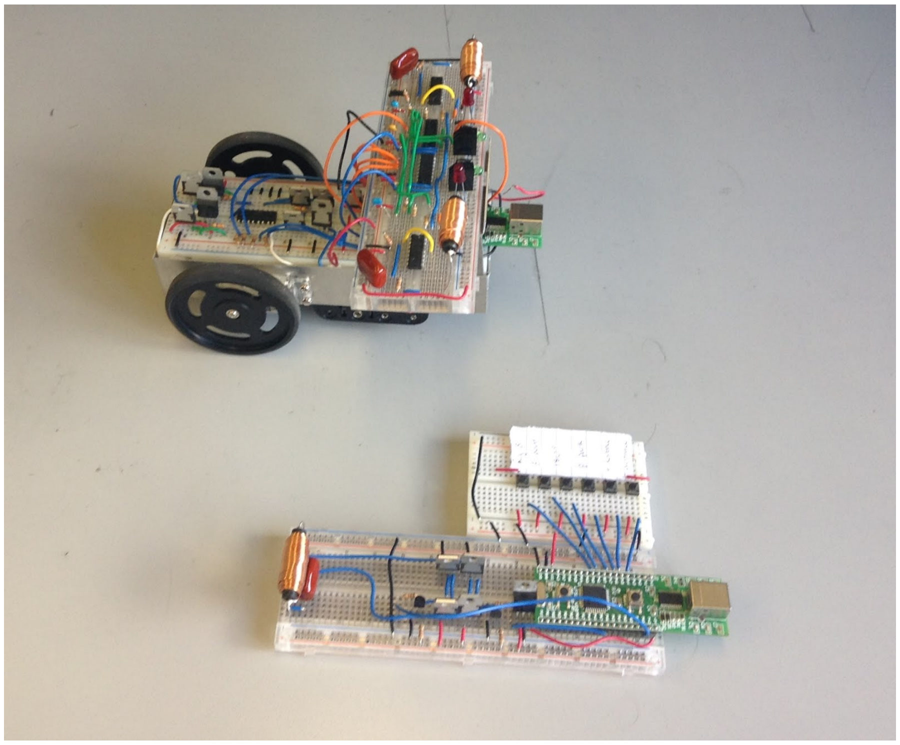
Electromagnetic Tether Robot Report

EECE 281

Section 201 L2A

Submitted to Jesus Calvino-Fraga



April 7, 2014

Submitted by Group A9:

Alec Ng, 32386120

Alex Charles, 3670012

Amy Davies, 35972124

Daniel Sallaway, 36615128

Maria Strasky 36615128

Paula Morales, 36053122

TABLE OF CONTENTS

1.0 INTRODUCTION

Objective:

The objective of this project was to create an autonomous robot that is tethered by electromagnetic waves. The robot was to maintain a fixed distance and orientation relative to this transmitter through the use of two motors. The robot also had to respond to commands sent via the transmitter.

Specifications:

The robot must be written using C language and programmed onto an LP51B board. It should be solely powered using batteries and we have to make a chassis attached with the provided wheels, two back wheels and a ball caster for the front wheel. The chassis must store the circuit for the motor made out of H-bridges consisting of NMOS and PMOS transistors and the structure should hold the battery pack to power the motor. The robot requires a transmitter controlled by a second LP51B board that sends a signal by 'bit bang'. The receiver on the robot reads this bit bang signal and translates the signal into a command and performs the command. The robot should be able to do are move closer, move farther, rotate 180 degrees and parallel park in space that is one and half times the length of the robot. The move closer and move farther functions should have at least four preset distances. The distance that the robot should be able track the transmitter from at least 50cm away. When the robot is not reading a transmission, it should maintain it's distance and orientation with respect to the transmitter.

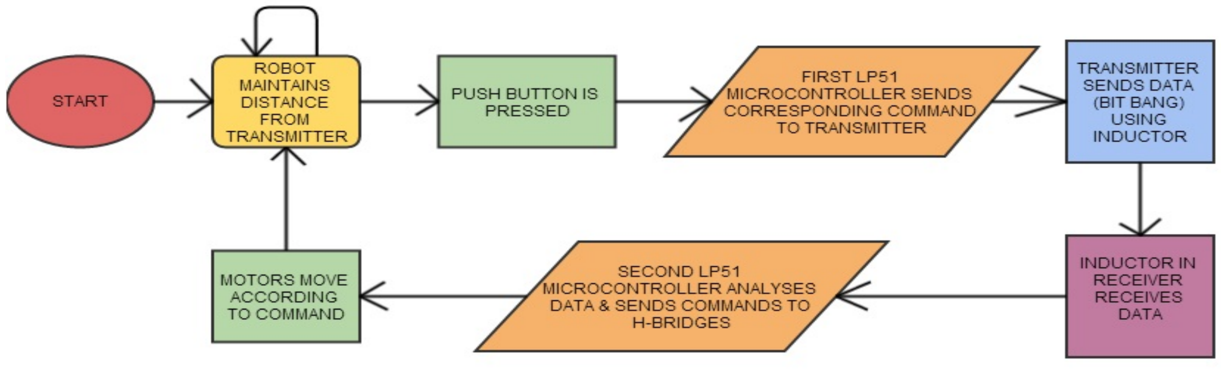

START
ROBOT MAINTAINS DISTANCE FROM TRANSMITTER
PUSH BUTTON IS PRESSED
FIRST LP51 MICROCONTROLLER SENDS CORRESPONDING COMMAND TO TRANSMITTER
TRANSMITTER SENDS DATA (BIT BANG) USING INDUCTOR
INDUCTOR IN RECEIVER RECEIVES DATA
SECOND LP51 MICROCONTROLLER ANALYSES DATA & SENDS COMMANDS TO H-BRIDGES
MOTORS MOVE ACCORDING TO COMMAND

Figure 1 - Basic Hardware Block Diagram

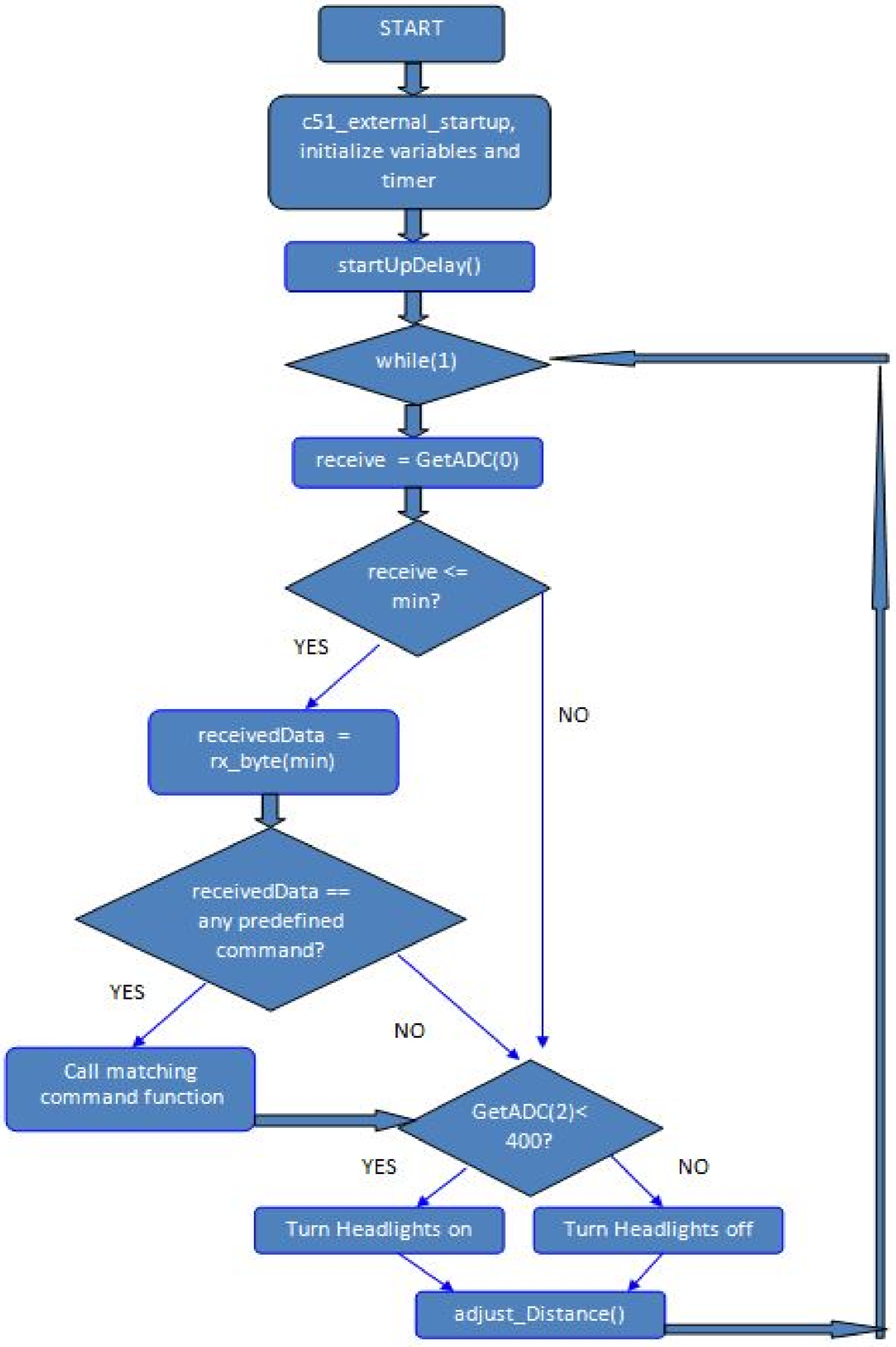

START
c51_external_startup, initialize variables and timer
startUpDelay()
while(1)
receive = GetADC(0)
receive <= min?
YES
NO
receivedData = rx_byte(min)
receivedData == any predefined command?
YES
NO
Call matching command function
GetADC(2)< 400?
YES
NO
Turn Headlights on
Turn Headlights off
adjust_Distance()

Figure 2 - System Block Diagram for Receiver.c

2.0 INVESTIGATION

2.1 Idea Generation

Before beginning the project, we had a team meeting where we discussed different aspects of the project, and divided the work into four sections. They were as follows: the transmitter, the receiver, the C code, and construction of the robot and the motor circuit. Tasks were divided according to who felt the most confident in regards to particular design aspects. After this distinction, we rebounded ideas off of each other for extra features. We decided to have headlights that turn on when a photoresistor senses that it is dark outside, as well as turning signals to indicate when the robot is parallel parking or turning left or right. Additionally, we decided that implementing a three-point turn and a figure eight feature would add greater functionality to the robot, by mimicking actions that cars perform in real life.

2.2 Investigation Design

The initial investigation and experimentation for each section was done in small groups.The transmitter was initially tested with a signal generator, to ensure that the proper resonant frequency had been calculated. Then, the oscilloscope was used to determine that the microcontroller was successfully outputting at this frequency. The receiver team referred to Module 4, where a peak detector was also utilized. It was determined that a very small capacitor was to be used, which would allow data to be transmitted faster, as the capacitor needed to discharge before a new value could be read, to prevent any errors in reading the data. The motor and robot team referred to Module 6 for instruction on how to produce the H-bridge circuit.

2.3 Data Collection

All teams used the provided lecture slides, as well as information gained from the previous modules when designing their sections. Additionally, for the design of the receiver, information for our circuit components was gathered from the data sheets for the ADC[1] and the op-amp[2]. Calculations were performed to estimate an appropriate amplification factor for the inverting amplifier. With these, we had to keep in mind the effect noise would have on our output. When working with the turning signals, we referred to the NE556 dual timer datasheet [3].

2.4 Data Synthesis

Throughout the development of the project, we conducted numerous tests on the transmitter, receiver, and code to ensure that they were functioning according to our expectations. Aspects of the motor control code was tested once the robot had been constructed. Different commands were sent via the serial port to ensure the motors were working as expected. The transmitter team’s primary goal was to ensure a consistent high voltage across the inductor. This was examined with the oscilloscope. Once the transmitter was working, we were able to perform tests using the oscilloscope at different points in the receiver circuit to see if the data was being transmitted properly.

2.5 Analysis of Results

After collecting data we ensured that our results were within a close range of the results that were expected. One of the main considerations we had to take into account was noise in our circuits. We needed to ensure that other EM sources did not have an effect on the performance of our robot. We analyzed the voltage values that we found in our receiver at several distances, to ensure that we had would have a large enough range and adjusted our amplification values accordingly. We also measured noise in our receiver circuit by turning off our transmitter and set buffers in our code in order to avoid having our robot react to noise. After this fine tuning, we ensured that the robot would not move when the transmitter was stationary but was still responsive when the transmitter was being moved. We performed final tests on all of the functions of the robot to ensure that it was fully working and that none of our modifications had had adverse effects on any other aspects of the robot.

3.0 DESIGN

3.1 Use of Process

We worked in partners to discuss possible ways to implement portions of the robot. We relied on previous knowledge as well as lecture slides, data sheets, and online resources in order to find possible implementations for the parts of our robot. We also discussed possible issues that could arise and ways to catch these early by testing frequently to avoid any standstills and lengthy troubleshooting. We kept our discussions open for brainstorming then discussed our ideas to find the best solutions. Each pair communicated with the others to ensure seamless integration of the different parts. Lastly, information from datasheets, circuit diagrams, and any calculations were recorded within our lab books and our code, ensuring a concrete reference for any member of our team.

3.2 Need and Constraint Identification

Throughout the project, numerous constraints were identified. One constraint was the use of batteries to power the circuit - 4 AA batteries for the robot and a square 9V for the receiver. Using batteries instead of a solid power source limited the current and power fed to the robot and transmitter and required us to work with a lower power supply, but in turn, increased mobility. In addition, there was a lack of space on the robot; we needed two breadboards for all the ICs and electrical components. As a result, we built a metal platform to strategically fit another breadboard on top of the one originally fitted on. Moreover, magnetic interference from other groups was a concern. As a solution, we set buffers in the code to ignore any small readings that were not meant to be read.

3.3 Problem Specification

After discussing the project and beginning to build and test some prototype circuits, it became evident that we needed to use multiple breadboards to fit all of our circuitry. We also discovered that there was a fair amount of noise in our receiver circuit, possibly caused by other groups, other electronic devices, or even the two tank circuits having an effect on each other. Hence, we needed to implement some sort of noise cancellation in order for the receiver to be able to properly read zero bits in the data transmissions. We also discovered that the range of our robot would be very small with the transmitter that we had, so we needed to incorporate an amplifier into the receiver circuit in order to still get large enough readings at larger distances.

3.4 Solution Generation

After realising that we would have trouble fitting our receiver circuit, motor controller circuit, and extra features on one breadboard, we discussed solutions such as having two breadboards side by side or having two directly stacked on one another. We decided to stack our receiver circuit on top of our motor controller circuit, and orient it sideways. This allowed us to put greater space between our two inductors in the receiver in order to get more accurate readings of the angle of the robot. We initially tried an amplification factor of 20 but found that our amplifier would saturate at short distance. We then considered the measurements we had taken, and settled on an amplification factor of 10 which worked properly in the required range. We also implemented a buffer in the code which would read any voltages below a certain point as being zero, which resolved the issue with noise in our circuit.

3.5 Solution Evaluation

To ensure that our solution was correct, we began by testing the individual parts of our robot. We started with the transmitter, ensuring that it output a steady sine wave at 15.1kHz and over 150V peak using the oscilloscope. We also tested the buttons to ensure that the transmitter properly transmitted our binary data. We next tested the receiver, starting with the LC tank circuit to ensure that it carried an appropriate sine wave that varied with distance from the transmitter. We measured the output of the amplifier to ensure that it was giving us the amplification factor of twenty, and that the signal would never saturate within the required range of the robot. We then tested the peak detector to ensure that it carried the correct voltage and changed quickly enough when receiving data bits. We built and tested the motor control circuitry separately and used it to test the code for the various functions that we had written. After we were sure that our circuitry and code was working correctly individually, we integrated it into one piece. We then tested this by setting up the robot and running it, adjusting any buffers and time values as required. Once we had this functioning as required by the criteria, we were confident that we had chosen a suitable final design for our robot.

3.6 Detailed Design

The following detail design pertains to the “Robot_Receiver.c” file that is the first .c file listed in the appendix. Each section referred to below is clearly labelled in the source file.

- Initialization

Set up and enabled timer 0 interrupts to provide PWM to power the DC motors. Also initialized other extra variables such as ones that control the turn signal LEDs.

- SPI Functions

The functions SPIWrite(), GetADC() and voltage() were taken from code given to us in Lab 5 of EECE 281, when we needed to implement a phasor meter.

- Big Bang Functions

These functions were taken from the Project 2 Lecture slides that were presented on March 17, 2014 during EECE 281.- refer to Project 2 lecture slides. The wait_bit_time() and wait_one_and_half_bit_time() functions were calibrated so they corresponded with the functions that had been written in the transmitter’s code. These delays were arrived at by experimentation to find delay that would give us the fastest transmission that was still accurate.

- Different Delays

This section contains 10 different delays that all are used in different functions. Delays that were used to delay for a fixed amount of time included startUpDelay, SlightDelay, motorDelay (used after robot movement to make motion more fluid) and threeQuartSec. Delays that were used in order to angle the robot a certain position included rotateDelay (time to rotate robot 180), fortyFive (rotate robot at a 45 degree angle) and arcTurnDelay(turn and end up at a 90 degree angle). The remaining delays were calibrated to have the robot move a certain distance. These delays include threePtBackDelay and distDelay.

- Motor Initialization Commands

Because we have two DC motors, we need four pins. We determined which pins correspond to what movement in the motor (e.g. LeftMotor_Forward would be the pin that would make the left motor on the robot go forward). There are 10 initialization commands to orient the motors to do specific tasks, such as move backwards or turn 45 degrees. Commands accounted for a reverse direction when the robot was reversed due to a command, like rotate 180. Each motor is assigned an appropriate PWM value, which were constants defined at the top of the file. Some initialization commands are straightforward, such as init_forwards, but some initialization commands such as init_45_arcTurn required testing. This specific function was designed so one motor would overpower the other, thus propelling the vehicle forward while simultaneously turning - hence the name.

- Different Commands

For all commands that involve movement, motion is created by following a specific sequence. First, the appropriate motor initialization command is called. Then, we set the timer 0 interrupt enable, call an appropriate delay (here is when the robot moves), then clear the timer 0 interrupt enable. We then finally call a motor initialization that stops all motors.

- *move_closer*

  Simply returns a modified, shorter distance (smaller value) to be handled by the adjust_distance function, which is explained later.

- *move_further*

  Same concept as move_closer, except returns a bigger value.

- rotate_180

Used the appropriate motor initialization command and delay to make the robot rotate 180. The orientation is switched and returned if the user wants to continue normal function with a reverse orientation resulting from a 180 rotation.

- parallel_park

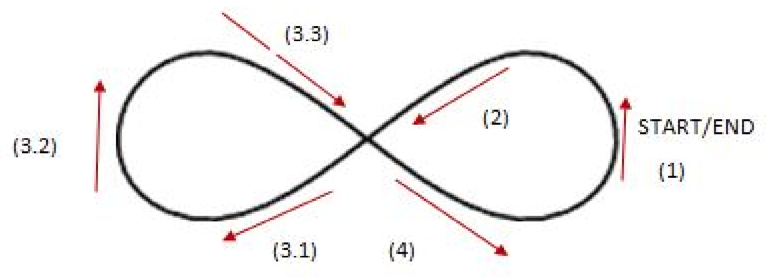
The robot is first oriented at a forty five degree angle so the rear (or front, depending on the orientation) is facing the parking stall. The robot then moves backwards a fixed distance and rotates again so it is straight. While it is in the stall, it constantly reads values from the ADC and waits for a specific parallel_park command from the transmitter to exit the stall. Once it receives the command, it does the above actions in reverse to return back to its original spot.

- three_point_turn

The robot goes forward a short distance, then makes a turn to the left until it is at a 90 degree angle. It then backs up an appropriate distance, then makes the same 90 degree left turn. It ends up at the original spot, except in reverse orientation. Like rotate_180, it returns the reverse orientation compared to what it was given.

-figure_Eight

The robot traces a figure eight shape. This is done by imagining that the robot is in the middle bottom spot of an 8 (1). The robot traces a half circle using one motor initialization (2), traces a full circle using an opposite motor initialization (3), then completes the first half circle by using the first motor initialization (4). The following figure illustrates the motion.



- Interrupt

The timer 0 interrupt is set up to be executed every 100us. This interrupt utilizes pulse width modulation to power the motors by comparing a value (e.g LeftMotor_Forward) that is between 0 and 100, to pwmcount, an variable that acts as an incrementing counter that is reset once it reaches 100.

- Adjust Distance function

The overall purpose of this function is to compare the voltage that is read from the left and right receivers, and turn on either the left or right motor accordingly until the robot is adjusted to the desired distance and orientation. To make sure that the rover is always facing the transmitter head-on, it starts by subtracting the left voltage from the right and comparing the difference between them to a preset ``buffer`` value, so that adjustments are only made if the difference is greater than the buffer. Then depending on whether the difference is positive or negative, the left or right motor is turned on until the difference is less than the buffer. Then it compares the voltage from only the right receiver to the desired voltage, and if the difference is greater than the buffer then the motors are turned on to move it either closer or farther away, depending on the difference. Inside each loop to adjust the distance and orientation, it checks continuously whether a command has been sent from the receiver, and if so then it exits the loop to execute that command. Once all the adjustments for distance and orientation have been made, it exits the function and returns to main.

- Main Function

Multiple initialization variables that affect the robot's movement are initialized here.The default standards for our robot is as follows. The default orientation is such that the front of the car (end with the SPI port of the LP51B is sticking out) is facing the transmitter. The pwm_val is set to 100 (defined as TURBO_SPEED) so the robot adjusts its distance with maximum motor power. The minimum voltage for a magnetic field to register, and the voltage for a reasonable starting distance, were determined by setting up the rover at an appropriate distance and measuring the field strength. They were set to 35mv for minimum and 1.2 V for distance. The main function then enters an infinite loop. It first reads a value from the channel 0 of the ADC and tries to determine if the transmitter is sending a message. If it has, it compares the message received to several predefined commands that are defined at the top of the file. If it matches, it executes the corresponding command. If there are no commands sent, it then reads the second channel of the ADC in order to determine whether or not to turn on or turn off the photosensitive headlights at the front of the robot. Finally, the loop calls adjust_distance to check whether the robot is at the required fixed distance from the transmitter.

- *Robot_Transmitter.c*

We modified a motor control program which used interrupts to output a square wave with a 50% duty cycle at 15.1 KHz, the value that we calculated to be the resonant frequency of our LC circuit. To send commands to the robot, we constantly read from 6 buttons and when one one was pressed, we would transmit an 8 bit binary number that corresponded to the command and button. We transmitted data by enabling and disabling the interrupt to toggle the magnetic field on and off in the order dictated by the binary number. We experimented with transmission timing using the transmitter and receiver circuits to settle on an appropriate baud rate. In order to account for the extra time taken by the interrupt when sending a one-bit, we had our wait_one_bit function wait twice as long for a zero-bit as it did for a one-bit.

Our transmitter was constructed using an H-Bridge and was powered with 9V. In order to achieve a 9V signal out of our microcontroller, we used BJTs to allow 9V at the gates of the MOSFETs. The H-Bridge was fed by two opposing square waves and fed into an LC circuit with an inductance of approximately 1.15mH and a capacitance of 0.1uF. By having the square wave oscillate at 15.1kHz, we were able to have our transmitter in resonance and achieve a voltage across the inductor of over 150V. The high voltage across the inductor allowed a voltage to be induced in the receiver's tank circuit from a considerable distance.

Our receiver was built using a tank circuit [4] to respond to the magnetic field from the transmitter. The output of the tank circuit was then fed into an inverting amplifier with a gain of 20 to increase the voltage reading. This result was fed into a peak detector, allowing the ADC to get accurate readings of the voltage corresponding to the field strength at that particular location. We used a small valued capacitor (.01uF) in the peak detector in order to have the circuit respond quickly to changes in the field when transmitting. This was necessary in order to receive accurate binary data in a short period of time. The receiver was composed of two of these tank circuits. By positioning the inductors on the left and right ends of the robot, we could detect if it was not aligned with the transmitter(as one inductor would be closer than the other, and thus, have a high voltage), and respond accordingly.

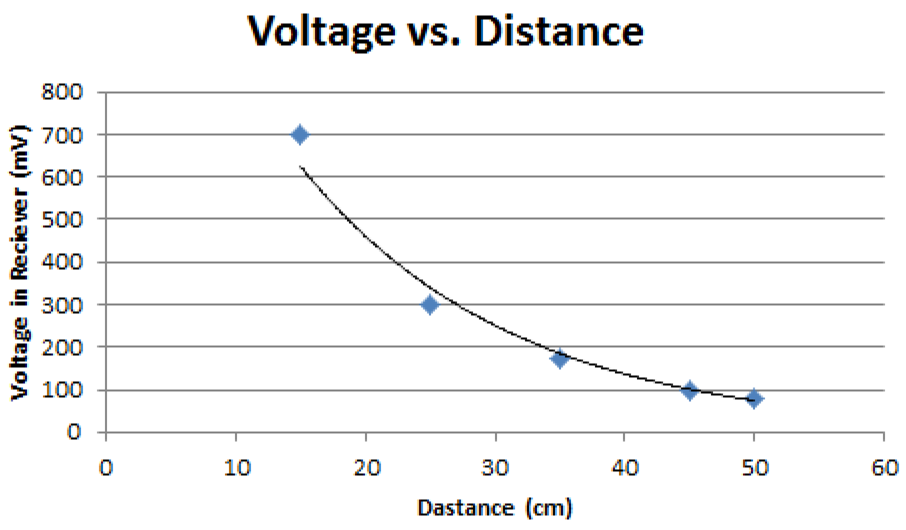
A voltage divider was created with the photoresistor in conjugation with a 20k resistor. This voltage was read by the ADC, to detect when to turn on the headlights.

To produce our turning signals, we used a .47uF capacitor and two 820 k resistors, to configure a 555 timer in astable mode. The output was wired to the LED. The capacitor and resistor values were chosen to greatly reduce the frequency of the output, to make the blinking of the LED obvious. The reset pin of the 555 was wired to a microcontroller port, which continuously fed a '0', until the light was to be turned on, when it sent a '1'.

3.7 Solution Assessment

Once our robot was completed we performed tests to ensure that it performed at the expected level. We tested the robot's range using a power supply and using batteries to examine its maximum range. To test the various functions that we had, we simply ran them and ensured that they had the expected result. We set up a parking space to test the parallel park, had the robot do a 180, a 3-point turn, and a figure eight to ensure that all of the functions worked as expected. We also ensured that we had set our buffers accurately by testing that the robot responded to movements of the transmitter, but did not react when there was no movement of the transmitter. We found that our robot's performance tended to vary with battery life and we often had to adjust delays and buffer values to calibrate for the level of battery life.

One weakness of our robot was the traction on the wheels. We found that the rubber did not have the right amount of traction, often slipping when placed on dusty or slippery surfaces like the floor. In addition, the performance of the robot seemed to vary on a daily basis, depending on the charge of the battery and whether components of the circuit had move around while being transported. On the other hand, one of the strengths of our project was that the transmission worked consistently. Also, the receiver code was set up to be readable and could easily be modified to fit any situation.



4.0 LIFE-LONG LEARNING

Over the past three weeks, we encountered numerous obstacles throughout the process of putting together our project. Through these obstacles, each member gained valuable experience, improving both their software and hardware skills.

One specific challenge we faced was adjusting the code to account for variations in our hardware. We needed to wait until the robot was fully functioning to be able to properly implement time delays needed for transmitting data and turning on the motors to complete certain tasks. The motors also changed speed depending on the charge of the batteries, which would affect the accuracy of the time delays day-to-day. We were unable to test the majority our code until the robot was working, and then it was sometimes hard to tell whether a problem was in the code or the hardware. However, after numerous alterations and some minor setbacks, we managed to implement a fully functioning program.

As a group, we learned essential skills which will be used for future projects. These skills include working with pulse-width modulation, integration of software and hardware, and the implementation of new components such as a transmitters, receivers, and motors. Through the opportunities to practice particular skills that this project presented, each individual was also able to improve his or her software and hardware skills, such as proper documentation, integration of code, and circuit design.

5.0 CONCLUSION

For this project, our team worked together to design and implement a semi-autonomous rover that would remain an adjustable distance away from a transmitter at all times. It would perform certain actions such as a parallel park or three-point turn on command. The main hardware components that we used to complete this task were two DC motors with an H-bridge, a receiver and transmitter using three capacitors and inductors and an amplifier, and two LP51B PCBs made from our previous project. The main software components were the C programming language, the 8051 assembly programming language, and the concept of modulation. Our end goal was that the rover would be powered by 9V batteries, would remain a certain distance at all times from the transmitter, and would perform the ordered commands accurately without needing adjustment.

The main problems that our group faced were the difference in performance of the rover depending on the charge of the batteries and the orientation of the inductors in the circuit, as well as the loss of traction of the wheels on the floor of the lab. To address those problems, we needed to implement a few strategies for testing the rover. The first strategy involved careful testing of all of our batteries with the multimeter before every use, and making sure that they were fully charged before we used them in our rover. The second strategy meant being very careful with how our rover was stored at the end of every day, to make sure that no wires were bent or twisted out of place. Finally, to address the third problem we implemented as much of our testing and demonstration as possible on the counter of the workbench, where the wheels had proper traction. When that wasn't possible, for example with our figure eight, we were careful to dust off the area of the floor to give the most grip to our wheels. These strategies made it possible to reliably test our rover and remain confident in the consistency of its performance.

Overall, the project took approximately 75 hours over a period of three weeks to complete. In the end, it remained a certain distance and orientation at all times from the transmitter and performed the ordered commands accurately. You would turn it on by connecting the red power supply wire to the breadboard. Then it would follow the transmitter at a preset distance which could be adjusted to different lengths by pressing either the move closer or move farther buttons, and it would orient itself so it was always parallel to the transmitter. We had six different buttons on the transmitter so the user could select between six different commands: move closer, move farther, parallel park, three-point turn, 180 degree turn, and figure eight. If the rover moved into a dim area, the headlights at the front would turn on automatically. Our group was able to have the rover stay the proper distance and orientation from the transmitter at all times, and have it do the commands all without incident, thus achieving our goal.

6.0 REFERENCES

[1] MCP3004/3008 2.7V 4-Channel/8-Channel 10-Bit A/D Converters with SPI™ Serial Interface. (n.d.). *MicroChip*. Retrieved February 28, 2014, from http://ww1.microchip.com/downloads/en/DeviceDoc/21295C.pdf

[2] TexasInstruments. (n.d.). LM324-N Low Power Quad Operational Amplifiers.*Texas Instruments*. Retrieved February 24, 2014, from http://www.ti.com/lit/ds/symlink/lm124-n.pdf

[3] NE556 Dual Precision Timers. (n.d.).*Electronic Components Datasheet Search*. Retrieved March 28, 2014, from http://www.alldatasheet.com/datasheet-pdf/pdf/17973/PHILIPS/NE556.html

[4] AN232 Low-Frequency Magnetic Transmitter Design. (n.d.). *MicroChip*. Retrieved March 24, 2014, from http://ww1.microchip.com/downloads/en/AppNotes/00232B.pdf

7.0 BIBLIOGRAPHY

Atmel. (n.d.). 8-bit Flash Microcontroller with 24K/32K bytes Program Memory.*Atmel*. Retrieved February 14, 2014, from http://www.atmel.ca/Images/doc3722.pdf

NationalSemiconductors. (n.d.). LM78XX Series Voltage Regulators. National Semiconductor. Retrieved March 15, 2014, from http://www.hep.upenn.edu/SNO/daq/parts/lm7815.pdf

STP12PF06 STripFET II Power Mosfet. (n.d.). *STMicroelectrics*. Retrieved February 28, 2014, from http://www.st.com/web/en/resource/technical/document/datasheet/CD00050425.pdf

STP16NF06 STripFET II Power MOSFET. (n.d.). *STMicroelectronics*. Retrieved February 28, 2014, from http://www.st.com/web/en/resource/technical/document/datasheet/CD00002501.pdf

TexasInstruments. (n.d.). LM555 Timer.*Texas Instruments*. Retrieved February 25, 2014, from http://www.ti.com.cn/cn/lit/ds/symlink/lm555.pdf

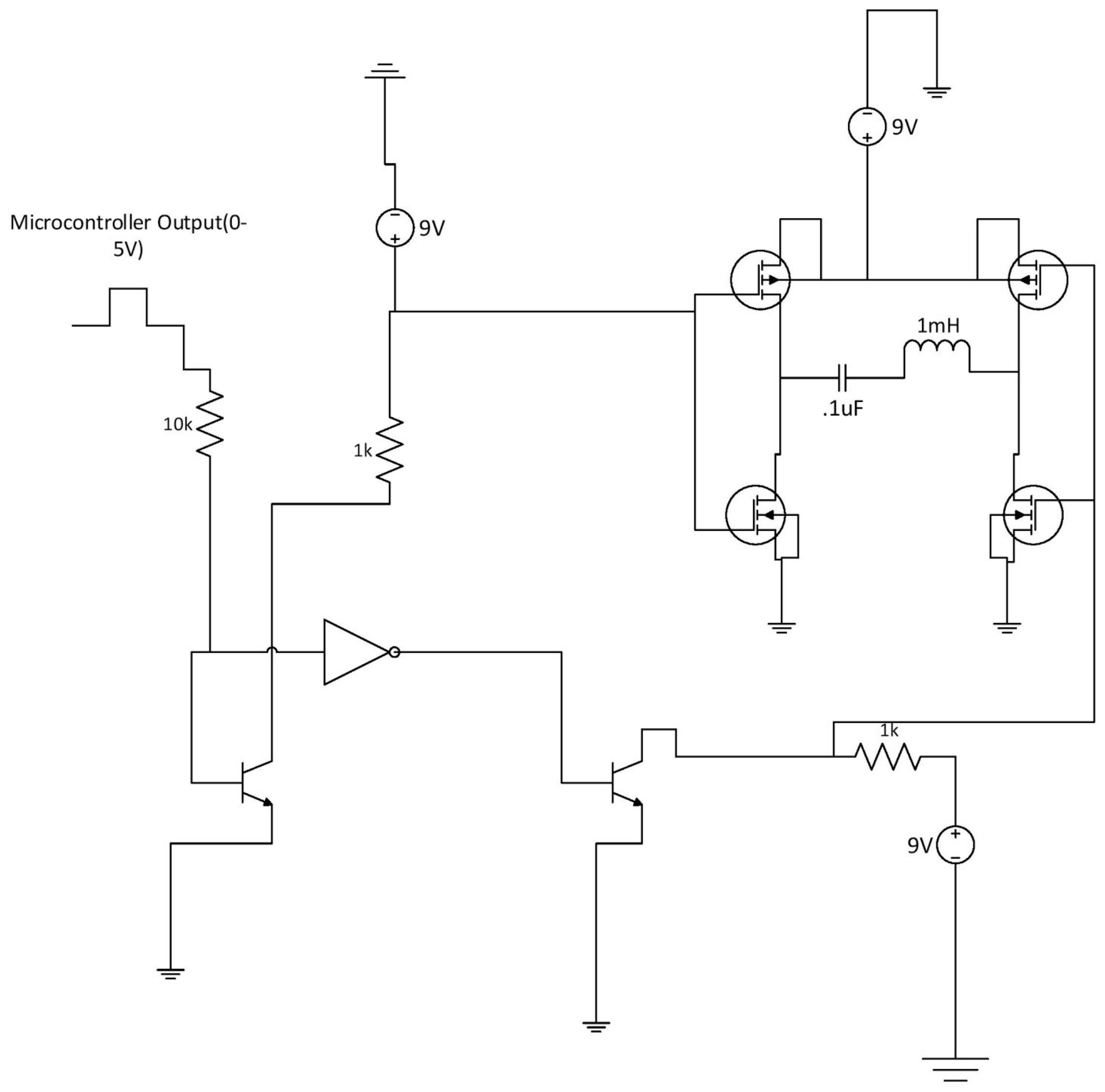
8.0 APPENDICES

Our project consists of five appendices, three are circuit diagrams and two are .c files

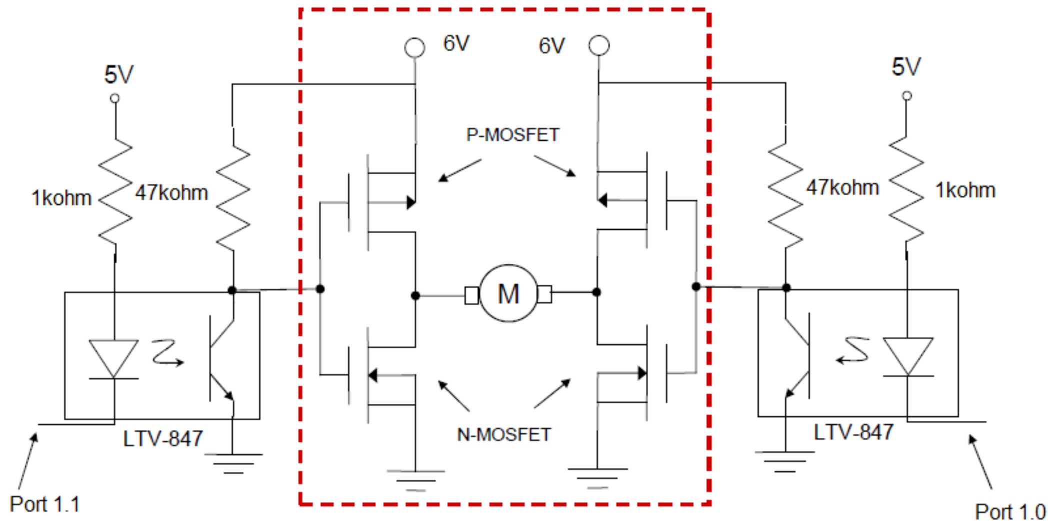
1. Receiver Circuit Diagram
2. Transmitter Circuit Diagram
3. Motor (H-Bridge) Circuit Diagram
4. Robot_Transmitter.c
5. Robot_Receiver.c

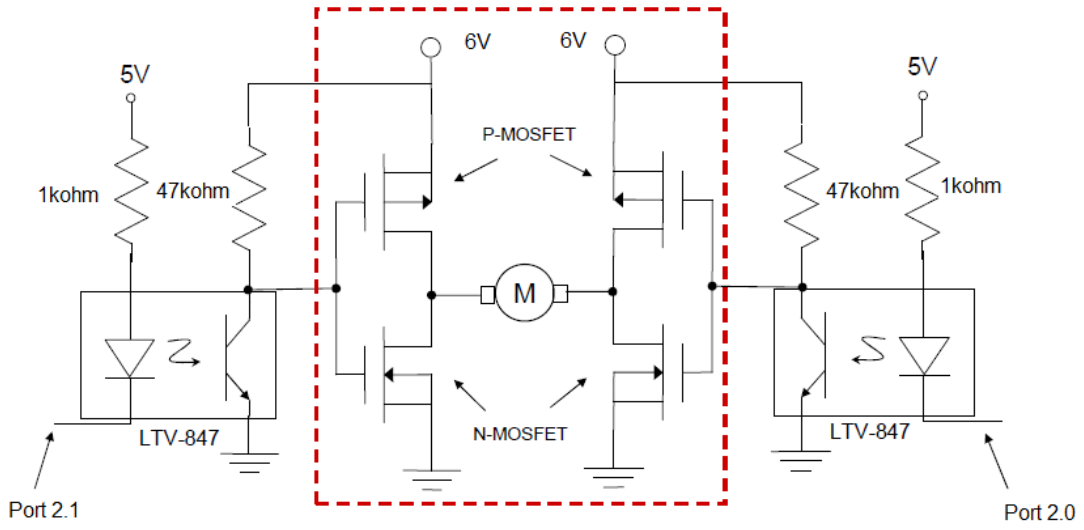
1. Receiver Circuit Diagram

2. Transmitter Circuit Diagram



3. Motor (H-Bridge) Circuit Diagram





4. Robot_Transmitter.c

```
#include <stdio.h>
#include <at89lp51rd2.h>

// ~C51~

#define CLK 22118400L
#define BAUD 115200L
#define BRG_VAL (0x100-(CLK/(32L*BAUD)))

#define MOVE_CLOSER 0B_0101_0101
#define MOVE_FURTHER 0B_1010_1010
#define ONE_EIGHTY 0B_0011_0011
#define PARALLEL_PARK 0B_1100_1100
#define THREE_POINT_TURN 0B_0110_0110
#define FIGURE_EIGHT 0B_1001_1001

//We want timer 0 to interrupt every 30 microseconds ((1/30000Hz)=30)
#define FREQ 30000L
#define TIMER0_RELOAD_VALUE (65536L-(CLK/(12L*FREQ)))

//These variables are used in the ISR
int onOff;

unsigned char _c51_external_startup(void)
{
       // Configure ports as a bidirectional with internal pull-ups.
       P0M0=0; P0M1=0B_0100_0001;
       P1M0=0; P1M1=0;
       P2M0=0; P2M1=0;
       P3M0=0; P3M1=0;
       AUXR=0B_0001_0001; // 1152 bytes of internal XDATA, P4.4 is a general purpose I/O
       P4M0=0; P4M1=0;

    // Initialize the serial port and baud rate generator
    PCON|=0x80;
       SCON = 0x52;
    BDRCON=0;
    BRL=BRG_VAL;
    BDRCON=BRR|TBCK|RBCK|SPD;

       // Initialize timer 0 for ISR 'pwmcounter()' below
       TR0=0; // Stop timer 0
       TMOD=0x01; // 16-bit timer
       // Use the autoreload feature available in the AT89LP51RB2
       // WARNING: There was an error in at89lp51rd2.h that prevents the
       // autoreload feature to work.  Please download a newer at89lp51rd2.h
       // file and copy it to the crosside\call51\include folder.
       TH0=RH0=TIMER0_RELOAD_VALUE/0x100;
       TL0=RL0=TIMER0_RELOAD_VALUE%0x100;
       TR0=1; // Start timer 0 (bit 4 in TCON)
       ET0=1; // Enable timer 0 interrupt
       EA=1;  // Enable global interrupts

       onOff = 0;

    return 0;
}

void waitOneBit (int val)
{
       if (val == 0){
              _asm
              ;For a 22.1184MHz crystal one machine cycle
```

```
                ;takes 12/22.1184MHz=0.5425347us
                mov R2, #2
         L2:    mov R1, #150
         L3:    mov R0, #150
         L4:    djnz R0, L4 ; 2 machine cycles-> 2*0.5425347us*184=200us
             djnz R1, L3 ; 200us*250=0.05s
             djnz R2, L2
             ret
     _endasm;
     }
     else if (val == 1){
                _asm
                ;For a 22.1184MHz crystal one machine cycle
                ;takes 12/22.1184MHz=0.5425347us
                mov R1, #150
         B1:    mov R0, #150
         B2:    djnz R0, B2 ; 2 machine cycles-> 2*0.5425347us*184=200us
             djnz R1, B1 ; 200us*250=0.05s
             ret
     _endasm;
     }
}

void sendMessage(char dat){
       int i;
       EA = 0;
       waitOneBit(0);
       for(i = 0; i < 8; i++){
              EA = dat >> 7;
              dat = dat << 1;
              waitOneBit(EA);
       }
       EA = 0;
       waitOneBit(0);
       waitOneBit(0);
       EA = 1;
       return;
}


// Interrupt 1 is for timer 0.  This function is executed every time
// timer 0 overflows: 100 us.
void outputFreq (void) interrupt 1
{
       if (onOff == 0){
              onOff = 1;
              P0_0 = 1;
              P0_6 = 0;
       }
       else if (onOff == 1){
              onOff = 0;
              P0_0 = 0;
              P0_6 = 1;
       }
}

void main (void)
{

       while(1){
              if (P1_4 == 1)
                     sendMessage(MOVE_CLOSER);
              else if (P1_5 == 1)
                     sendMessage(MOVE_FURTHER);
              else if (P1_6 == 1)
                     sendMessage(ONE_EIGHTY);
                else if (P1_7 == 1)
                        sendMessage(PARALLEL_PARK);
                else if (P4_1 == 1)
                        sendMessage(THREE_POINT_TURN);
                else if (P3_2 == 1)
                        sendMessage(FIGURE_EIGHT);
        }
}
```

5. Robot_Receiver.c

```
#include <stdio.h>
#include <stdlib.h>
#include <at89lp51rd2.h>

// ~C51~

// definitions for motor components
#define DESIRED_VOLTAGE 1.2
#define MINIMUM 25
#define SLOW_SPEED 50
#define MEDIUM_SPEED 75
#define TURBO_SPEED 100
#define FORWARD 1
#define BACKWARD 0
#define OFF 0
#define ON 1
#define NORMAL_ORIENTATION 1
#define REVERSE_ORIENTATION -1
#define LEFT_TURN_SIG P1_0
#define RIGHT_TURN_SIG P3_6


//initialization definitions
#define CLK 22118400L
#define BAUD 115200L
#define BRG_VAL (0x100-(CLK/(32L*BAUD)))

//timer 0 will interrupt every 100ms
#define FREQ 10000L
#define TIMER0_RELOAD_VALUE (65536L-(CLK/(12L*FREQ)))

//command definitions
#define MOVE_CLOSER 0B_0101_0101
#define MOVE_FURTHER 0B_1010_1010
#define ONE_EIGHTY 0B_0011_0011
#define PARALLEL_PARK 0B_1100_1100
#define THREE_POINT_TURN 0B_0110_0110
#define FIGURE_EIGHT 0B_1001_1001

//volatile variables for the interrupt
//Names are based on Normal Orientation
volatile int pwmcount;
volatile int LeftMotor_Forward;
volatile int LeftMotor_Backward;
volatile int RightMotor_Forward;
volatile int RightMotor_Backward;

/**
=====================================================================
                             Initialization
=====================================================================
**/


unsigned char _c51_external_startup(void)
{
       // Configure ports as a bidirectional with internal pull-ups.
       P0M0=0; P0M1=0;
       P1M0=0; P1M1=0;
       P2M0=0; P2M1=0;
       P3M0=0; P3M1=0;
       AUXR=0B_0001_0001; // 1152 bytes of internal XDATA, P4.4 is a general purpose I/O
```

```
        P4M0=0; P4M1=0;

    //User built-in baud rate generator instead of timer to generate the clock
    //for the serial port

    PCON|=0x80;
        SCON = 0x52;
    BDRCON=0;
    BRL=BRG_VAL;
    BDRCON=BRR|TBCK|RBCK|SPD;

    // Initialize timer 0 for ISR 'pwmcounter()' below
        TR0=0; // Stop timer 0
        TMOD=0B_00010001; // 16-bit timer for timer 0 and 1
        // Use the autoreload feature available in the AT89LP51RB2
        TH0=RH0=TIMER0_RELOAD_VALUE/0x100;
        TL0=RL0=TIMER0_RELOAD_VALUE%0x100;
        TR0=1; // Start timer 0 (bit 4 in TCON)
        ET0=1; // Enable timer 0 interrupt
        ET0 = 0; // Disable timer 0 -Only set it when we want it to move
        EA=1;
    pwmcount=0;
    RIGHT_TURN_SIG = OFF;
    LEFT_TURN_SIG = OFF;

    return 0;
}


/**
====================================================================
                             SPI Functions
====================================================================
**/


void SPIWrite(unsigned char value)
{
        SPSTA&=(~SPIF); // Clear the SPIF flag in SPSTA
        SPDAT=value;
        while((SPSTA & SPIF)!=SPIF); //Wait for transmission to end
}

// Read 10 bits from the MCP3004 ADC converter
unsigned int GetADC(unsigned char channel)
{
        unsigned int adc;

        // initialize the SPI port to read the MCP3004 ADC attached to it.
        SPCON&=(~SPEN); // Disable SPI
        SPCON=MSTR|CPOL|CPHA|SPR1|SPR0|SSDIS;
        SPCON|=SPEN; // Enable SPI

        P1_4=0; // Activate the MCP3004 ADC.
        SPIWrite(channel|0x18);      // Send start bit, single/diff* bit, D2, D1, and D0 bits.
        for(adc=0; adc<10; adc++); // Wait for S/H to setup
        SPIWrite(0x55); // Read bits 9 down to 4
        adc=((SPDAT&0x3f)*0x100);
        SPIWrite(0x55);// Read bits 3 down to 0
        P1_4=1; // Deactivate the MCP3004 ADC.
        adc+=(SPDAT&0xf0); // SPDR contains the low part of the result.
        adc>>=4;

        return adc;
}
```

```
float voltage (unsigned char channel)
{
       return ( (GetADC(channel)*4.84)/1023.0 ); // VCC=4.84V (measured)
}


/**
====================================================================
                           Bit Bang Functions
====================================================================
**/

void wait_one_and_half_bit_time(){
                      _asm
              mov R2, #3
       N2:    mov R1, #150
       N3:    mov R0, #150
       N4:    djnz R0, N4
           djnz R1, N3
           djnz R2, N2
           ret
    _endasm;
}


void wait_bit_time (){

              _asm
              mov R2, #2
       Q2:    mov R1, #150
       Q3:    mov R0, #150
       Q4:    djnz R0, Q4
           djnz R1, Q3
           djnz R2, Q2
           ret
    _endasm;

}

unsigned char rx_byte ( int min ){
       unsigned char j, val;
       int v;

       val=0;
       wait_one_and_half_bit_time();

       for(j=0; j<8; j++)
       {
              v=GetADC(0);
              val|=(v>min)?(0x01<<j):0x00;
              wait_bit_time();
       }

       wait_one_and_half_bit_time();
       return val;
}


/**
====================================================================
                             Different Delays
====================================================================
**/
// For a 22.1184MHz crystal one machine cycle takes 12/22.1184MHz=0.5425347us

void startUpDelay(){
        _asm

            mov R2, #40
        A3:    mov R1, #248
        A2:    mov R0, #184
        A1:    djnz R0, A1
            djnz R1, A2
            djnz R2, A3
            ret
    _endasm;
}

void rotateDelay(){
        _asm

            mov R2, #16
        H3:    mov R1, #232
        H2:    mov R0, #184
        H1:    djnz R0, H1
            djnz R1, H2
            djnz R2, H3
            ret
    _endasm;
}

void SlightDelay(){
        _asm
            mov R2, #2
        E3:    mov R1, #248
        E2:    mov R0, #184
        E1:    djnz R0, E1
            djnz R1, E2
            djnz R2, E3
            ret
    _endasm;
}


void motorDelay(){
        _asm
            mov R2, #2
        L3:    mov R1, #248
        L2:    mov R0, #184
        L1:    djnz R0, L1
            djnz R1, L2
            djnz R2, L3
            ret
    _endasm;
}

void motorsDelay(){
        _asm
           mov R2, #4
        M3:    mov R1, #248
        M2:    mov R0, #184
        M1:    djnz R0, M1
            djnz R1, M2
            djnz R2, M3
            ret
    _endasm;
}

void fortyFive(){
        _asm
            mov R2, #5
        K3:    mov R1, #253
```

```
       K2:    mov R0, #184
       K1:    djnz R0, K1
           djnz R1, K2
           djnz R2, K3
           ret
    _endasm;
}

void distDelay(){
       _asm
           mov R2, #27
       Y3:    mov R1, #250
       Y2:    mov R0, #184
       Y1:    djnz R0, Y1
           djnz R1, Y2
           djnz R2, Y3
           ret
    _endasm;
}

void threeQuartSec(){
       _asm
           mov R2, #5
       W3:    mov R1, #248
       W2:    mov R0, #184
       W1:    djnz R0, W1
           djnz R1, W2
           djnz R2, W3
           ret
    _endasm;
}

void arcTurnDelay(){
       _asm
           mov R2, #22
       O3:    mov R1, #241
       O2:    mov R0, #184
       O1:    djnz R0, O1
           djnz R1, O2
           djnz R2, O3
           ret
    _endasm;
}

void threePtBackDelay(){
       _asm
           mov R2, #21
       Z3:    mov R1, #248
       Z2:    mov R0, #184
       Z1:    djnz R0, Z1
           djnz R1, Z2
           djnz R2, Z3
           ret
    _endasm;
}


/**
===================================================================
                      Motor Initialization Commands
===================================================================
**/

void init_backwards(int orientation){
       if(orientation == NORMAL_ORIENTATION){
              LeftMotor_Forward = OFF;                    RightMotor_Forward = OFF;
```

```
		LeftMotor_Backward = MEDIUM_SPEED;	RightMotor_Backward = MEDIUM_SPEED;
	}
	if(orientation == REVERSE_ORIENTATION){
		LeftMotor_Forward = MEDIUM_SPEED;	RightMotor_Forward = MEDIUM_SPEED;
		LeftMotor_Backward = OFF;		RightMotor_Backward = OFF;
	}
}

void init_Fastbackwards(int orientation){
	if(orientation == NORMAL_ORIENTATION){
		LeftMotor_Forward = OFF;		RightMotor_Forward = OFF;
		LeftMotor_Backward = TURBO_SPEED;	RightMotor_Backward = TURBO_SPEED;
	}
	if(orientation == REVERSE_ORIENTATION){
		LeftMotor_Forward = TURBO_SPEED;	RightMotor_Forward = TURBO_SPEED;
		LeftMotor_Backward = OFF;		RightMotor_Backward = OFF;
	}
}

void init_forwards(int orientation){
	if(orientation == NORMAL_ORIENTATION){
		LeftMotor_Forward = MEDIUM_SPEED;	RightMotor_Forward = MEDIUM_SPEED;
		LeftMotor_Backward = OFF;		RightMotor_Backward = OFF;
	}
	if(orientation == REVERSE_ORIENTATION){
		LeftMotor_Forward = OFF;		RightMotor_Forward = OFF;
		LeftMotor_Backward = MEDIUM_SPEED;	RightMotor_Backward = MEDIUM_SPEED;
	}
}

void init_45_arcTurn(int orientation){
	if(orientation == NORMAL_ORIENTATION){
		LeftMotor_Forward = SLOW_SPEED;			RightMotor_Forward =
TURBO_SPEED;
		LeftMotor_Backward = OFF;			RightMotor_Backward = OFF;
	}
	if(orientation == REVERSE_ORIENTATION){
		LeftMotor_Forward = OFF;			RightMotor_Forward = OFF;
		LeftMotor_Backward = TURBO_SPEED;		RightMotor_Backward = SLOW_SPEED;
	}
}

void init_180(){
	LeftMotor_Forward = MEDIUM_SPEED;		RightMotor_Forward = OFF;
	RightMotor_Backward = MEDIUM_SPEED;		LeftMotor_Backward = OFF;
}

void init_45_CW(){
	LeftMotor_Forward = MEDIUM_SPEED;		RightMotor_Forward = OFF;
	LeftMotor_Backward = OFF;			RightMotor_Backward = MEDIUM_SPEED;
}

void init_45_CCW(){
	LeftMotor_Forward = OFF;			RightMotor_Forward = MEDIUM_SPEED;
	LeftMotor_Backward = MEDIUM_SPEED;	RightMotor_Backward = OFF;
}

void init_figEight_14(){
	LeftMotor_Forward = SLOW_SPEED;			RightMotor_Forward = TURBO_SPEED;
	LeftMotor_Backward = OFF;			RightMotor_Backward = OFF;
}

void init_figEight_23(){
	LeftMotor_Forward = TURBO_SPEED;		RightMotor_Forward = SLOW_SPEED;
	LeftMotor_Backward = OFF;			RightMotor_Backward = OFF;
}
```

```
void init_Stop(){
       P0_2 = 0;                                                    P0_3 = 0;
       P1_3 = 0;                                                    P1_2 = 0;
}

/**
====================================================================
                             Different Commands
====================================================================
**/

float move_closer( float distance ){
       distance = distance * (2.0/3.0);

       return distance;
}


float move_further( float distance ){

       distance = distance * (3.0/2.0);

       return distance;
}

int rotate_180(int orientation){

       init_180();
       ET0 = ON;      rotateDelay();        ET0 = OFF;
       init_Stop();

       orientation *= -1;
       return orientation;
}

void parallel_park(int orientation){

       unsigned char receivedData = 5;
       float receive;
       unsigned int min = MINIMUM;

       init_45_CCW();
       ET0 = ON;      fortyFive();          ET0 = OFF;     init_Stop();   SlightDelay();

       if(orientation == NORMAL_ORIENTATION)
              RIGHT_TURN_SIG = ON;
       else
              LEFT_TURN_SIG = ON;

       init_backwards(orientation);
       ET0 = ON;      distDelay();   ET0 = OFF;     init_Stop();   SlightDelay();
       LEFT_TURN_SIG = OFF;          RIGHT_TURN_SIG = OFF;

       init_45_CW();
       ET0 = ON;      fortyFive();   ET0 = OFF;     init_Stop();   SlightDelay();


       while(1){
              receive = GetADC(0);
                     if( receive <= MINIMUM ){
                            receivedData = rx_byte(min);
                            if (receivedData == PARALLEL_PARK)
                                   break;
                     }
       }
```

```
        init_45_CCW();
        ET0 = ON;      fortyFive();   ET0 = OFF;     init_Stop();   SlightDelay();

        if(orientation == NORMAL_ORIENTATION)
               LEFT_TURN_SIG = ON;
        else
               RIGHT_TURN_SIG = ON;

        init_forwards(orientation);
        ET0 = ON;      distDelay();   ET0 = OFF;     init_Stop();   SlightDelay();
        LEFT_TURN_SIG = OFF;          RIGHT_TURN_SIG = OFF;

        init_45_CW();
        ET0 = ON;      fortyFive();   ET0 = OFF;     init_Stop();   SlightDelay();
}

int three_point_turn(int orientation){


        init_forwards(orientation);
        ET0 = ON;      threeQuartSec();      ET0 = OFF;     init_Stop();

        init_45_arcTurn(orientation);

        if(orientation == NORMAL_ORIENTATION)
               LEFT_TURN_SIG = ON;
        else
               RIGHT_TURN_SIG = ON;

        ET0 = ON;      arcTurnDelay();              ET0 = OFF;     init_Stop();   SlightDelay();
        LEFT_TURN_SIG = OFF;          RIGHT_TURN_SIG = OFF;


        init_Fastbackwards(orientation);
        ET0 = ON;      threePtBackDelay();   ET0 = OFF;     init_Stop();   SlightDelay();

        init_45_arcTurn(orientation);

        if(orientation == NORMAL_ORIENTATION)
               LEFT_TURN_SIG = ON;
        else
               RIGHT_TURN_SIG = ON;

        ET0 = ON;      arcTurnDelay();              ET0 = OFF;     init_Stop();
        LEFT_TURN_SIG = OFF;                 RIGHT_TURN_SIG = OFF;

        init_forwards(orientation);
        ET0 = ON;      threeQuartSec();      ET0 = OFF;     init_Stop();

        orientation *= -1;
        return orientation;
}

void figure_Eight(){

        init_figEight_14();
        ET0 = ON;      LEFT_TURN_SIG = ON;
        arcTurnDelay();              arcTurnDelay();
        ET0 = OFF;     LEFT_TURN_SIG = OFF;

        init_figEight_23();
        ET0 = ON;      RIGHT_TURN_SIG = ON;
        arcTurnDelay();       arcTurnDelay();              arcTurnDelay();       arcTurnDelay();
        ET0 = OFF;     RIGHT_TURN_SIG = OFF;

        init_figEight_14();
        ET0 = ON;      LEFT_TURN_SIG = ON;
        arcTurnDelay();               arcTurnDelay();
        ET0 = OFF;     LEFT_TURN_SIG = OFF;

        init_Stop();
}


        float distance = DESIRED_VOLTAGE;
        int pwm_val = TURBO_SPEED;
        int min = MINIMUM;


/**
====================================================================
                               Interrupt
====================================================================
**/

// timer 0 overflows: 100 us.
void pwmcounter (void) interrupt 1
{
        if(++pwmcount>99) pwmcount=0;
        P1_3=(LeftMotor_Forward>pwmcount)?1:0;
        P1_2=(LeftMotor_Backward>pwmcount)?1:0;
        P0_2=(RightMotor_Forward>pwmcount)?1:0;
        P0_3=(RightMotor_Backward>pwmcount)?1:0;
}


/**
====================================================================
                         Adjust Distance Function
====================================================================
**/

void moveMotor1( int pwm_val, int direction ){
        RightMotor_Forward = OFF;
        RightMotor_Backward = OFF;

        if( direction == FORWARD){
               LeftMotor_Forward = pwm_val;
               LeftMotor_Backward = OFF;
        }
        if( direction == BACKWARD){
               LeftMotor_Forward = OFF;
               LeftMotor_Backward = pwm_val;
        }
        ET0 = ON;
        motorDelay();
        ET0 = OFF;
        init_Stop();
}


void moveMotor2( int pwm_val, int direction ){

        LeftMotor_Forward = OFF;
        LeftMotor_Backward = OFF;

        if( direction == FORWARD){
               RightMotor_Forward = pwm_val;
               RightMotor_Backward = OFF;
        }
        if( direction == BACKWARD){
                RightMotor_Forward = OFF;
                RightMotor_Backward = pwm_val;
        }
        ET0 = ON;
        motorDelay();
        ET0 = OFF;
        init_Stop();
}

void moveMotors( int pwm_val, int direction ){

        if( direction == FORWARD){
                RightMotor_Forward = pwm_val;
                RightMotor_Backward = OFF;
                LeftMotor_Forward = pwm_val;
                LeftMotor_Backward = OFF;
        }
        if( direction == BACKWARD){
                RightMotor_Forward = OFF;
                RightMotor_Backward = pwm_val;
                LeftMotor_Forward = OFF;
                LeftMotor_Backward = pwm_val;
        }
        ET0 = ON;
        motorsDelay();
        ET0 = OFF;
        init_Stop();
}

void adjust_Distance(float distance, int pwm_val, int orientation)
{

        float receive;
        float bufferL = 0.15;
        float bufferT = 0.1;

        float distance0 = voltage(0);
        float distance1 = voltage(1);

        float tempDistance = distance0-distance1;


        while( tempDistance < (-bufferL) ){

                receive = GetADC(0);

                if( receive <= MINIMUM ){
                        break;
                }

                RightMotor_Forward = OFF;
                RightMotor_Backward = OFF;

                if(orientation == NORMAL_ORIENTATION){
                        LeftMotor_Forward = pwm_val;
                        LeftMotor_Backward = OFF;
                }
                else{
                        LeftMotor_Forward = OFF;
                        LeftMotor_Backward = pwm_val;
                }
                ET0 = ON;

                distance0 = voltage(0);
```

```
		distance1 = voltage(1);

		tempDistance = distance0-distance1;
	}

	ET0 = OFF;
	init_Stop();

	while( tempDistance > bufferL ){


		receive = GetADC(0);

		if( receive <= MINIMUM ){
			break;
		}

		LeftMotor_Forward = OFF;
		LeftMotor_Backward = OFF;

		if(orientation == NORMAL_ORIENTATION){
			RightMotor_Forward = pwm_val;
			RightMotor_Backward = OFF;
		}
		else{
			RightMotor_Forward = OFF;
			RightMotor_Backward = pwm_val;
		}
		ET0 = ON;

		distance0 = voltage(0);
		distance1 = voltage(1);

		tempDistance = distance0-distance1;
	}

	ET0 = OFF;
	init_Stop();

	while( distance0 > distance+bufferT){


		receive = GetADC(0);

		if( receive <= MINIMUM ){
			break;
		}

		if(orientation == NORMAL_ORIENTATION){
			RightMotor_Forward = OFF;
			RightMotor_Backward = pwm_val;
			LeftMotor_Forward = OFF;
			LeftMotor_Backward = pwm_val;
		}
		else{
			RightMotor_Forward = pwm_val;
			RightMotor_Backward = OFF;
			LeftMotor_Forward = pwm_val;
			LeftMotor_Backward = OFF;
		}
		ET0 = ON;

		distance0 = voltage(0);

		printf("distance0 is %u\n", distance0);
	}
```

```
        ET0 = OFF;
        init_Stop();

        while( distance0 < distance - bufferT ){


                receive = GetADC(0);

                if( receive <= MINIMUM ){
                        break;
                }

                if(orientation == NORMAL_ORIENTATION){
                        RightMotor_Forward = pwm_val;
                        RightMotor_Backward = OFF;
                        LeftMotor_Forward = pwm_val;
                        LeftMotor_Backward = OFF;
                }
                else{
                        RightMotor_Forward = OFF;
                        RightMotor_Backward = pwm_val;
                        LeftMotor_Forward = OFF;
                        LeftMotor_Backward = pwm_val;
                }
                ET0 = ON;

                distance0 = voltage(0);

                printf("distance0 is %u\n", distance0);
        }

        ET0 = OFF;
        init_Stop();

}


/**
====================================================================
                             Main Funcion
====================================================================
**/

void main ()
{

        float distance = DESIRED_VOLTAGE;
        int pwm_val = TURBO_SPEED;
        int direction = FORWARD;
        int orientation = NORMAL_ORIENTATION;

        unsigned int min = MINIMUM;
        unsigned char receivedData;

        startUpDelay();

        while(1)
        {

                float receive = GetADC(0);
                printf("Receive is - %f\n", receive*4.84/1023.0);

                if( receive <= MINIMUM ){
                        printf("In if statement\n");
                        receivedData = rx_byte(min);
                if( receivedData == MOVE_CLOSER)
                distance = move_closer(distance);

                else if (receivedData == MOVE_FURTHER)
                distance = move_further(distance);

                else if (receivedData == ONE_EIGHTY)
                        orientation = rotate_180(orientation);
                else if (receivedData == PARALLEL_PARK)
                         parallel_park(orientation);
                else if (receivedData == THREE_POINT_TURN)
                        orientation = three_point_turn(orientation);
                else if (receivedData == FIGURE_EIGHT)
                        figure_Eight();
        }

        if (GetADC(2) < 400){
                P2_0 = 0;
                P2_1 = 0;
        }
        else{
                P2_0 = 1;
                P2_1 = 1;
        }

        adjust_Distance(distance, pwm_val, orientation);
    }
}
```