Mobile Platform

In order to adhere to the objectives and a dual-motor setup using only two wheels and a ball caster the platform is circular with the motors and wheels on the bottom of the unit and all the electronics on the top portion of the robot. The motor controller is the Pololu Low-Voltage Dual Serial Motor Controller which is rated to work at low voltages as the name implies and it is also able to provide

the high current necessary to operate the DC motors. On paper this seemed as the ideal part, my experiences with this motor controller were less than pleasant. Although, the part is said to provide 5 Amperes of power to each motor, in reality the motor controller overheated at 15% of that and shut down until it cooled down. After installing various heat sinks the performance was increased to 25%. I recommend advising future students to not use low voltage DC motors because the controllers for them are scarce. If it is unavoidable I recommend building a custom motor controller rather than using the Pololu product.

Actuation

Two low-voltage DC motors are used to drive the two wheels of the robot. A ball caster is used to provide stability, while a motor controller is used to interface the motors to the microcontroller. Due to the high current requirements of DC motors the microcontroller cannot be connected directly to the motors. A low-voltage dual motor controller is used to avoid this problem.

Sensors

Sensors were chosen to permit the robot to be autonomous. Different sensors are implemented to allow detection of objects at various ranges. Sonar allows detection of objects up to 254 inches away but objects that are less than 6 inches away are not easy to detect. The IR sensors on the other hand do a little better at closer ranges, minimum of about 4 inches. Another advantage of the IR is that it takes measurements slightly faster than the sonar. The third alternative is a bump switch, which allows detection of objects at close ranges. In addition the microphone is considered as a special sensor. The microphone detects audio and outputs a wave, the wave's magnitude corresponds to the volume of the audio while its frequency and period matches that of the audio.

IR Sensor

Figure 5: Sharp IR

Vendor: www.sparkfun.com (Sparkfun Electronics)
Part Number: SEN-00242
Model: Infrared Proximity Sensor - Sharp GP2Y0A21YK

Scope

Uses light to detect objects at a distance. This sensor provides an analog output from Vcc-0.3V to 0.4V depending on the distance of the object. The

sensor detects said distance by timing how long it takes the emitted pulse to bounce back into the receiver. The Sharp IR has an effective range of 10 to 80 cm.

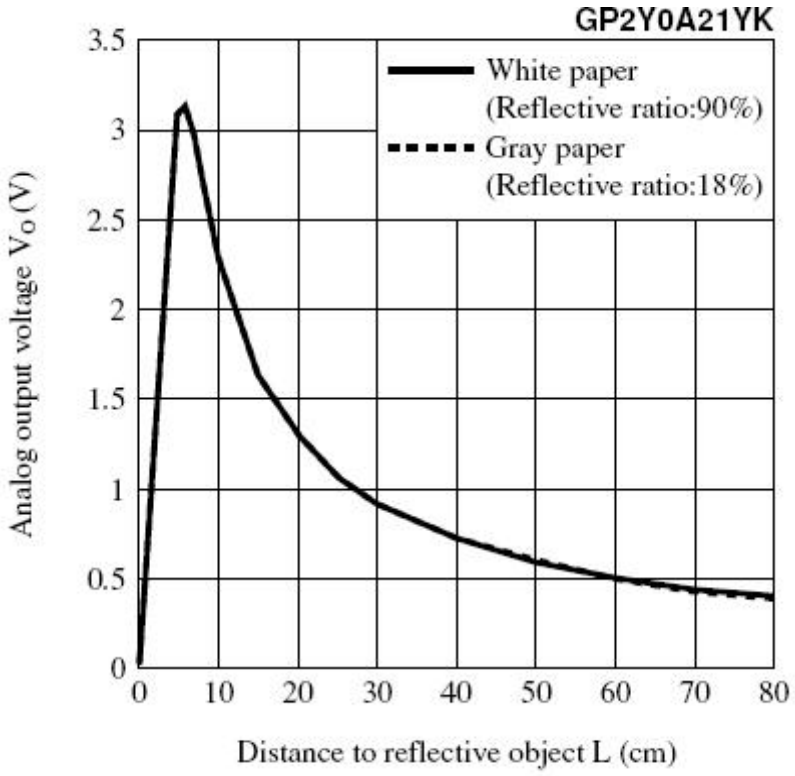

Figure 6: Analog Output Voltage vs. Distance to Reflective Object

Objective

The Sharp IR sensor provides a cheap alternative to detecting objects with a decent range. This sensor is also more resilient to ambient light which is usually a great concern with IR sensors. In the µBot I use IR sensors for obstacle avoidance and wall following.

Remarks

The Sharp IR sensor does not provide the best range. It also has a very small cone of detection which renders it useless when a small object has to be detected. On the other hand the small size of the robot compensates for the thin cone and the price of the sensor makes the lack in range more than bearable.

Sonar

Figure 7: Ultrasonic Range Finder

Vendor: www.sparkfun.com (Sparkfun Electronics)
Part Number: SEN-00639
Model: Infrared Ultrasonic Range Finder - Maxbotix LV-EZ1

Scope

The Ultrasonic Rangefinder use high frequency sound waves to detect objects at a distance. This sensor provides a multiple array of outputs but for the purposes of this project I will use the analog output which ranges from Vcc to ~0V depending on the distance of the object. The sensor detects said distance by timing how long it takes the emitted pulse to bounce back into the receiver. The Ultrasonic Rangefinder has an effective range of 6 to 254 inches.

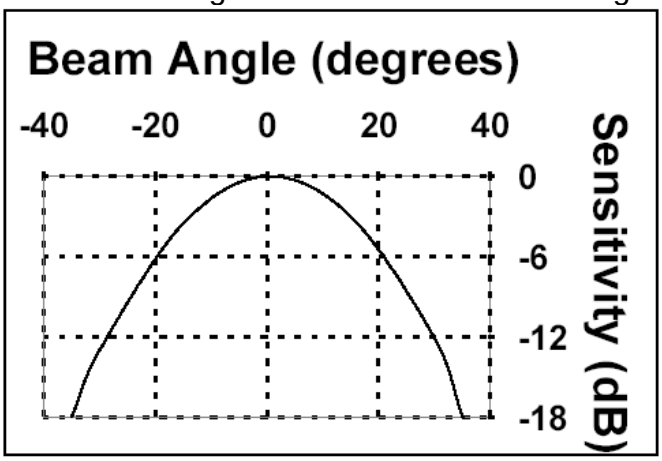

Figure 8: Beam Angle vs. Sensitivity of Sonar Sensor

Objective

The Ultrasonic Rangefinder is the perfect solution for finding objects at any range applicable to this project. In addition this sensor has a considerable detection cone thus making it a viable alternative for detection of obstacles in non-critical directions. The power consumption of this sensor is also considerably less than that of an IR sensor. In the µBot I will use Ultrasonic Rangefinder sensors to detect obstacles when moving left or reversing.

Remarks

The Ultrasonic Rangefinder sensor would be the perfect sensor for the µBot but considering its price tag (twice as much as the Sharp IR) only a limited number of them will be used in this project.

Special Sensor

Figure 9: Microphone

Scope

The special sensor is an audio sensor composed of a microphone, an amplifier, a digital signal processing microcontroller and a transmitter. The purpose of this sensor is to provide a command for the robot which may be a simple command such as turn right, left, go straight, stop or the command can set behaviors such as obstacle avoidance, wall following.

Objective

The microphone used is an Omnidirectional Electret Condenser Microphone with operation voltage of 4.5V DC (Model Number: CZ034A Series). The signal produced by the microphone is then routed through a passive low pass filter to an amplifier which provides a gain of 1000. The amplified signal is routed to the microcontroller which performs a FFT operation. The microcontroller calculates a value and compares the result to a number of stored constants, if it falls within the desired limits it becomes the command. The following table shows a typical result for each word and at what frequencies the peaks occurred.

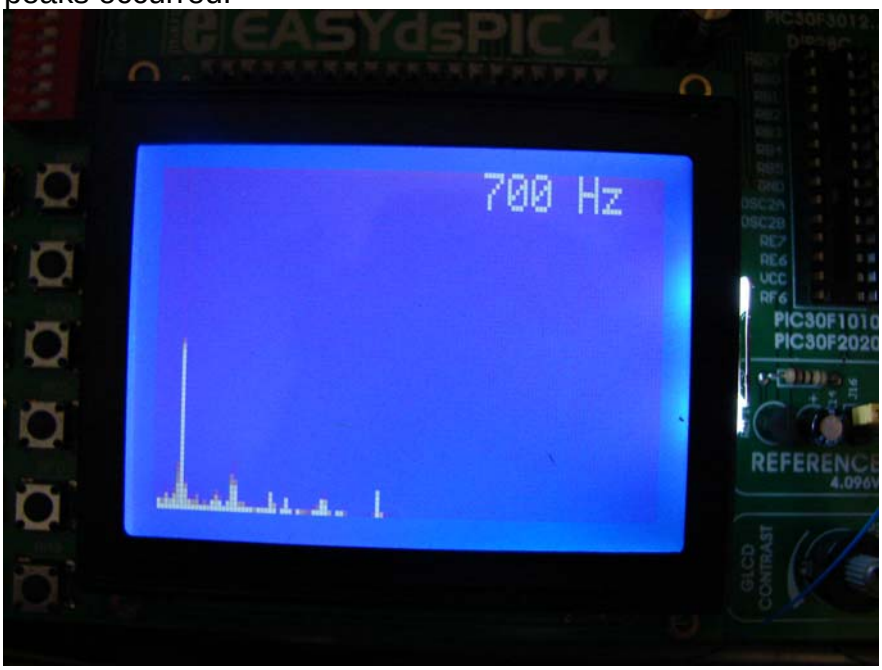

Figure 10: Example of FFT

Remarks

The microphone needs a great deal of amplification. For this project I used an amplifier with a gain of 1000, which produced remarkable results since I accidentally overshot the required peak to peak voltage it greatly increased the range of the unit. It practically became about six feet with no yelling required. As a consequence the results became less reliable at a close range which led me to change the comparison algorithm to distinguish peaks instead of magnitudes.

Behaviors

Obstacle Avoidance

The robot uses two IR sensors in front of the robot to determine which side to steer to. If one sensor is tripped the robot steers to the opposite direction. When both sensors in the front are blocked the back sensor (Sonar) is checked to determine if the robot can back up. If front and back sensors are blocked robot spins to the left for a random amount of time.

Wall Following

The robot uses two IR sensors on the right side of the robot in order to detect walls as well as the front sensors described in “Obstacle Avoidance”. If a wall is detected the robot checks front sensors to distinguish corners, the robot then positions itself to either face the wall or turn the corner. If only one IR senses the wall the robot adjusts so both sensors can sense the wall. If no wall is detected the robot moves forward until there is an object in front of it, the robot then turns to have side IR’s face obstacle.

Experimental Layout and Results

Microphone Data

The histogram below shows the distribution of where peaks occurred while the commands were repeated. Although this method does not take into account the order of the peaks it can be observed that almost all of the words follow a set pattern but with noticeable differences that would have allowed the distinction of the words. This method did yield a higher accuracy in the sense that words were identified more often (at about 60%-70%). But due to the nature of the data and the close proximity of the patterns words would also be misidentified more often, as opposed to finding no match.

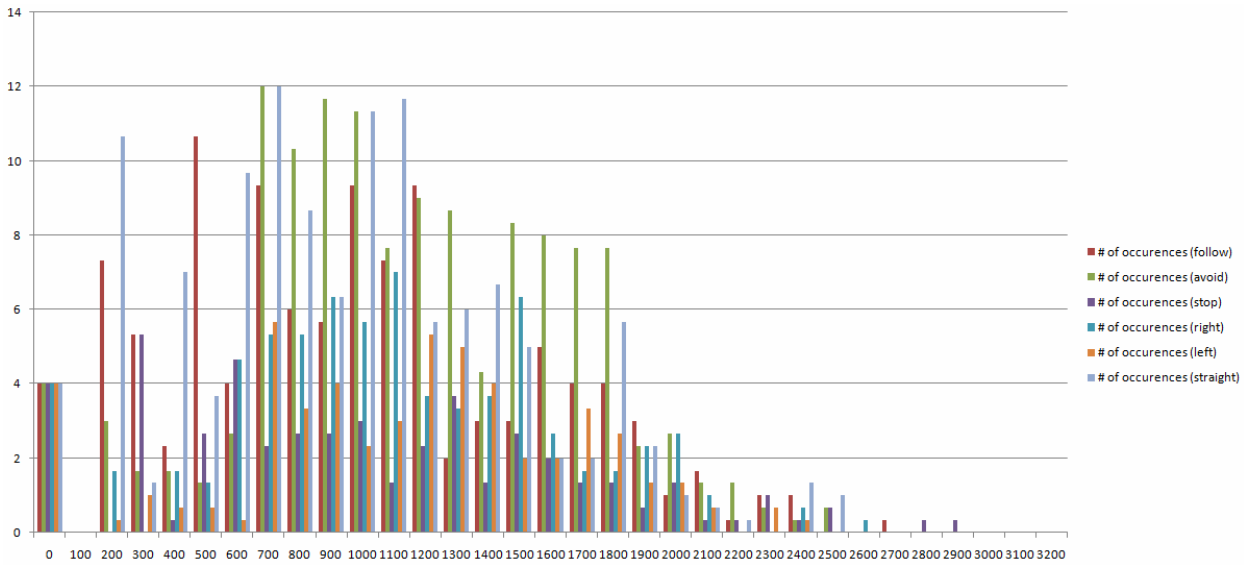

Figure 11: Histogram of Frequency Analysis

Freq (Hz)	follow	avoid	stop	right	left	straight
0	4	4	4	4	4	4
100	0	0	0	0	0	0
200	7.333333333	3	0	1.666666667	0.333333333	10.66666667
300	5.333333333	1.666666667	5.333333333	0	1	1.333333333
400	2.333333333	1.666666667	0.333333333	1.666666667	0.666666667	7
500	10.66666667	1.333333333	2.666666667	1.333333333	0.666666667	3.666666667
600	4	2.666666667	4.666666667	4.666666667	0.333333333	9.666666667
700	9.333333333	12	2.333333333	5.333333333	5.666666667	12
800	6	10.33333333	2.666666667	5.333333333	3.333333333	8.666666667
900	5.666666667	11.66666667	2.666666667	6.333333333	4	6.333333333
1000	9.333333333	11.33333333	3	5.666666667	2.333333333	11.33333333
1100	7.333333333	7.666666667	1.333333333	7	3	11.66666667
1200	9.333333333	9	2.333333333	3.666666667	5.333333333	5.666666667
1300	2	8.666666667	3.666666667	3.333333333	5	6
1400	3	4.333333333	1.333333333	3.666666667	4	6.666666667
1500	3	8.333333333	2.666666667	6.333333333	2	5
1600	5	8	2	2.666666667	2	2
1700	4	7.666666667	1.333333333	1.666666667	3.333333333	2
1800	4	7.666666667	1.333333333	1.666666667	2.666666667	5.666666667
1900	3	2.333333333	0.666666667	2.333333333	1.333333333	2.333333333
2000	1	2.666666667	1.333333333	2.666666667	1.333333333	1
2100	1.666666667	1.333333333	0.333333333	1	0.666666667	0.666666667
2200	0.333333333	1.333333333	0.333333333	0	0	0.333333333
2300	1	0.666666667	1	0	0.666666667	0
2400	1	0.333333333	0.333333333	0.666666667	0.333333333	1.333333333
2500	0	0.666666667	0.666666667	0	0	1
2600	0	0	0	0.333333333	0	0
2700	0.333333333	0	0	0	0	0
2800	0	0	0.333333333	0	0	0
2900	0	0	0.333333333	0	0	0
3000	0	0	0	0	0	0
3100	0	0	0	0	0	0
3200	0	0	0	0	0	0

Table 1: Table of Results for Frequency Analysis

The preceding table shows how different each one of the words are once a frequency analysis is applied, thus proving that the words can be distinguished. The values were obtained from the averages of three trial runs of each word repeated rapidly in succession, for about ten seconds. These values can now be rounded and hard-coded into the device in order to be able to match an incoming command with the saved values.

The second method tested was to include a running sum that took into account the distribution of the signal across the frequency spectrum and a certain sample's distance from the natural frequency of the user. The sum was

calculated by assigning a value of 1 for the natural frequency and incrementing this value after every 100Hz up to 6.4 kHz, the value is then multiplied by the ratio of its magnitude compared to the maximum magnitude in the set of samples and then added to the previous values. The sum is then scaled down to fit an integer value. The table below shows the values obtained.

Command	Calculated Sum
Follow	~43
Go	~26
Obstacle avoidance	~113
Right	~34
Left	~62
Stop moving	~77

Table 2: Frequency Analysis Results for Running Sum

Conclusion

This was a very interesting project to work on. Almost all the proposed goals were achieved. The robot does wall following and obstacle avoidance. The microphone unit is able to decode a command and retransmit it, by doing an FFT operation on the incoming audio signal. The robot responds to the commands issued by the microphone.

What was not as proposed is the RF Link, due to some problems with the device the RF link could not be implemented on the project and instead of wireless communication it is presented with two long wires running from the microphone unit to the platform.

I was greatly limited by the amount of memory on the microcontroller which determined what methods I could use. A second method (running sum) had to be developed because it occupies one fiftieth of the memory peak comparison method occupied. As an effect the accuracy was greatly reduced.

I was also spending a lot of time trying to make the Pololu motor controller work which did not live up to the specs on the data sheet. In retrospect I should have just scratch the whole system and replaced it with a completely different one.

For future work I would definitely pick a better motor, servo option. I would also spend more time working on the platform to make the project look more presentable. I would also spend more time finding a wireless solution.

Appendix and Documentation

Microphone Unit's Blocks

Power Supply

The power supply provides a DC voltage of 4.5V and -4.5V to the microphone unit from a 9V battery. It also provides a stable ground to protect the circuit.

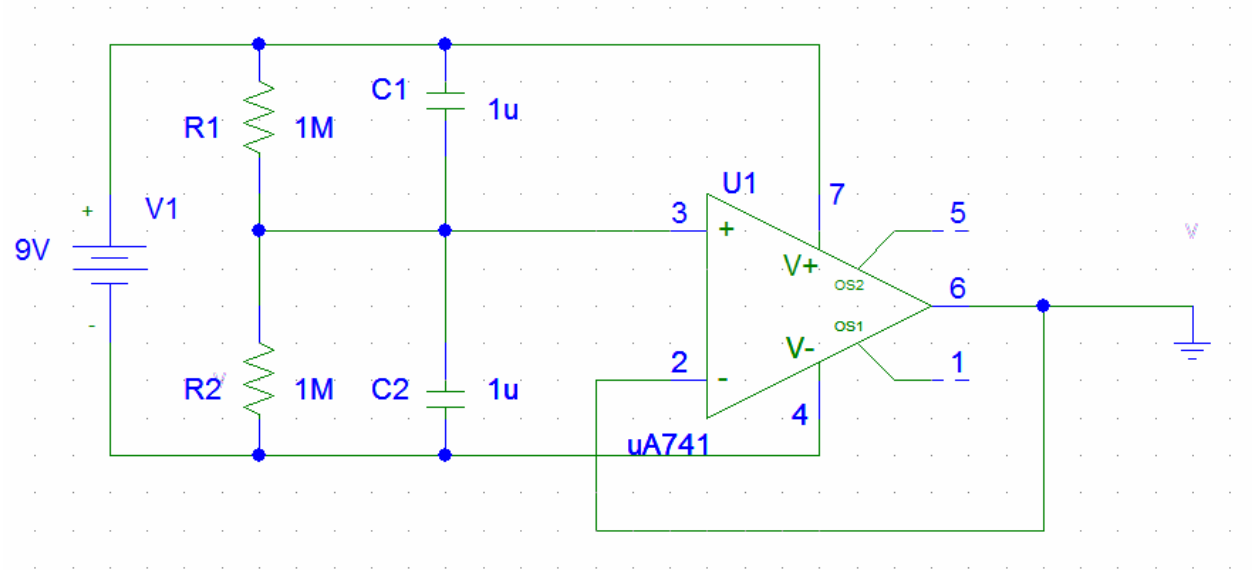

Figure 12: Circuit Diagram of Power Supply

Low Pass Filter

The purpose of the filter is to removes unwanted high frequency noise. For this project I will use a fourth order filter with a cutoff frequency at 3 kHz.

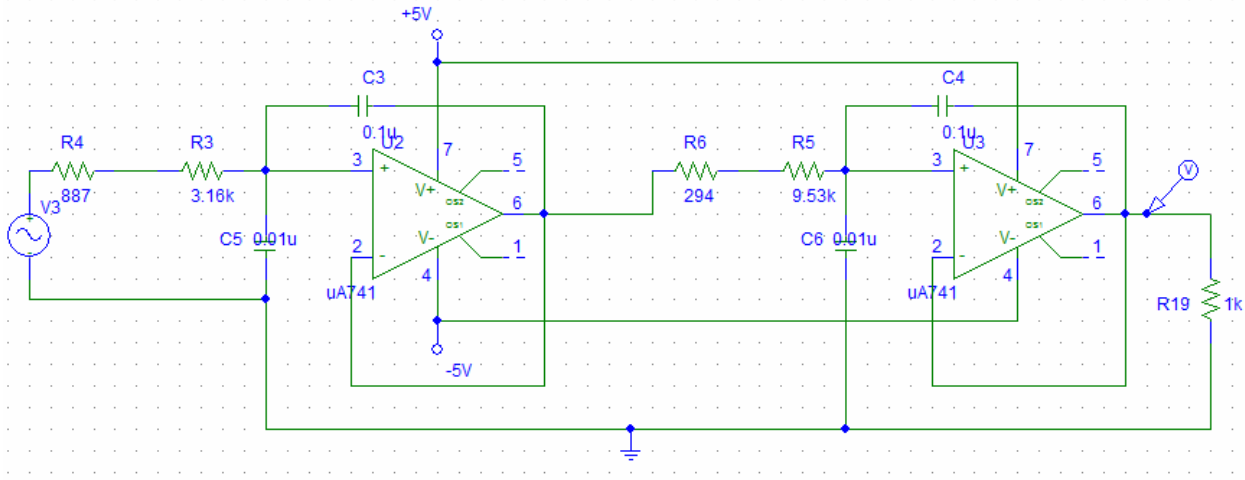

Figure 13: Circuit Diagram for Low Pass Filter

Amplifier

The purpose of the amplifier is to increase the magnitude of the signal. In my case the output of the microphone has a peak to peak voltage in the order of mV and it had to be amplified greatly, in order to produce a signal that the microcontroller can read.

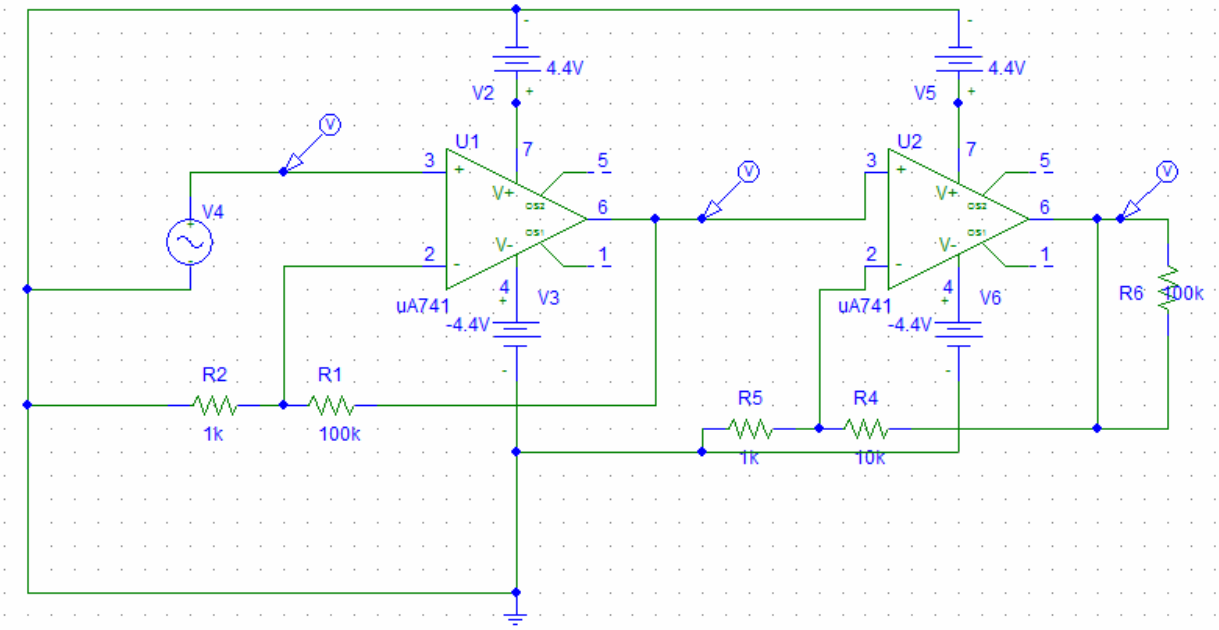

Figure 14: Circuit Diagram for Two-Stage Amplifier

The amplifier I made has a gain of roughly 1000. Which turned out to be more than needed but incidentally it made the microphone function from a greater distance (about six feet).

Microcontroller

The microcontroller employs an FFT algorithm to map the microphone's output signal in the frequency domain and keeps a running sum to identify which command it corresponds to.

References

Lathi, B.P.. Linear Systems and Signals. Oxford University Press, June 18, 2004.

"Fast Fourier transform". Wikimedia Foundation, Inc.. 29 January 2008 <http://en.wikipedia.org/wiki/Fast_fourier_transform>.

Code

Robot.c

```
unsigned rx1;
unsigned i,j,right,left,k;
unsigned long counter1,counter2,counter3;
char txt[6] = "mikro";
unsigned adcRes;
const unsigned int right_forward=0b00000001;
const unsigned int right_reverse=0b00000000;
const unsigned int left_forward=0b00000011;
const unsigned int left_reverse=0b00000010;
const unsigned int motor_delay=50;
const unsigned int IR_threshold=14;
const unsigned int IR_wall_threshold=10;
signed int right_motor,left_motor;
unsigned int behavior,last_behavior;

void motor_init()
{
    /*Uart1_Write_Char(0x80);
    Uart1_Wait_Tx();
    Uart1_Write_Char(0x02);
    Uart1_Wait_Tx();
    Uart1_Write_Char(0);
    Uart1_Wait_Tx();*/
    Delay_100ms();
    right_motor=0;
    left_motor=0;
    PORTCbits.RC14=0;
    Delay_100ms();
    Delay_ms(10);
    LATCbits.LATC14=1;
    PORTCbits.RC14=1;
}

void increase_right()
{
    if(right_motor<-7)
    {
       right_motor+=7;
       Uart1_Write_Char(0x80);
       Uart1_Wait_Tx();
       Uart1_Write_Char(0x00);
       Uart1_Wait_Tx();
```

```
    Uart1_Write_Char(right_reverse);
    Uart1_Wait_Tx();
    Uart1_Write_Char(abs(right_motor));
    Uart1_Wait_Tx();
  }
  else if(right_motor<0)
  {
    right_motor=0;
    Uart1_Write_Char(0x80);
    Uart1_Wait_Tx();
    Uart1_Write_Char(0x00);
    Uart1_Wait_Tx();
    Uart1_Write_Char(right_reverse);
    Uart1_Wait_Tx();
    Uart1_Write_Char(abs(right_motor));
    Uart1_Wait_Tx();
  }
  else if(right_motor<14)
  {
    right_motor+=7;
    Uart1_Write_Char(0x80);
    Uart1_Wait_Tx();
    Uart1_Write_Char(0x00);
    Uart1_Wait_Tx();
    Uart1_Write_Char(right_forward);
    Uart1_Wait_Tx();
    Uart1_Write_Char(right_motor);
    Uart1_Wait_Tx();
  }
  else if(right_motor<21)
  {
    right_motor=21;
    Uart1_Write_Char(0x80);
    Uart1_Wait_Tx();
    Uart1_Write_Char(0x00);
    Uart1_Wait_Tx();
    Uart1_Write_Char(right_forward);
    Uart1_Wait_Tx();
    Uart1_Write_Char(right_motor);
    Uart1_Wait_Tx();
  }
  else
  {
    left_motor=0;
    Uart1_Write_Char(0x80);
    Uart1_Wait_Tx();
```

```
        Uart1_Write_Char(0x00);
        Uart1_Wait_Tx();
        Uart1_Write_Char(left_forward);
        Uart1_Wait_Tx();
        Uart1_Write_Char(left_motor);
        Uart1_Wait_Tx();
    }
}
void decrease_right()
{
    if(right_motor>7)
    {
        right_motor-=7;
        Uart1_Write_Char(0x80);
        Uart1_Wait_Tx();
        Uart1_Write_Char(0x00);
        Uart1_Wait_Tx();
        Uart1_Write_Char(right_forward);
        Uart1_Wait_Tx();
        Uart1_Write_Char(right_motor);
        Uart1_Wait_Tx();
    }
    else if(right_motor>0)
    {
        right_motor=0;
        Uart1_Write_Char(0x80);
        Uart1_Wait_Tx();
        Uart1_Write_Char(0x00);
        Uart1_Wait_Tx();
        Uart1_Write_Char(right_forward);
        Uart1_Wait_Tx();
        Uart1_Write_Char(right_motor);
        Uart1_Wait_Tx();
    }
    else if(right_motor>-14)
    {
        right_motor-=7;
        Uart1_Write_Char(0x80);
        Uart1_Wait_Tx();
        Uart1_Write_Char(0x00);
        Uart1_Wait_Tx();
        Uart1_Write_Char(right_reverse);
        Uart1_Wait_Tx();
        Uart1_Write_Char(abs(right_motor));
        Uart1_Wait_Tx();
    }
```

```
    else if(right_motor>-21)
    {
       right_motor=-21;
       Uart1_Write_Char(0x80);
       Uart1_Wait_Tx();
       Uart1_Write_Char(0x00);
       Uart1_Wait_Tx();
       Uart1_Write_Char(right_reverse);
       Uart1_Wait_Tx();
       Uart1_Write_Char(abs(right_motor));
       Uart1_Wait_Tx();
    }
    else
    {
       left_motor=0;
       Uart1_Write_Char(0x80);
       Uart1_Wait_Tx();
       Uart1_Write_Char(0x00);
       Uart1_Wait_Tx();
       Uart1_Write_Char(left_forward);
       Uart1_Wait_Tx();
       Uart1_Write_Char(left_motor);
       Uart1_Wait_Tx();
    }
}

void increase_left()
{
    if(left_motor<-7)
    {
       left_motor+=7;
       Uart1_Write_Char(0x80);
       Uart1_Wait_Tx();
       Uart1_Write_Char(0x00);
       Uart1_Wait_Tx();
       Uart1_Write_Char(left_reverse);
       Uart1_Wait_Tx();
       Uart1_Write_Char(abs(left_motor));
       Uart1_Wait_Tx();
    }
    else if(left_motor<0)
    {
       left_motor=0;
       Uart1_Write_Char(0x80);
       Uart1_Wait_Tx();
       Uart1_Write_Char(0x00);
```

```
        Uart1_Wait_Tx();
        Uart1_Write_Char(left_forward);
        Uart1_Wait_Tx();
        Uart1_Write_Char(left_motor);
        Uart1_Wait_Tx();
    }
    else if(left_motor<14)
    {
        left_motor+=7;
        Uart1_Write_Char(0x80);
        Uart1_Wait_Tx();
        Uart1_Write_Char(0x00);
        Uart1_Wait_Tx();
        Uart1_Write_Char(left_forward);
        Uart1_Wait_Tx();
        Uart1_Write_Char(left_motor);
        Uart1_Wait_Tx();
    }
    else if(left_motor<21)
    {
        left_motor=21;
        Uart1_Write_Char(0x80);
        Uart1_Wait_Tx();
        Uart1_Write_Char(0x00);
        Uart1_Wait_Tx();
        Uart1_Write_Char(left_forward);
        Uart1_Wait_Tx();
        Uart1_Write_Char(left_motor);
        Uart1_Wait_Tx();
    }
    else
    {
        left_motor=0;
        Uart1_Write_Char(0x80);
        Uart1_Wait_Tx();
        Uart1_Write_Char(0x00);
        Uart1_Wait_Tx();
        Uart1_Write_Char(left_forward);
        Uart1_Wait_Tx();
        Uart1_Write_Char(left_motor);
        Uart1_Wait_Tx();
    }
}
void decrease_left()
{
    if(left_motor>7)
```

```
{
   left_motor-=7;
   Uart1_Write_Char(0x80);
   Uart1_Wait_Tx();
   Uart1_Write_Char(0x00);
   Uart1_Wait_Tx();
   Uart1_Write_Char(left_forward);
   Uart1_Wait_Tx();
   Uart1_Write_Char(abs(left_motor));
   Uart1_Wait_Tx();
}
else if(left_motor>0)
{
   left_motor=0;
   Uart1_Write_Char(0x80);
   Uart1_Wait_Tx();
   Uart1_Write_Char(0x00);
   Uart1_Wait_Tx();
   Uart1_Write_Char(left_forward);
   Uart1_Wait_Tx();
   Uart1_Write_Char(left_motor);
   Uart1_Wait_Tx();
}
else if(left_motor>-14)
{
   left_motor-=7;
   Uart1_Write_Char(0x80);
   Uart1_Wait_Tx();
   Uart1_Write_Char(0x00);
   Uart1_Wait_Tx();
   Uart1_Write_Char(left_reverse);
   Uart1_Wait_Tx();
   Uart1_Write_Char(abs(left_motor));
   Uart1_Wait_Tx();
}
else if(left_motor>-21)
{
   left_motor=-21;
   Uart1_Write_Char(0x80);
   Uart1_Wait_Tx();
   Uart1_Write_Char(0x00);
   Uart1_Wait_Tx();
   Uart1_Write_Char(left_reverse);
   Uart1_Wait_Tx();
   Uart1_Write_Char(abs(left_motor));
   Uart1_Wait_Tx();
```

```
    }
    else
    {
       left_motor=0;
       Uart1_Write_Char(0x80);
       Uart1_Wait_Tx();
       Uart1_Write_Char(0x00);
       Uart1_Wait_Tx();
       Uart1_Write_Char(left_forward);
       Uart1_Wait_Tx();
       Uart1_Write_Char(left_motor);
       Uart1_Wait_Tx();
    }
}

void stop()
{
        while(right_motor<0)
        {
                increase_right();
                Delay_ms(motor_delay);
        }
        while(right_motor>0)
        {
                decrease_right();
                Delay_ms(motor_delay);
        }
        while(left_motor<0)
        {
                increase_left();
                Delay_ms(motor_delay);
        }
        while(left_motor>0)
        {
                decrease_left();
                Delay_ms(motor_delay);
        }
}
void turn_right()
{
        while(right_motor<21)
        {
                increase_right();
                Delay_ms(motor_delay);
        }
      while(left_motor<21)
```

```
        {
                increase_left();
                Delay_ms(motor_delay);
        }
}
void turn_left()
{
        while(left_motor>-21)
        {
                decrease_left();
                Delay_ms(motor_delay);
        }
      while(right_motor>-21)
        {
                decrease_right();
                Delay_ms(motor_delay);
        }
}

void go_straight()
{
        while(left_motor<21)
        {
                increase_left();
                Delay_ms(motor_delay);
        }
      while(right_motor>-21)
        {
                decrease_right();
                Delay_ms(motor_delay);
        }
}

void reverse()
{
        while(right_motor<21)
        {
                increase_right();
                Delay_ms(motor_delay);
        }
      while(left_motor>-21)
        {
                decrease_left();
                Delay_ms(motor_delay);
        }
}
```

```
void print_LCD(int val,int pos)
{
   /*IntToStr(val,txt);
   switch(pos)
   {
      case 1: Lcd_Custom_Out(1,6,txt);break;
      case 2: Lcd_Custom_Out(2,3,txt);break;
      case 3: Lcd_Custom_Out(2,8,txt);break;
      case 4: Lcd_Custom_Out(1,3,txt);break;
      case 5: Lcd_Custom_Out(1,8,txt);break;
      default: Lcd_Custom_Out(1,6,"ERROR");break;
   }*/
}
int display_Sensor(unsigned int ANX,unsigned int pos)
{
   if(ANX>1&&ANX<13)
   {
      adcRes=Adc_Read(ANX);
      adcRes=((255-(adcRes/16))/10);
      print_LCD(adcRes,pos);
      return adcRes;
   }
   else if(ANX==0||ANX==1)
   {
      adcRes = Adc_Read(ANX);
      adcRes=adcRes/8;
      print_LCD(adcRes,pos);
      return adcRes;
   }
   else{return 0;}
}

// determines whether the path in front is clear and adjusts motors
// accordingly
void behavior_obstacle_avoid()
{
    right =  display_Sensor(12,3);
    left = display_Sensor(11,2);
    if(right>IR_threshold&&left>IR_threshold)
    { reverse(); }
    else if(right<=IR_threshold&&left<=IR_threshold)
    { go_straight(); }
    else if(left<=IR_threshold)
    { turn_right(); }
    else if(right<=IR_threshold)
    { turn_left(); }
```

```
    else
    {
       Lcd_Custom_Out(1,6,"ERROR");
       while(1);
    }
    Delay_ms(motor_delay);
    if(left_motor<0&&right_motor<0)
    {
       if(display_Sensor(0,1)<10)
       {
          stop();
          i=rand()+1;
            i=i%6;
            turn_left();
            for(j=0;j<i;j++)
          {
                  Delay_1sec();
          }
            stop();
            Delay_1sec();
       }
    }
    Delay_ms(500);
}

void behavior_wall_following()
{
    int a= display_Sensor(6,4);
    int b= display_Sensor(5,5) ;
    right =  display_Sensor(12,3);
    left = display_Sensor(11,2);
    if((a>=IR_wall_threshold&&a<=16)&&(b>=IR_wall_threshold&&b<=16))
    {
       if(right>=IR_threshold&&left>=IR_threshold)
       { reverse(); }
       if(right<IR_threshold||left<IR_threshold)
       { turn_right(); }
       Delay_ms(500);
    }
    else
if((a>=16&&(b>=IR_wall_threshold&&b<=16))||(b<=10&&(a>=IR_wall_threshold&
&a<=16)))
    {
       if(right>=IR_threshold&&left>=IR_threshold)
       { turn_left(); }
       if(right<IR_threshold||left<IR_threshold)
```

```
        { turn_right(); }
        Delay_ms(500);
    }
    else
if((b>=16&&(a>=IR_wall_threshold&&a<=16))||(a<=10&&(b>=IR_wall_threshold&
&b<=16)))
    {
        turn_right();
        Delay_ms(500);
    }
    else
    {
        if(right>=IR_threshold&&left>=IR_threshold)
        { reverse(); }
        if(right<IR_threshold||left<IR_threshold)
        { turn_right(); }
        Delay_ms(500);
    }
}
void behavior_go()
{
    if(behavior!=last_behavior)
    {
      k=rand()+1;
        k=k%6;
        go_straight();
        for(j=0;j<k;j++)
        {
                Delay_1sec();
        }
        stop();
        Delay_1sec();
    }
}

void behavior_right()
{
    if(behavior!=last_behavior)
    {
      k=rand()+1;
        k=k%6;
        turn_right();
        for(j=0;j<k;j++)
        {
                Delay_1sec();
        }
```

```
        stop();
        Delay_1sec();
    }
}

void behavior_left()
{
    if(behavior!=last_behavior)
    {
      k=rand()+1;
        k=k%6;
        turn_left();
        for(j=0;j<k;j++)
        {
                Delay_1sec();
        }
        stop();
        Delay_1sec();
    }
}

void behavior_stop()
{
    if(behavior!=last_behavior)
    {
        stop();
        Delay_1sec();
    }
}

void print_behavior()
{
   switch(behavior)
   {
      case 0: Lcd_Custom_Out(1,5,"avoid");break;
      case 1: Lcd_Custom_Out(1,5,"follow");break;
      case 2: Lcd_Custom_Out(1,5,"go");break;
      case 3: Lcd_Custom_Out(1,5,"right");break;
      case 4: Lcd_Custom_Out(1,5,"left");break;
      case 5: Lcd_Custom_Out(1,5,"stop");break;
      default: Lcd_Custom_Out(1,5,"ERROR");break;
   }
   /*switch(behavior)
   {
      case 0: Lcd_Custom_Out(2,5,"avoid");break;
      case 1: Lcd_Custom_Out(2,5,"follow");break;
```

```
        case 2: Lcd_Custom_Out(2,5,"go");break;
        case 3: Lcd_Custom_Out(2,5,"right");break;
        case 4: Lcd_Custom_Out(2,5,"left");break;
        case 5: Lcd_Custom_Out(2,5,"stop");break;
        default: Lcd_Custom_Out(2,5,"ERROR");break;
    }*/
}

void check_behavior()
{
    //Lcd_Custom_Out(1,6,"Start");
    k=0;
    counter1=0;
    counter2=0;
    counter3=0;
    if(PORTCbits.RC13==1)
    {
        while(PORTCbits.RC13==1 && counter3<320) {counter3++;}
        while(PORTCbits.RC13==0 && counter1<320)  {counter1++;}
        while(PORTCbits.RC13==1 && counter2<320) {counter2++;}
        k=(counter1+counter2)  ;
    }
    else
    {
        while(PORTCbits.RC13==0 && counter3<320) {counter3++;}
        while(PORTCbits.RC13==1 && counter2<320)  {counter2++;}
        while(PORTCbits.RC13==0 && counter1<320) {counter1++;}
        k=(counter1+counter2)  ;
    }
    if(k==320){;}
    else if(k==41)
    {
        last_behavior=behavior;
        behavior=2;
        Lcd_Custom_Out(2,2, "GO");
    }
    else if(k==61)
    {
        last_behavior=behavior;
        behavior=1;
        Lcd_Custom_Out(2,2, "WF");
    }
    else if(k==81)
    {
        last_behavior=behavior;
        behavior=0;
```

```
        Lcd_Custom_Out(2,2, "OA");
    }
    else if(k==101)
    {
        last_behavior=behavior;
        behavior=4;
        Lcd_Custom_Out(2,2, "TL");
    }
    else if(k==121)
    {
        last_behavior=behavior;
        behavior=3;
        Lcd_Custom_Out(2,2, "TR");
    }
    else if(k==141)
    {
        last_behavior=behavior;
        behavior=5;
        Lcd_Custom_Out(2,2, "ST");
    }
    else
    { Delay_10ms(); }
    IntToStr(k,txt);
    Lcd_Custom_Out(2,8, txt);
}

void Main_Init()
{
    PORTB = 0x0000;
    TRISB = 0xFFFF;       // set pin as input - needed for ADC to work
    TRISF=0;
    TRISD=0;
    TRISC=0xFFFF;
    last_behavior=0;
    behavior=rand()%2;          //code for obstacle avoidance
    Lcd_Custom_Config(&PORTF, 5,4,1,0, &PORTD, 0,2,1);
    Lcd_Custom_Out(1,5,"START") ;
    Uart1_Init(19200);               // initialize USART module
    Delay_100ms();
    motor_init();                   //stops motors and initializes internal counters
    Delay_ms(motor_delay);                  // pause for usart lines stabilization
    rx1 = Uart1_Read_Char();          // perform dummy read to clear the register
}

void main() {
     Main_Init();
```

```
    while(1)
    {
       check_behavior();
       if(behavior==0)
       { behavior_obstacle_avoid(); }
       else if(behavior==1)
       { behavior_wall_following(); }
       else if(behavior==2)
       { behavior_go(); }
       else if(behavior==3)
       { behavior_right(); }
       else if(behavior==4)
       { behavior_left(); }
       else if(behavior==5)
       { behavior_stop(); }
       else
       { Lcd_Custom_Out(2,1,"error"); }
       //Lcd_Custom_Out(2,5,"Cont") ;
       print_behavior();
    }
}//~!
```

Microphone.c

```
#include "Glcd_Fonts.h"


  unsigned Samples[256] absolute 0x0C00 ;  // Y data space for P30F4013-
required by FFT routine
                                // See datasheet for your dsPIC to see Y data space
limits.
  unsigned freq;                        // Auxiliary variables
  char txt[5];
  unsigned Written[64];
  char rx1[3];
  float sumpartial,sumtotal;
  unsigned maximum,a,counter,command;
  int sum;
  const unsigned int go=26;
  const unsigned int stop=77;
  const unsigned int right=35;
  const unsigned int left=62;
  const unsigned int avoid=113;
  const unsigned int follow=43;
// The following trap procedures are not really needed here, they
```

```
//   are used here just for the sake of demonstration.
void OscillatorFailTrap() org 0x06 { // if oscillator fails, the code jumps here
  trisf = 0;
    asm{
    MOV [w15-34], w13
    LSR  w13, #8, w13
    MOV w13, LATF
    ;LSR  w15, #8, w13
    ;MOV w13, LATB

  }
  while(1);
}

void AddressTrap() org 0x08 {  // if the addressing mode is wrong, the code
jumps here
    trisd = 0;
    asm{
    MOV [w15-34], w13
    ;LSR  w13, #8, w13
    MOV w13, LATd
  }
  while(1);
}

void StackErrorTrap() org 0x0A { // stack overflow, underflow...
trisf = 0;
    asm{
    MOV [w15-34], w13
    LSR  w13, #8, w13
    MOV w13, LATF
    ;LSR  w15, #8, w13
    ;MOV w13, LATB

  }  while(1);
}

void MathErrorTrap() org 0x0C { // div by zero etc...
  trisf = 0;
    asm{
    MOV [w15-34], w13
    LSR  w13, #8, w13
    MOV w13, LATF
    ;LSR  w15, #8, w13
    ;MOV w13, LATB
```

```
  }
  while(1);
}

//-------------- Initialization of AD converter
void InitAdc() {

  ADPCFG = 0x00FF; // PORTB<8:15> is analogue, PORTB<0:7> is digital
  TRISB.F8 = 1;    // RB8 as input pin

  ADCHS  = 8;      // Connect RBxx/ANxx as CH8 input. RB8 is input pin
  ADCSSL = 0;      //
  ADCON3 = 0x1F3F; // sample time = 31 Tad.
  ADCON2 = 0;      //
  ADCON1 = 0x83E0; // turn ADC ON, fractional result, internal counter ends
conversion

}//~



//-------------- Initialize GLCD for EASYdsPIC4 board
void InitGlcd() {

  Glcd_Init_EasydsPIC3();
  Glcd_Set_Font(FontSystem5x8, 5, 8, 32);
  Glcd_Fill(0xAA); // Show stripes on GLCD to signalize startup

  Delay_ms(500);   // Wait for a while
  Glcd_Fill(0x00); // Clear screen
}//~

//-------------- Main Initialization
void MainInit() {

  TRISDbits.TRISD8=0;
  LATDbits.LATD8=0;
  TRISCbits.TRISC14=1;
  LATCbits.LATC14=0;
  TRISCbits.TRISC13=1;
  LATCbits.LATC13=0;
  TRISAbits.TRISA11=0;
  LATAbits.LATA11=0;
  InitAdc();
  InitGlcd();
  Twiddle_Factors_Init();
```

```
  Vector_Set(Written, 64, 0xFFFF);  // Fill "Written" with $FFFF
  Glcd_Write_Text(" Hz", 100, 0, 1);
  command=0;
  counter=0;

}//~


//-------------- Auxiliary function for converting 1.15 radix point to
//               IEEE floating point variable (needed for sqrt).
float Fract2Float(int input) {

   if (input < 0)
     input = - input;
   return (input / 32768.);
}//~


//-------------- Data output procedure. It draws FFT components on GLCD.
//               GLCD coordinate system starts at top left corner. Therefore,
//               line drawing had to be modified in order to achieve
//               a viewable spectrum on screen.
//               "Samples" at this moment contains DFT of the signal in the manner
Re, Im, Re, Im...
void WriteData() {

  unsigned Re, Im, tmpw, j, k, l;
  float    Rer, Imr, tmpR;
  l      = 0;
  j      = 0;                   // If you want to skip DC component then make j >= 1
  k      = 0;
  maximum    = 0;
  freq = 0;                      // Reset current max. frequency for new reading

  while (k <= 63) {
     Re = Samples[j++];          // Real part of DFT sample

     Im = Samples[j++];          // Imaginary part of DFT sample

     Rer = Fract2Float(Re);      // conversion to IEEE floating point
     Imr = Fract2Float(Im);      // conversion to IEEE floating point

     tmpR = Rer * Rer;           // Re^2
     Rer  = tmpR;
     tmpR = Imr * Imr;           // Im^2
     Imr  = tmpR;
```

```
    tmpR = sqrt(Rer + Imr);      // Amplitude of current DFT sample
    Rer  = tmpR * 256.;          // DFT is scaled down by 1/N, we need to
                                 //   take it back in order to have visible
                                 //   components on GLCD
    Re = Rer;
    sumpartial+=Re*k;
    if (Re > 63)
      if(k != 0)
        //--- NOTE: rejecting values for k=0 removes strong DC component
        Re = Written[k-1];       // k = 0? beware of the glitch
      else
        Re = 0;

    if (Re > maximum) {
        maximum = Re;
        freq = k;                // This should be the center frequency of the signal
    }

    tmpw = Written[k];
    if (tmpw != Re) {             // Draw only those components that changed
        l = 64 - tmpw;           // 64 lines on GLCD on Y axis
        while (l <= 63) {        // Clear line to the bottom of the screen
          Glcd_Dot(k, l, 0);
          l++;
        }

         l = 64 - Re;            // Draw line to the bottom of the screen
         while (l <= 63) {
            Glcd_Dot(k, l, 1);
            l++;
         }
         Written[k] = Re;        // Mark that the current sample has been drawn
    }


    k++;                         // Move current X coordinate
  }

  //--- Write the frequency of max. amplitude sample
  /*freq *= 100;
  WordToStr(freq, txt);
  Glcd_Write_Text(txt, 70, 0, 1);*/
}//~


//---------- Takes current sample
```

```
unsigned ReadAdc() {

  ADCON1.F1 = 1;                // Start AD conversion
  while (ADCON1.F0 == 0)           // Wait for ADC to finish
    asm nop;
  return ADCBUF0;               // Get ADC value
}//~


//-------------- Fills "Samples" with input samples in manner Re, Im, Re, Im... where
Im = 0
void SampleInput() {

  int i = 0;

  while (i <= 255) {
       Samples[i++] = Adc_Read(8); //Re
       Samples[i++] = 0;          // Im
       Delay_us(167);
  }
  // "Samples" now contains 128 pairs of <Re, Im> samples
}//~

void id_command(int x)
{
     if(abs(x-go)<=3)
     {
       command=1;
       Glcd_Write_Text("GO FORWARD",10,3,1);
     }
     else if(abs(x-follow)<=3)
     {
       command=2;
       Glcd_Write_Text("W. FOLLOWING",10,3,1);
     }
     else if(abs(x-avoid)<=3)
     {
       command=3;
       Glcd_Write_Text("O. AVOIDANCE",10,3,1);
     }
     else if(abs(x-left)<=3)
     {
       command=4;
       Glcd_Write_Text("TURN LEFT",10,3,1);
     }
     else if(abs(x-right)<=3)
```

```
      {
        command=5;
        Glcd_Write_Text("TURN RIGHT",10,3,1);
      }
      else if(abs(x-stop)<=3)
      {
        command=6;
        Glcd_Write_Text("STOP",10,3,1);
      }

}

void transmit_command()
{
    int i=0;
    PORTAbits.RA11=1;
    for(i=0;i<=command;i++){ Delay_10us(); }
    PORTAbits.RA11=0;
    for(i=0;i<=command;i++){ Delay_10us(); }
    PORTAbits.RA11=1;
    for(i=0;i<=command;i++){ Delay_10us(); }
    PORTAbits.RA11=0;
    for(i=0;i<=command;i++){ Delay_10us(); }
    PORTAbits.RA11=1;
    for(i=0;i<=command;i++){ Delay_10us(); }
    PORTAbits.RA11=0;
    for(i=0;i<=command;i++){ Delay_10us(); }
}

//-------------- Main program starts here
void main() {
  unsigned i;
  MainInit();                  // Initialize all
  while (1) {                  // Infinite loop
      if(PORTCbits.RC14==1)
      {
          Delay_1sec();
          PORTDbits.RD8=1;
          sumtotal=0.0;
          for(a=0;a<50;a++)
          {
              sumpartial=0.0;
              SampleInput();            // Sample input signal

              // Perform FFT (DFT), 7 stages, 128 samples of complex pairs
              // Twiddle factors are taken from the <TwiddleFactors.dpas> unit
```

```
            FFT(7, TwiddleCoeff_128, Samples);

            // DFT butterfly algorythm bit-reverses output samples.
            // We have to restore them in natural order.
            BitReverseComplex(7, Samples);

            Glcd_Fill(0x00); // Clear screen
            // Draw DFT samples on GLCD
            WriteData();

            if(maximum!=0)
            {
               sumtotal+=sumpartial/maximum;
               counter++;
            }

         }
         PORTDbits.RD8=0;
         while(PORTCbits.RC13==0){;}
         Glcd_Fill(0x00); // Clear screen
         sum=(int)(ceil(sumtotal/counter));

         WordToStr(sumtotal,rx1);
         Glcd_Set_Font(System3x6, 3, 6, 0x20);
         Glcd_Write_Text(rx1, 10 ,0, 1);
         id_command(sumtotal);
         transmit_command();
         counter=0;
      }
      transmit_command();
   }
}//~!
```