Tactile Operated, Obstacle Avoiding, Motorized Wheelchair (TOM Chair)

11th April, 2021

Group 14's Team Members and Individual Participation:

Ashwin Mani – 25%

Nikhil Kumar Kuppa – 25%

Prathik Shetty – 25%

Sahil Raj Gupta – 25%

- Group 14's work was divided such that through the course duration, nobody felt overburdened. We're confident when we say all the four of us contributed 25% each, in our ways, without which the project wouldn't have reached this stage.

Table of Contents

1) <u>Background Information</u>

The ongoing COVID-19 Pandemic has demanded the attention of a huge portion of the hospital domestic staff to cater to the patient needs. Due to the exorbitant prices of motorized wheelchairs in India, hospitals have wheelchairs that usually require a nurse's help to navigate according to a patient's needs. A lot of times, patients want to operate their wheelchairs by themselves too. Thus, to primarily address these issues, we worked on developing a <u>Tactile operated, Obstacle avoiding, Motorized wheelchair</u>, or as we call it the *TOM Chair*. It is aimed at being safe and cost effective; perfect for the Indian market.

2) <u>User Manual</u>

Instruction Manual – TOM Chair

PLEASE READ THE INSTRUCTION MANUAL CAREFULLY BEFORE USING THE DEVICE TO ACHIEVE BEST POSSIBLE RESULTS

Introduction:

Touch operated, Obstacle avoiding, Motorized wheelchair - TOM chair, is built to help people (ages 6 and above) with special abilities; or patients who can at least move one part of their body, (a finger, toe, elbow, etc). Moving a single part of the body will be sufficient to control the device. The wheelchair contains safety mechanisms which try to ensure that the patient is not hit by any obstacles. This project is only a miniature prototype of a fully scaled, functional wheelchair with the intention of approaching a cost-effective solution for patients who need this, especially in places like some parts of India.

General Safety Precautions

- The user must wear the protective seatbelt provided, at all times.
- The user is recommended to use the portability of the joystick to suit their comfort, and is advised not to venture into operating the joystick with their non-dominant side under no supervision of a fully fit individual.
- The user must setup the joystick position before starting the device to avoid untoward consequences.
- When starting the vehicle, it is recommended to have the Battery State of Charge (SoC) be at 100.
- The user is also recommended to carry a pair of 9V batteries (not included) with them at all times in the event of battery discharge.
- The user is recommended to carry the parking blocks provided to place them against the wheel in the event of parking on an incline.

Initial Assembly:

Plug in two 9V batteries parallelly to the two battery ports available.

Principles of Operation:

- The user control module is portable over the wheelchair. So we can change its position to either side.
- The user will control the direction of the wheelchair using the joystick. The direction's corresponding sensor gets actuated, and obstacles will be detected accordingly.
- If the obstacle is in a range less than 10cms from the vehicle, all functions will be overridden, and the device will come to a halt. The user is then expected to direct the chair in another direction using the joystick.
- Through the ON time of the device, the LCD displays the battery state of charge, which direction the device is heading in, and in case an obstacle is detected, it sends out the alert message, "ALERT!".
- Terrains in which this wheelchair is expected to be functional are all types of floorings that include tiles, wood, marble, cement, or outdoors. The wheelchair is not expected to work on a very rocky or slippery terrain.

Control Instructions:

i) <u>Device Start Up</u>: Plug in the batteries and turn the kill-switch on.

ii) <u>Forward Drive</u>: Push the joystick towards the front direction.

iii) <u>Backward Drive</u>: Pull the joystick towards the back direction.

iv) <u>Left Drive</u>: Push the joystick towards the left direction.

v) <u>Right Drive</u>: Push the joystick towards the right direction.

vi) <u>Stop Motion</u>: Press the joystick downwards like a button.

vii) <u>Device Shut Down</u>: Turn off the kill-switch/plug the batteries off.

viii) <u>Parking</u>: This version of TOM Chair doesn't have braking systems developed yet, so we strongly recommend the user to shut the system down on a flat surface, or place the blocks, provided in the packaging, against the wheels to stop the vehicle from moving in the event of of parking on an incline.

Troubleshooting:

Problem	**Likely Cause(s)**	**Solution(s)**
Vehicle doesn't run	- Kill-switch is "OFF" - Loose battery connections - Weak batteries	- Turn the kill-switch "ON" - Check Battery Connector - Replace Batteries
Sensor Malfunction	- Physical Damage - Electrical Damage - Loose sliding slots	- Disconnect the wires and slide the sensor panel to replace it with a new sensor. - Fix them tightly to the walls of the chair
Slow Speed/ Jerky Movements	- Dust and grimes in the gearbox - Periodic wear of the wheels	- Clean and lubricate the gearbox - Wheel replacement

Display Issues	- Contrast needs to be set	- Rotate the knob next to the display to the point where you can comfortably see the display.

For Best Use:

- Plug the batteries off when not in use.
- Turn the kill-switch off when not in use.
- Replace the batteries periodically.
- Replace the wheels when worn out.
- Replace the motors periodically.
- Clean the processors built in chassis carefully and thoroughly to avoid dust and grime settling inside.
- Keep away from wet surfaces as much as possible.

WARNING! Changes or modifications to this unit not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

3) Hardware Section:

a) Block Diagram:

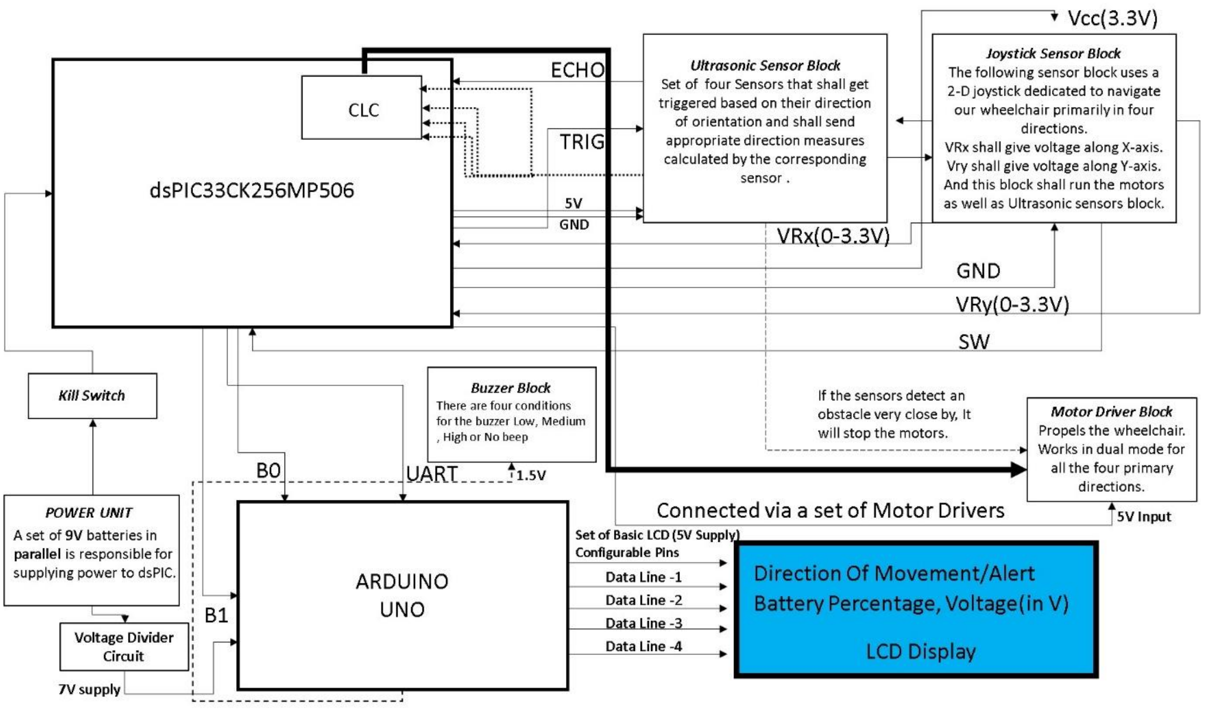
Vcc(3.3V)
ECHO
TRIG
5V
GND
CLC
dsPIC33CK256MP506
Ultrasonic Sensor Block
Set of four Sensors that shall get triggered based on their direction of orientation and shall send appropriate direction measures calculated by the corresponding sensor .
Joystick Sensor Block
The following sensor block uses a 2-D joystick dedicated to navigate our wheelchair primarily in four directions.
VRx shall give voltage along X-axis. Vry shall give voltage along Y-axis. And this block shall run the motors as well as Ultrasonic sensors block.
VRx(0-3.3V)
GND
VRy(0-3.3V)
SW
Kill Switch
Buzzer Block
There are four conditions for the buzzer Low, Medium , High or No beep
If the sensors detect an obstacle very close by, It will stop the motors.
Motor Driver Block
Propels the wheelchair. Works in dual mode for all the four primary directions.
B0
UART
1.5V
POWER UNIT
A set of 9V batteries in parallel is responsible for supplying power to dsPIC.
Connected via a set of Motor Drivers
5V Input
Set of Basic LCD (5V Supply) Configurable Pins
ARDUINO UNO
B1
Data Line -1
Data Line -2
Data Line -3
Data Line -4
Direction Of Movement/Alert
Battery Percentage, Voltage(in V)
LCD Display
Voltage Divider Circuit
7V supply

Fig. (3.a)

b) Functional Description of Each Block:

Input Section:

i. Joystick:

- Joystick is an analog sensor using High Speed 12-bit resolution ADC peripheral of the dsPIC with SAR technique for conversion.
- The voltage supplied to the sensor is 3.3V.
- The Range of values from VRx and VRy on moving the joystick is (0 – 4096).
- The system uses a Shared Core ADC with Fosc as its corresponding oscillator.
- Input Clock to the shared ADC core is divided into two source clock periods.
- The Sampling time for the core is set as 32*TADCore.
- A Common Software Trigger is used to trigger taking inputs from the joystick.
- The buffer register values of the ADC channels associated with the VRx and VRy is read to direct the wheelchair.

ii. Ultrasonic Sensors:

- It is the digital sensor responsible for detecting obstacles within an arc of variable radius of 15 degrees. The Trigger pin of the US sensor is given a small pulse of 10us and the following pulse is received from the echo pin before another pulse is sent across the Trig pin.
- The voltage supplied to the sensor module is 5v.
- Timer-1 of the dsPIC is used to measure the width of the echo pulse received which is basically the time taken for pulse to leave the trig pin, bounce back and received in the echo pin.
- The timer is switched on for the duration when echo pin is high, and it is stopped when the pin reads low.
- Practical accuracy for the range of this sensor in this project is taken to be 120cm.

Output Section:

iii. Battery State of Charge:

- Since ATmega328P takes a max i/p of 5V, there arises the need for a voltage divider circuit. The difference between the two analog readings (A0 – A1) is multiplied by 2 times the resolution to get the battery voltage reading.
- A 9V is considered dead at 5.4V so these extreme values are mapped between 100 and 0.

iv. Liquid Crystal Display:

- The LCD is used to display the direction the wheelchair is going in, and the state of battery charge that is left.
- It is powered by Arduino, and information is communicated to it through the dsPIC via UART protocol.
- The LCD’s display contrast can be controlled by a potentiometer BK10.

v. *Motor Drivers:*

- This involves two DC motors driving the two rear wheels of the wheelchair as its rear wheel driven.
- The motors are connected to a motor driver from Allegro Microsystems called a3916GESTR-T.
- This consists of a dual DMOS full bridge setup which means there are 8 MOSFETS controlled by 4 inputs. From this we get two control inputs for each motor.
- The pins on the dsPIC33ck256MP506 that convey these 4 inputs are RC11, RC10, RC5 and RC4.
- Two control inputs per motor would imply 4 modes of operation, which are as follows:

MOTOR A:

INPUT 1 (RC11)	**INPUT 2 (RC10)**	**OPERATION**
0	0	Disable
1	0	Forward
0	1	Reverse
1	1	Brake

Table (3.1)

MOTOR B:

INPUT 3 (RC5)	**INPUT 4 (RC4)**	**OPERATION**
0	0	Disable
1	0	Forward
0	1	Reverse
1	1	Brake

Table (3.2)

- This 2-motor setup gives us 4 movements for the car: Forward, backward, clockwise rotation and anti-clockwise rotation which is better explained in the table below:

MOTOR A (left wheel)	**MOTOR B (right wheel)**	**CAR MOVEMENT**
Forward	Forward	Forward
Forward	Reverse	Clockwise (right)
Reverse	Forward	Anti-clockwise (left)
Reverse	Reverse	Reverse

Table (3.3)

- The motor driver setup is controlled by the joystick until an obstacle is detected by one of the US sensors in action.

vi. *Override setup using CLC (Combinational Circuit Logic):*

- dsPIC33ck256MP506 provides us with an option of CLC wherein we can use digital logic gates to perform certain operations.
- The output thus driven, can be used to trigger an interrupt, an event, or a digital output.
- In our case we use a 4 input AND gate which takes its inputs from the 4 US sensors.
- Since we need an OR operation, DeMorgan’s law is used, and logical inverters are placed at the 4 inputs which change our AND gate to a NOR gate. Hence another logical inverter is used at the output to give us a 4-input OR operation.

- Consider the following table:

US 1	US 2	US 3	US 4	OUTPUT
0	0	0	0	0
0	1	X	X	1
0	0	1	X	1
0	0	0	1	1
1	X	X	X	1

1: object detected   0: no object detected   X: don't care condition

Table (3.4)

- Peripheral Pin Select feature is used in here to enable inputs and output through the I/O pins of dsPIC33ck256MP506.
- The output is used to control a relay connecting the joystick to the dsPIC. When any one of the sensors detects an obstacle, a logic 1 is read at the CLC input and irrespective of other sensor readings we get a logic high at the output.
- This high output pulls open the **normally closed** relay thereby disconnecting the joystick from the processor. At this point the wheelchair comes to an immediate halt and stops taking inputs from the joystick until the obstacle is dodged.
- This hardware-based switching is better than the straightforward software-based switching as it reduces the size of the code and makes sure that the only delay associated with it is the propagation delay which is considerably small. This way we can save some time and memory.

4) <u>Software Section:</u>

a) Block Diagram:

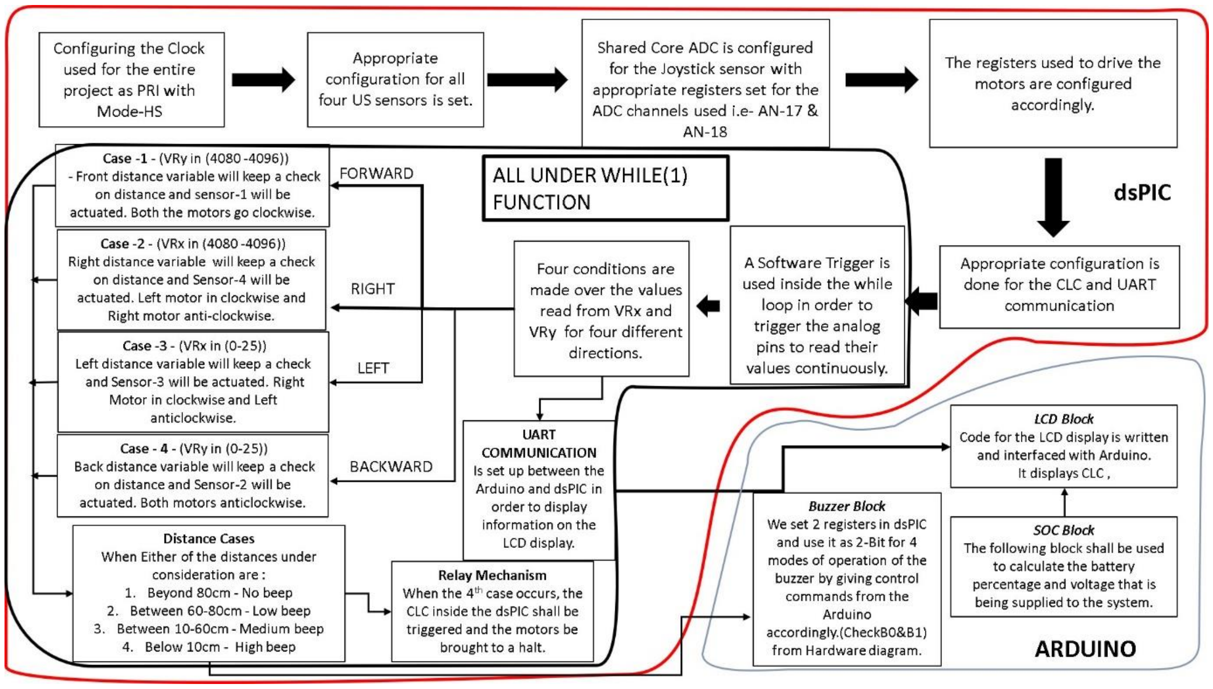
Configuring the Clock used for the entire project as PRI with Mode-HS
Appropriate configuration for all four US sensors is set.
Shared Core ADC is configured for the Joystick sensor with appropriate registers set for the ADC channels used i.e- AN-17 & AN-18
The registers used to drive the motors are configured accordingly.
dsPIC
Appropriate configuration is done for the CLC and UART communication
ALL UNDER WHILE(1) FUNCTION
A Software Trigger is used inside the while loop in order to trigger the analog pins to read their values continuously.
Four conditions are made over the values read from VRx and VRy for four different directions.
FORWARD
RIGHT
LEFT
BACKWARD
Case -1 - (VRy in (4080 -4096)) - Front distance variable will keep a check on distance and sensor-1 will be actuated. Both the motors go clockwise.
Case -2 - (VRx in (4080 -4096)) Right distance variable will keep a check on distance and Sensor-4 will be actuated. Left motor in clockwise and Right motor anti-clockwise.
Case -3 - (VRx in (0-25)) Left distance variable will keep a check and Sensor-3 will be actuated. Right Motor in clockwise and Left anticlockwise.
Case - 4 - (VRy in (0-25)) Back distance variable will keep a check on distance and Sensor-2 will be actuated. Both motors anticlockwise.
Distance Cases When Either of the distances under consideration are : 1. Beyond 80cm - No beep 2. Between 60-80cm - Low beep 3. Between 10-60cm - Medium beep 4. Below 10cm - High beep
Relay Mechanism When the 4th case occurs, the CLC inside the dsPIC shall be triggered and the motors be brought to a halt.
UART COMMUNICATION Is set up between the Arduino and dsPIC in order to display information on the LCD display.
Buzzer Block We set 2 registers in dsPIC and use it as 2-Bit for 4 modes of operation of the buzzer by giving control commands from the Arduino accordingly.(CheckB0&B1) from Hardware diagram.
LCD Block Code for the LCD display is written and interfaced with Arduino. It displays CLC ,
SOC Block The following block shall be used to calculate the battery percentage and voltage that is being supplied to the system.
ARDUINO

Fig. (4.a)

b) Functional Description of the Blocks:

- The following project entirely uses PRI Oscillator with HS mode. FCY is set at 16MHz and all the other clock configurations are set accordingly.

- We continue our code file by writing all the configuration bits for the peripherals/sensors that we use, inside the main function starting with the setup for US sensors.
- Timer-1 control register is set to be able to calculate the distance between obstacles. TECS=3 means that our clock for timer is set to FRC and timer's maximum value is set to 0xFFFF indicated by PR1. Four sensors are used and hence a set of variables initialized for calculations over distance and then followed by setting pins to trigger and echo for the four sensors.

- CLC is setup to override the joystick input when the obstacle is close. Four input sources are taken (One from each Ultrasonic sensor) and a 4 input OR Logic is set up such that whenever we get a logic high, we trigger to stop the motors. Four of these inputs are set using Peripheral Pin Select and CLC output is initialized to 0 initially. (LCEN=0) The mode used to set the CLC is 4-Input AND is used and since we need an OR operation, we invert our inputs by using GxPOL and also the Outputs by LCPOL=1. True inputs are taken from all the gates by setting True Enable bits and disabling negated bits.

- The ADC is then setup to read values from the Joystick Sensor. We take two analog pins (AN-17 and AN-18) to calculate the values of VRx and VRy, hence using a SHARED CORE for conversion of the analog values encountered by powering the Shared core (i.e.- SHRPWR) and enabling it (SHREN). A wait is set for the core to be ready using while loop and sampling time is set using SHRSAMC.

- UART is setup next by configuring its U1Mode control registers. The UARTEN and UTXEN are setup to start the transmission. UART is being used in Half duplex mode.

- A set of registers are set to enable Motor control via the Motor drivers placed on the dev board.

- The Infinite loop starts by triggering a Software Trigger, which is used to gather values from the VRx and VRy pins of the joystick and using the interrupts we read the Buffer registers of AN-17 and AN-18 respectively into Joy_x and Joy_y respectively and get four conditions based on the values of VRx and VRy as shown in the Software Diagram.

  A general working under each such condition is:

  i) Timer is started to collect values of only one Ultrasonic Sensor corresponding to the condition. Then taking the distance of only one sensor dedicated to that direction.
  ii) Buzzer conditions are enabled in each block to beep in their corresponding four conditions.
  NOTE: The Pins enabled shall be sent to Arduino directly for controlling the buzzer.

iii) A UART snippet is used to transmit information about the direction of motion of the Wheelchair and displayed onto the LCD display.
iv) Motor registers are given appropriate values to function.

- A critical condition is taken when either the brakes are applied or the CLC output goes high. In such a case, an Alert message is sent to LCD display and the Motors are stopped to function.

- Arduino is used to Setup the LCD display and Battery SOC calculation. For their detailed specifics and calculations, please refer to their individual schematics and descriptions.

5) Description of the System

The Power System block of this system essentially consists of two 9V batteries connected in parallel which later continues to power the circuit passing through a kill switch. The battery pack of two 500mAh batteries ensures enough current to drive two motors through a motor driver.

From here we can view the system as four different subsystems or blocks around the main processor:
1. The first block is for the battery-pack state determination.
2. The second block represents the tactile input (joystick) which takes physical inputs from the user and converts them into voltages later read by the processor.
3. The third block is the sensor block which represents four ultrasonic sensors put in place to detect the obstacles in the path of the wheelchair.
4. The fourth block is the output block which represents two kinds of outputs:
   a) Output driven by the Dual DMOS Full Bridge motor driver
   b) Output driven by Arduino which is interfaced with the main processor, dsPIC33ck256MP506 using UART protocol.

The battery-pack state block is made possible with the help of ATmega328P used in Arduino. Since it cannot take more than 5V, we have a voltage divider in place. The voltage is read from this, processed and mapped between 0% and 100%.
NOTE: It’s advised to have a battery-replacement ready at 33% SoC as the battery will be dead at 5.4V.
From the tactile input block, the joystick movements are detected. The wheelchair is made to move accordingly. The chair movements are: straight front, straight back, spin right (clockwise), spin left (counter-clockwise). Hence the input taken is put through a condition check. This condition check is what connects the input block to the output block. The motor driver drives the two wheels in the rear based on this check.
If the command received is to move straight front or back, both the motors rotate in the same direction (either forward or backward respectively).
But for the spin commands the motors rotate in different directions.

The Sensor block is also put through a series of condition checks. The 4 US sensors are placed in the North, South, East and West directions of the wheelchair. The

sensors look for objects in the path of the wheelchair. For instance, if the joystick commands the chair to go left, the left sensor checks for obstacles before initiating the turn sequence. This sums up the 1st condition check.
The 2nd condition is a 3-part condition nested in the 1st one.
If the obstacle is at a distance:

i) *equal to 60cm:* If true the buzzer starts beeping at a low frequency.
ii) *in between 60cm and 15cm:* If true the frequency of beeping increases as the distance keeps shortening.
iii) *less than 15cm:* If true the tactile controls are overridden, and an emergency braking sequence is initiated. This is made possible by the Combinational Logic Circuit (CLC) used in dsPIC33ck256MP506. 4 US sensor-alerts correspond to 4 gate inputs at CLC.

At smooth cruising conditions, the CLC inputs read zero but when any of the sensors encounter a close proximity condition, that corresponding gate input reads logic high giving a logic high at its output as well. This is used as a flag to initiate the emergency braking sequence.
In case of a sensor malfunction, if the tactile inputs are overridden, the user can quickly toggle the kill switch and shut the system down. This applies to any emergency shut down situations.
The display which is one of the components of our output block, displays:

1. the state of the battery-pack (SoC and pack voltage),
2. the direction in which the chair is moving,
3. alert message when any obstacle in encountered.

6) System Analysis

i) *Joystick:*

Fosc = 32MHz

TCoresrc = (1/32MHz) = 31.25ns

TADCore = TCoresrc /2= 15.625ns

Conversion Time = [8*(TCoresrc) + (12+2.5) * (TADCore)] = 476.56ns

Sampling Rate = 32* TADCore = 500ns

ii) *Ultrasonic Sensors:*

Frequency of FRC= 8MHz.

Time (in seconds) = TMR1/8MHz

Distance (in cm) = (Speed * Time) / 2

[Speed = 34000 cm/s]

iii) Battery SoC:

Dividing 9V voltage: [9*1000/ (1000+1000)] = 4.5V

10-bit ADC's resolution = 5/1024 = 4.88mV

Voltage Reading = (A0-A1) * 2 * resolution

5.4V -> 0%; 9.0V -> 100%

7) Detailed Block Design

a) Joystick:

- *Schematic:*

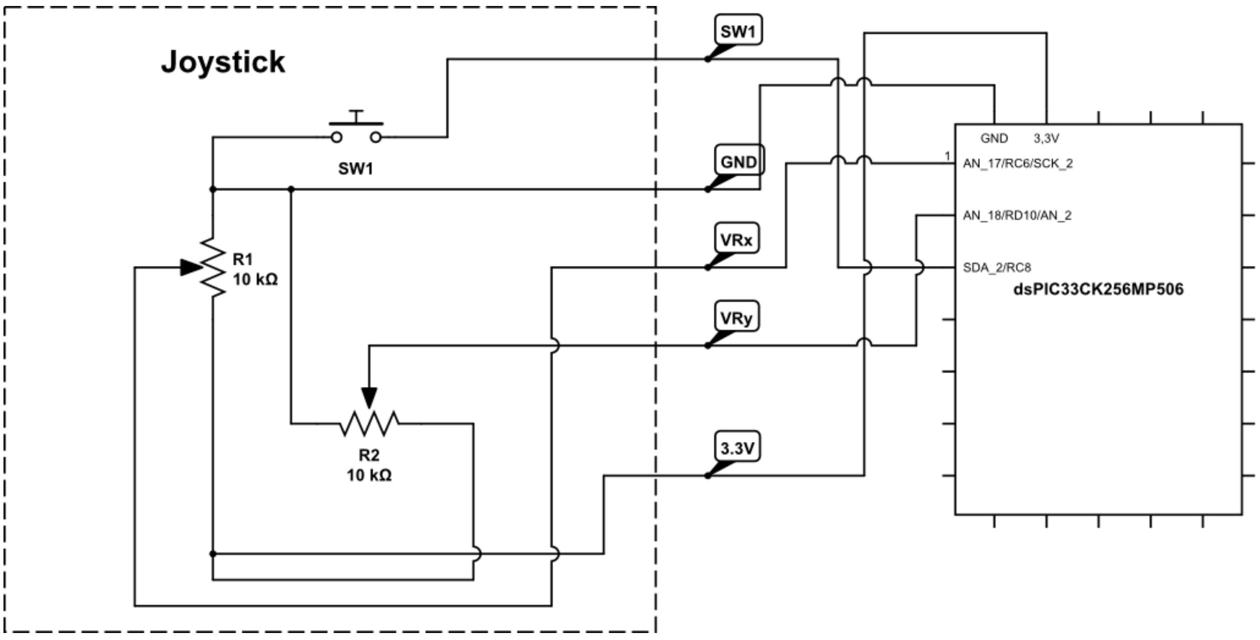

Fig. (7.a)

- *Test Plan:*
  Glow an LED bulb for each direction respectively.
- *Test Log:*
  Test Successful.

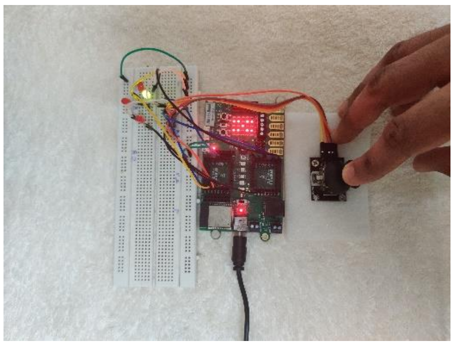

Fig. (7.b) Back

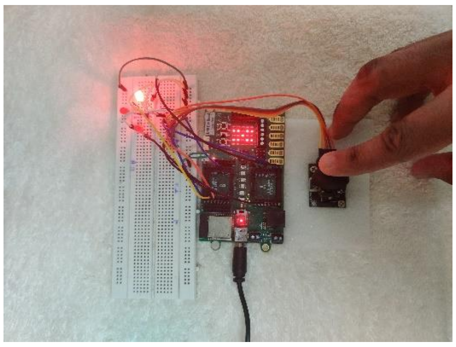

Fig. (7.c) Right

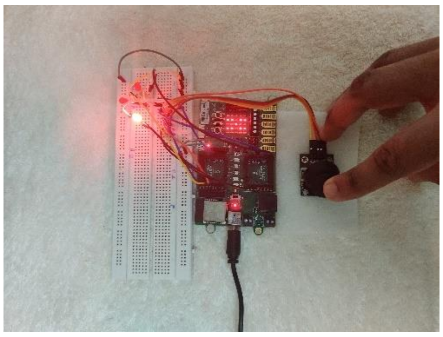
Fig. (7.c) Left

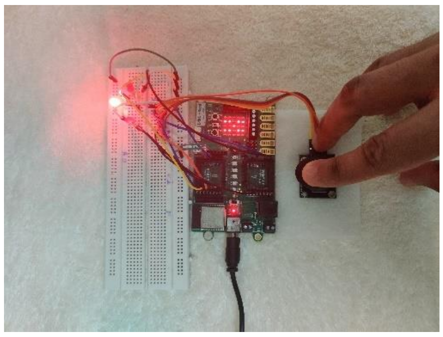
Fig. (7.d) Front

b) Ultrasonic Sensors:

- *Schematic:*

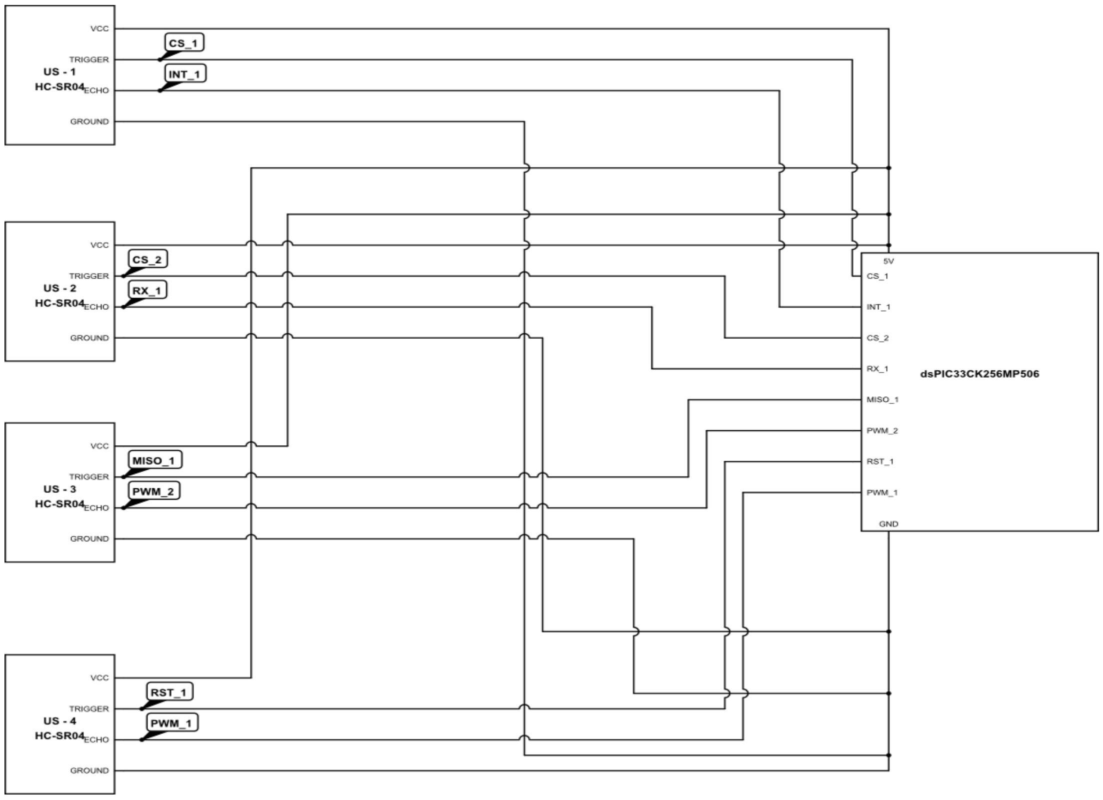

Fig. (7.e)

- *Test Plan:*
  Range testing - Using wall as the reference, move the sensor towards and away from it. Calculate the output values and cross check with the actual distance.
- *Test Log:*
  Upon noting the values after changing the distance condition several times, it was observed that the error increased with distance.
  The maximum distance that the sensor could detect was 128-129cms.
  Whenever the distance of the obstacle was beyond the capacity of the sensor, say the distance between the censor and the ceiling, it always returned dummy values in the range of 80-90cms.

c) Motor Driver:

- *Schematic:*

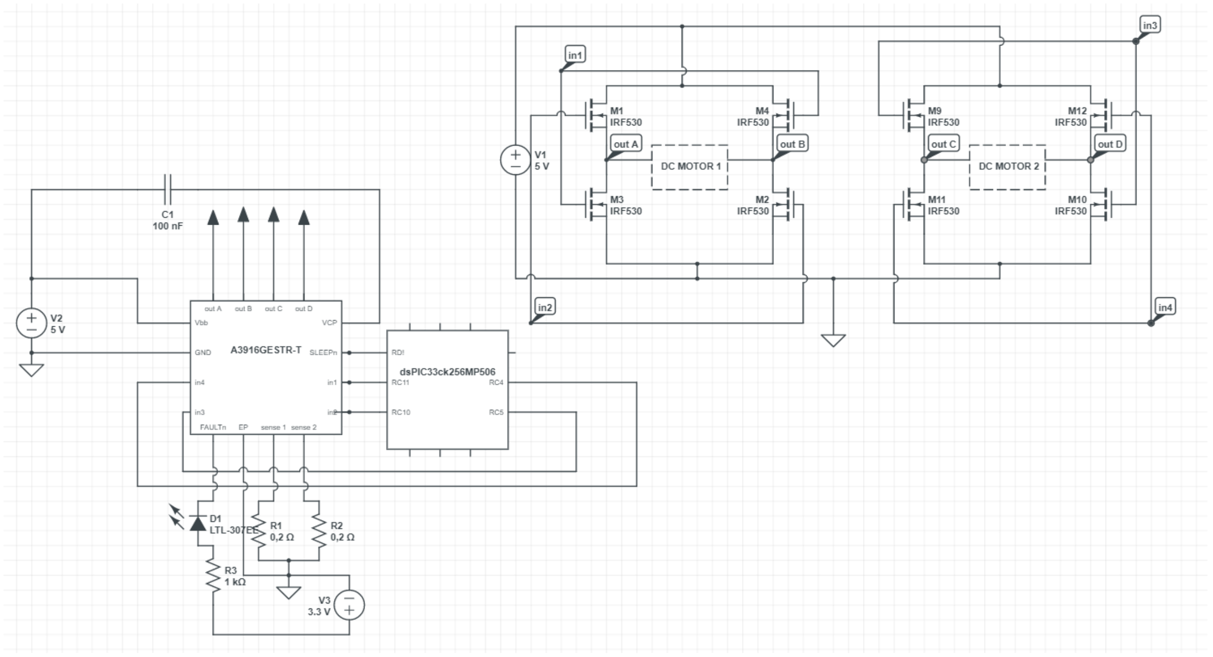

Fig. (7.f)

- *Test Plan:*
  Multiple iterations of different motor controls for 10 minutes to check its behavior at elevated motor temperatures.
- *Test Log:*
  Sustained normal behavior.

d) Liquid Crystal Display:

- *Schematic:*

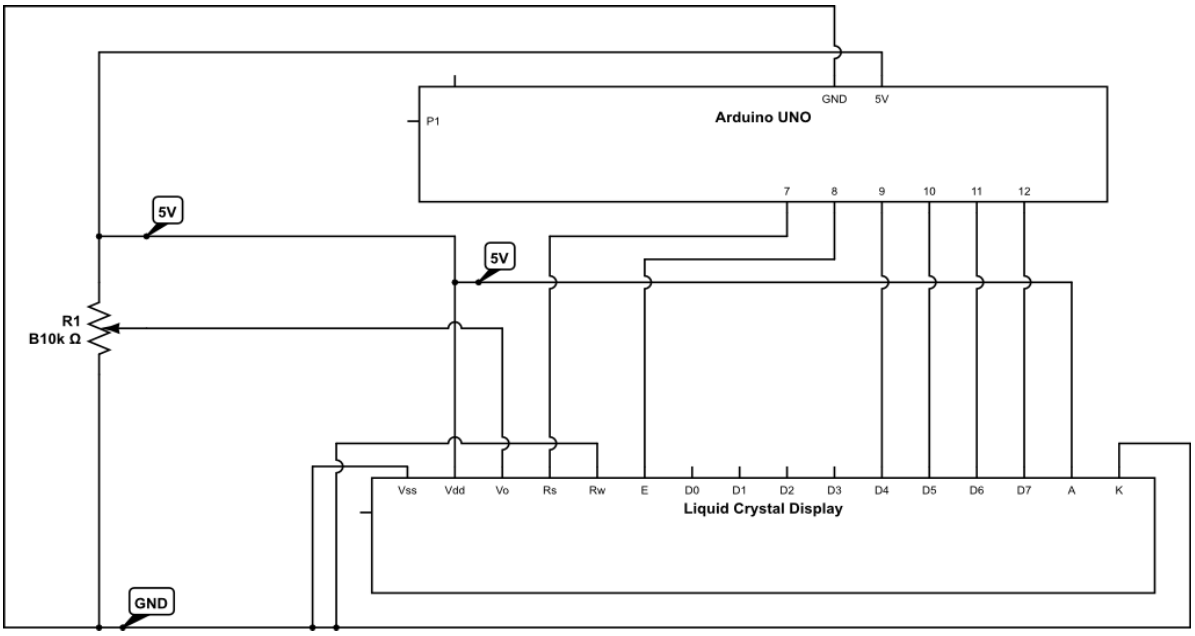

Fig. (7.g)

- *Test Plan:*
  Receive the string "TOM" from the dsPIC on the arduino and print it on the LCD screen.
- *Test Log:*
  String "TOM" and printed on the LCD screen successfully.

e) Battery SoC:

- *Schematic:*

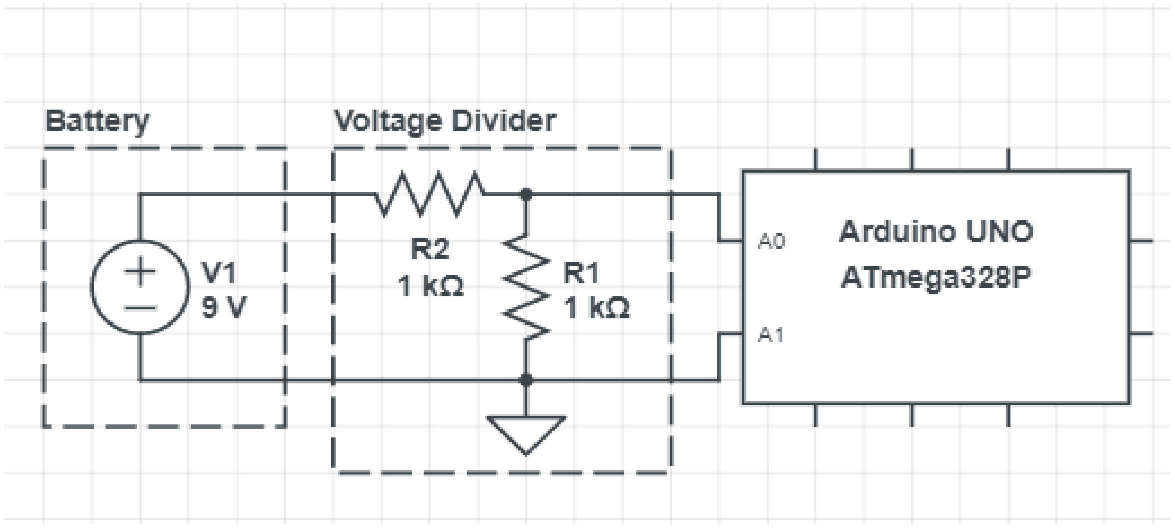

Fig. (7.h)

- *Test Plan:*
  Running a discharging setup with a potentiometer and batteries connected in parallel.
- *Test Log:*
  Shows full swing of voltage with 4 levels of SoC- 100, 66, 33 0.
  Simulation Link: https://www.tinkercad.com/things/aILuYYxZKOT-stunning-gogo/editel?sharecode=uJoRs3k1k5m7dZMAONlA_5mkZjfjt6B8yL4eYDCDgKU

f) Combinational Circuit Logic:

- *Schematic:*

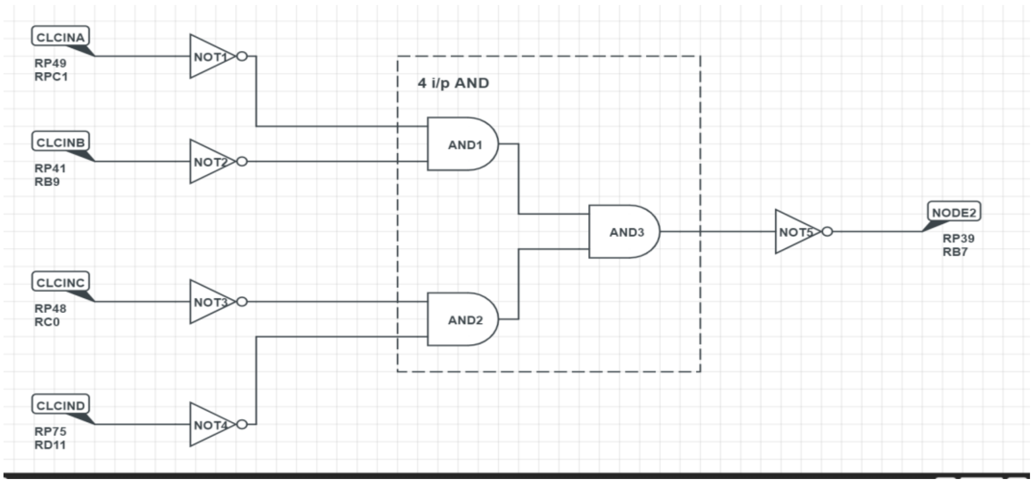

Fig. (7.i)

- *Test Plan:*
  Placing an obstacle less than 10cm away from the chair.
- *Test Log:*
  Vehicle stops.

8) <u>System Testing:</u>

- *Test Plans:*

  a) Speed Test

  b) Testing the car in cramped up spaces to check behavior

  c) Brake Test

  d) Emergency Brake Test

  e) Stopping distance measurement post brake application

  f) Incline and Decline Maneuverability

  g) Turning Maneuverability on Turns with different Turn Angles and Turn Radii

  h) Ground Clearance Test

9) <u>Conclusion:</u>

Right from the get-go, our team met with a whole lot of challenges on a consistent basis- be it the lab kits getting stuck in transit, or be it the COVID-19 outbreak, multiple burnouts, or even the dev boards frying up. However, throughout the term there was always at least one motivated individual who'd inspire the rest to keep pushing through. Making the TOM Chair was certainly not half as easy as we thought it'd be. But it is most definitely going to be the one project we'll always look back to and pat ourselves on the back for giving it an honest shot.

ECE3232 has been a rollercoaster, honestly. We strengthened our teamworking skills although for the most part we were physically away from each other, learnt the importance of modularity, about the challenges embedded designers face on a regular basis, about efficient project management, the importance of communication, the necessity of technical vocabulary, how to read a datasheet(finally!), and most importantly, the impact asking questions can have. We've asked more questions in one course than we have in any semester till date! We're pretty excited about implementing all that we could learn regarding communication protocols, interrupts, and so on.

There were a lot of areas where we struggled. Our biggest fear was that of frying our dev boards up with no technicians equipped enough to have it fixed for us; and also with our academic area at college being restricted. The snap frequently acted up, and the same code when built into the target device, would sometimes give different outputs. Another area where we struggled the most is with testing. We had extremely little time (because of the curfews imposed owing to COVID-19) and resources to get our project going full swing, but that’s where we emphasized on and learnt about optimization. We struggled a little in understanding how UART works between the dsPIC and Arduino, because though the test plan worked, we weren’t able to print required data on the LCD so far.

If we had to modify our project, we’d use 8 sensors instead of the 4 sensors we settled for, we’d use faster motors with PWM, add voice recognition, make it touch operated, use rechargeable batteries, and provide shock absorption to it.

10) References:

- Joystick Internal Diagram - Electronicshub.com

- dsPIC33 FRM for ADC

- dsPIC33 FRM for TIMER1

- dsPIC33ck256MP508 Datasheet

- Electrosome for dsPIC to interface Ultrasonic Sensor

- arduino.cc/reference/en/language/functions/communication/serial

- usermanual.wiki/Jakks-Pacific/RC24063T

- Arduino Tutorial 48: Connecting and Using an LCD Display by Paul McWhorter

- CLC Reference Manual

- a3916 Datasheet

11) Appendix

MPLab:

```
#pragma config FNOSC = PRI          // Oscillator Source Selection
(Internal Fast RC (FRC) Oscillator with postscaler)
#pragma config IESO = ON                 // Two-speed Oscillator Start-
up Enable bit (Start up device with FRC, then switch to user-selected
oscillator source)

#pragma config ICS=2
// FOSC
#pragma config POSCMD = HS               // Primary Oscillator Mode
Select bits (HS Crystal Oscillator Mode)
#pragma config OSCIOFNC = ON             // OSC2 Pin Function bit (OSC2
is general purpose digital I/O pin)
#pragma config FCKSM = CSDCMD            // Clock Switching Mode bits
(Both Clock switching and Fail-safe Clock Monitor are disabled)
#pragma config PLLKEN = ON               // PLL Lock Status Control (PLL
lock signal will be used to disable PLL clock output if lock is lost)
#pragma config XTCFG = G3                // XT Config (24-32 MHz
crystals)
#pragma config XTBST = ENABLE            // XT Boost (Boost the kick-
start)
#define FCY 16000000UL

#include <libpic30.h>
#include <p33CK256MP506.h>
#include "xc.h"
volatile unsigned short dataAN17;
volatile unsigned short dataAN18;

void init_uart(){
    U1MODEbits.UARTEN = 1;
    U1MODEbits.UTXEN = 1;
    U1MODEbits.USIDL = 1;
    U1MODEbits.BRGH = 0;
    U1BRG = 0x67;
    U1STAHbits.UTXISEL = 7;
    U1MODEHbits.BCLKSEL = 0;
    U1MODEHbits.HALFDPLX = 1;
}

void init_timer(){
    TMR1 = 0;

}

int main(void) {

    //US Sensor Setting Up
    T1CON = 0x0;
    T1CONbits.TECS = 3;
    T1CONbits.TCS = 1;
```

```
PR1 = 0xFFFF;
double count = 0;
double time = 0;
double frontDistance = 0;
double count2 = 0;
double time2 = 0;
double backDistance = 0;
double rightDistance =0 ;
double count3 =0;
double time3 =0;
double leftDistance = 0;
double count4 = 0;
double time4 = 0;
int keyValue0 = 0;
int keyValue1 = 0;
//taking key data
//TRISDbits.TRISD3 = 1;
//TRISDbits.TRISD4 = 1;

//CLC
ANSELBbits.ANSELB7=0;//CLC output pin
ANSELDbits.ANSELD11=0;//CLCINDR input source4
ANSELCbits.ANSELC0=0;//CLCINCR input source 3
ANSELBbits.ANSELB9=0;//CLCINBR input source 2
ANSELCbits.ANSELC1=0;//CLCINAR input source 1
TRISBbits.TRISB7=0;
TRISDbits.TRISD11=1;
RPINR47bits.CLCINDR=75;
TRISCbits.TRISC0=1;
RPINR46bits.CLCINCR=48;
TRISBbits.TRISB9=1;
RPINR46bits.CLCINBR=41;
TRISCbits.TRISC1=1;
RPINR45bits.CLCINAR=49;
CLC1CONLbits.LCEN=0;
CLC1CONLbits.INTP=0;
CLC1CONLbits.INTN=0;
CLC1CONLbits.LCOE=1;
CLC1CONLbits.LCOUT=1;
CLC1CONLbits.LCPOL=1;
CLC1CONLbits.MODE=2;
CLC1CONHbits.G4POL=1;
CLC1CONHbits.G3POL=1;
CLC1CONHbits.G2POL=1;
CLC1CONHbits.G1POL=1;
CLC1SELbits.DS4=5;
CLC1SELbits.DS3=0;
CLC1SELbits.DS2=0;
CLC1SELbits.DS1=0;
CLC1GLSLbits.G2D4T=1;
CLC1GLSLbits.G2D4N=0;
CLC1GLSLbits.G2D3T=1;
CLC1GLSLbits.G2D3N=0;
CLC1GLSLbits.G2D2T=1;
CLC1GLSLbits.G2D2N=0;
CLC1GLSLbits.G2D1T=1;
CLC1GLSLbits.G2D1N=0;
CLC1GLSLbits.G1D4T=1;
```

```
CLC1GLSLbits.G1D4N=0;
CLC1GLSLbits.G1D3T=1;
CLC1GLSLbits.G1D3N=0;
CLC1GLSLbits.G1D2T=1;
CLC1GLSLbits.G1D2N=0;
CLC1GLSLbits.G1D1T=1;
CLC1GLSLbits.G1D1N=0;
CLC1GLSHbits.G4D4T=1;
CLC1GLSHbits.G4D4N=0;
CLC1GLSHbits.G4D3T=1;
CLC1GLSHbits.G4D3N=0;
CLC1GLSHbits.G4D2T=1;
CLC1GLSHbits.G4D2N=0;
CLC1GLSHbits.G4D1T=1;
CLC1GLSHbits.G4D1N=0;
CLC1GLSHbits.G3D4T=1;
CLC1GLSHbits.G3D4N=0;
CLC1GLSHbits.G3D3T=1;
CLC1GLSHbits.G3D3N=0;
CLC1GLSHbits.G3D2T=1;
CLC1GLSHbits.G3D2N=0;
CLC1GLSHbits.G3D1T=1;
CLC1GLSHbits.G3D1N=0;

//SENSOR 3//left
TRISBbits.TRISB8 = 0;// MISO1
ANSELBbits.ANSELB8 = 0;
TRISCbits.TRISC12 = 1;//PWM2
//SENSOR1//forward
ANSELBbits.ANSELB2 = 0;
TRISBbits.TRISB2 = 0;//CS1
TRISCbits.TRISC14 = 1;//INT1
//SENSOR2//back
TRISCbits.TRISC3 = 0;//CS2
ANSELCbits.ANSELC3 = 0;
TRISCbits.TRISC15 = 1;//RX1
//SENSOR4//right
TRISCbits.TRISC7 = 0;//RST1
ANSELCbits.ANSELC7 = 0;
TRISCbits.TRISC13 = 1;//PWM1
TRISBbits.TRISB15 =  0;


//Motor Control Setup
TRISDbits.TRISD1=0;
TRISCbits.TRISC11=0;//MOT1
TRISCbits.TRISC10=0;//MOT1
TRISCbits.TRISC5=0;//MOT2
TRISCbits.TRISC4=0;//MOT2
LATCbits.LATC4=1;
LATCbits.LATC5=1;
LATCbits.LATC10=1;
LATCbits.LATC11=1;
LATDbits.LATD1=1;

//ADC Configuration
ANSELCbits.ANSELC1=0;
//ANSELBbits.ANSELB2=0;//
```

```
    ANSELCbits.ANSELC6=1;
    ANSELDbits.ANSELD10=1;
    TRISCbits.TRISC6=1; //Vrx(AN_17/SCK-2)
    TRISDbits.TRISD10=1;//Vry(AN_18/AN2)

    TRISDbits.TRISD4=0; //buzzer-0
    TRISCbits.TRISC9=0; //buzzer-1
    TRISCbits.TRISC8=1;//sw                        //Switch in Joystick

    ADCON3Hbits.CLKSEL= 1;
    ADCON3Hbits.CLKDIV = 0;
    ADCON1Hbits.FORM=0;
    ADCORE0Hbits.ADCS= 0;//4 Source Clock periods
    ADCORE1Hbits.ADCS= 0;
    ADCON1Hbits.SHRRES = 3;
    ADCON3Lbits.REFSEL=0;
    ADMOD1Lbits.DIFF17 = 0; // AN17/RC6
    ADMOD1Lbits.DIFF18 = 0; // AN18/RD10
    ADMOD1Lbits.SIGN17 = 0; // AN17/RC6
    ADMOD1Lbits.SIGN18 = 0; // AN18/RD10


    ADCON5Hbits.WARMTIME= 5;
   // Turn on analog power for dedicated core 0

    ADCON5Lbits.SHRPWR = 1;
// Wait when the shared core is ready for operation
// Turn on digital power to enable triggers to the shared core
    PORTCbits.RC1=1;
    ADCON3Hbits.SHREN = 1;
    while(ADCON5Lbits.SHRRDY == 0);
    ADCON2Lbits.SHRADCS=0;
    ADCON2Hbits.SHRSAMC= 00001000000;

    ADIEHbits.IE17=1;
    ADIEHbits.IE18=1;



    _ADCAN17IF = 0;
    _ADCAN17IE = 1;
    _ADCAN18IF = 0;
    _ADCAN18IE = 1;

    ADCON1Lbits.ADON=1;
    double joy_x=0;
    double joy_y=0;
    double sw=0;





    //UART SETUP
     int i;
    char msg[7] = "FORWARD";
    char msg1[7] = "REVERSE";
    char msg2[5] = "RIGHT";
    char msg3[4] = "LEFT";
    char msg4[5] = "BRAKE";
    RPOR6bits.RP45R =1;
    INTCON2bits.GIE = 1;
    IEC0bits.U1TXIE = 1;

    int w=0;
    int x=0;
    int y=0;
    int z=0;
    while(1)
    {
        sw = PORTCbits.RC8;

        //CLC
        CLC1CONLbits.LCEN=1;
         w=PORTDbits.RD11;
         x=PORTCbits.RC0;
         y=PORTBbits.RB9;
         z=PORTCbits.RC1;
        RPOR3bits.RP39R=40;

        ADCON3Lbits.SWCTRG=1;

        _TRGSRC17 = 1;
        joy_x= ADCBUF17;

        Nop();
        Nop();
        Nop();
        _TRGSRC18 = 1;
        joy_y= ADCBUF18;

        Nop();
        Nop();
        Nop();
        Nop();

        if((joy_y >= 4080)&&(joy_y <= 4096))
        {   //SENSOR1//FORWARD
            init_uart();
            for(i = 0; i<16640; i++){
                 Nop();
            }
            for(i = 0;i<7;i++){
                if(U1STAHbits.UTXBF == 0){
                U1TXREG = msg[i];//loading
                }
            }
             while(1);
            init_timer();
            LATBbits.LATB2 = 1;
            __delay_us(10);
            LATBbits.LATB2 = 0;
            while(!PORTCbits.RC14);
            T1CONbits.TON = 1;
            while(PORTCbits.RC14);
            T1CONbits.TON = 0;
            count = TMR1;
            time = count/8000000;
            frontDistance = (time * 34000) / 2;
            //keyValue0 = PORTDbits.RD3;
            //keyValue1 = PORTDbits.RD4;

            //Buzzer
            if((60<=frontDistance)&&(frontDistance<=80)){
              LATDbits.LATD4= 0;
              LATCbits.LATC9=1;
            }
            else if((frontDistance>=10)&&(frontDistance<=60)){
                LATDbits.LATD4=1;
                LATCbits.LATC9=0;
            }
             else if(frontDistance <10 ){
                LATDbits.LATD4=1;
                LATCbits.LATC9=1;
            } else
            {
               LATDbits.LATD4=0;
               LATCbits.LATC9=0;
            }

            init_timer();

            //FORWARD
             LATCbits.LATC11=1;
             LATCbits.LATC10=0;
             LATCbits.LATC5=1;
             LATCbits.LATC4=0;
             __delay_ms (3000);
             Nop();
             Nop();
        }

        else if((joy_x >= 0)&&(joy_x <= 25))
        {
            //Sensor3 //LEFT
             init_uart();
            for(i = 0; i<16640; i++){
                 Nop();
            }
            for(i = 0;i<4;i++){
                if(U1STAHbits.UTXBF == 0){
                U1TXREG = msg3[i];//loading
                }
            }
              while(1);
            init_timer();
            LATBbits.LATB8 = 1;
            __delay_us(10);
            LATBbits.LATB8 = 0;
            while(!PORTCbits.RC12);
            T1CONbits.TON = 1;
            while(PORTCbits.RC12);
            T1CONbits.TON = 0;
            count3 = TMR1;
            time3 = count3/8000000;
            leftDistance = (time3 * 34000) / 2;

            //keyValue0 = PORTDbits.RD3;
            //keyValue1 = PORTDbits.RD4;
            if((60<=leftDistance)&&(leftDistance<=80)){
              LATDbits.LATD4= 0;
              LATCbits.LATC9=1;
            }
            else if((10<=leftDistance)&&(leftDistance<=60)){
                LATDbits.LATD4=1;
                LATCbits.LATC9=0;
            }
            else if(leftDistance <10 ){
                LATDbits.LATD4=1;
                LATCbits.LATC9=1;
            } else
            {
                 LATDbits.LATD4=0;
                 LATCbits.LATC9=0;
            }

            init_timer();

                 //LEFT TURN (CCW)
            LATCbits.LATC11=1;
            LATCbits.LATC10=0;
            LATCbits.LATC5=0;
            LATCbits.LATC4=1;
            __delay_ms (3000);
            Nop();
            Nop();
        }

        else if((joy_x >= 4080)&&(joy_x <= 4096))
        {
            //SENSOR4
            init_uart();
            for(i = 0; i<16640; i++){
                 Nop();
            }
            for(i = 0;i<7;i++){
                if(U1STAHbits.UTXBF == 0){
                    U1TXREG = msg2[i];//loading
                }
            }
            while(1);

            init_timer();
            LATCbits.LATC7 = 1;
            __delay_us(10);
            LATCbits.LATC7 = 0;
            while(!PORTCbits.RC13);
            T1CONbits.TON = 1;
            while(PORTCbits.RC13);
            T1CONbits.TON = 0;
            count4 = TMR1;
            time4 = count4/8000000;
            rightDistance = (time4 * 34000) / 2;
            Nop();
            Nop();
            Nop();

            if((60<=rightDistance)&&(rightDistance<=80)){
              LATDbits.LATD4= 0;
              LATCbits.LATC9=1;
            }
            else if((10<=rightDistance)&&(rightDistance<=60)){
                LATDbits.LATD4=1;
                LATCbits.LATC9=0;
            }
             else if(rightDistance <10 ){
                LATDbits.LATD4=1;
                LATCbits.LATC9=1;
            } else
            {
                 LATDbits.LATD4=0;
                 LATCbits.LATC9=0;
            }


            init_timer();


                //RIGHT TURN (CW)
            LATCbits.LATC11=0;
            LATCbits.LATC10=1;
            LATCbits.LATC5=1;
            LATCbits.LATC4=0;
             __delay_ms (3000);

            Nop();
            Nop();
        }
        else if((joy_y >= 0)&&(joy_y <= 25))
        {
            //SENSOR2
            init_uart();
            for(i = 0; i<16640; i++){
                 Nop();
            }
            for(i = 0;i<7;i++){
                if(U1STAHbits.UTXBF == 0){
                    U1TXREG = msg1[i];
                }
            }
            while(1);
            LATCbits.LATC3 = 1;
            __delay_us(10);
            LATCbits.LATC3 = 0;
            while(!PORTCbits.RC15);
            T1CONbits.TON = 1;
            while(PORTCbits.RC15);
            T1CONbits.TON = 0;
            count2 = TMR1;
```

```
            time2 = count2/8000000;
            backDistance = (time2 * 34000) / 2;

            init_timer();


             //REVERSE
            LATCbits.LATC11=0;
            LATCbits.LATC10=1;
            LATCbits.LATC5=0;
            LATCbits.LATC4=1;
            __delay_ms (3000);
            if((60<=backDistance)&&(backDistance<=80)){
               LATDbits.LATD4= 0;
               LATCbits.LATC0=1;
            }
            else if((10<=backDistance)&&(backDistance<=60)){
                LATDbits.LATD4=1;
                LATCbits.LATC9=0;
            }
             else if(backDistance <10 ){
                LATDbits.LATD4=1;
                LATCbits.LATC9=1;
            } else
            {
                 LATDbits.LATD4=0;
                 LATCbits.LATC9=0;
            }
                Nop();
                Nop();
                }

            else if((sw==1)||(PORTBbits.RB7=1))// Add CLC here.
            {
                    //BRAKE
                init_uart();
                for(i = 0; i<16640; i++){
                     Nop();
                }
                for(i = 0;i<5;i++){
                    if(U1STAHbits.UTXBF == 0){
                        U1TXREG = msg4[i];//loading
                    }
                }
                 while(1);
                LATCbits.LATC11=1;
                LATCbits.LATC10=1;
                LATCbits.LATC5=1;
                LATCbits.LATC4=1;
                __delay_ms (3000);
        }
        //reset conditions

    }

    return 0;
}
```

```
  // ADC AN12 ISR
void __attribute__((interrupt, no_auto_psv)) _ADCAN17Interrupt(void)
{
dataAN17 = ADCBUF17; // read conversion result
_ADCAN17IF = 0; // clear interrupt flag
}
  // ADC AN18 ISR
void __attribute__((interrupt, no_auto_psv)) _ADCAN18Interrupt(void)
{
dataAN18 = ADCBUF18; // read conversion result
_ADCAN18IF = 0; // clear interrupt flag
}
void __attribute__((__interrupt__, no_auto_psv )) _U1TXInterrupt(void)
{
    IFS0bits.U1TXIF = 0; // clear TX interrupt flag

}
```

Arduino

```
#include <LiquidCrystal.h>
int rs=7;
int en=8;
int d4=9;
int d5=10;
int d6=11;
int d7=12;
LiquidCrystal lcd(rs,en,d4,d5,d6,d7);
String received_data;

void setup() {
  // put your setup code here, to run once:
lcd.begin(16,2);
Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
lcd.setCursor(0,0);
lcd.print("on");
lcd.setCursor(0,1);
if(Serial.available())
  {
    lcd.print("data:");
    received_data = Serial.readString();
    lcd.print(received_data);
    delay(1000);
    lcd.clear();
  }
}
```