ELECTRONIC DESIGN PROJECT REPORT

“SELF PARKING CAR”

Aayush Rathi (06D07004)

Ashay Awate (06D07012)

Narendra Shiradkar (06D07016)

Milind Kothekar (06D07017)

INDEX

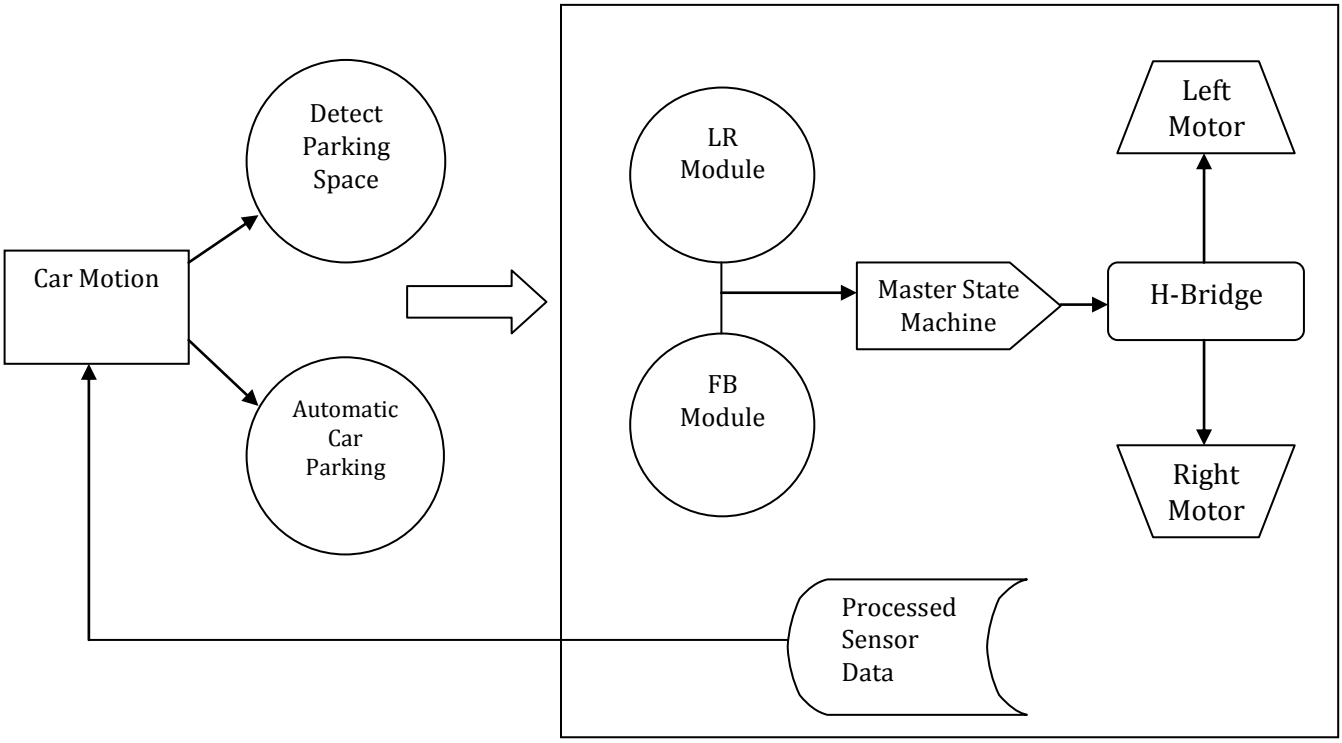
INTRODUCTION

Given a constrained space, many car drivers today face a problem in parking their cars. Parking a car in a small space requires a series of forward and reverse motions and turns which prove to be a complicated task for most car drivers to handle. We have tried to address this problem in our current EDL project by making an automated parking car which parks itself at the touch of a button.

Our project will be a miniature model of a real life car which will sense the availability of parking space and park itself automatically in the available parking area. The surrounding terrain details are continuously sensed by using a rotating distance measurement sensor which will sense the distances of the car from adjacent walls and the angle of car with respect to the surroundings. Using this continuous feedback we plan to accomplish both parallel as well as perpendicular parking schemes in our design.

HIGH LEVEL DESIGN

Logic structure



Control System

The control system contains all the hardware and its associated software. It allows Self Parking and Parking Detection algorithms to interface with the car. The Software can be broken down into three major sections: Distance and Angle Calculations, Left-Right and Front-Back software modules and the Master State Machine. The LR/FB software module determines which direction to move the car based on flags set by Parking space Detection and Automatic Car Parking algorithms. Once this module decides on the direction of motion of the car, the Master State Machine implements this movement by sending relevant input signals to the H-Bridge (L298). The Distance calculations are done independently in real time. To calculate the angle, the distance readings are transmitted in real time to the microcontroller which then computes the angle based on motor rotation.

Hardware Tradeoffs

Distance Measurement Sensors

- While selecting the sensors there was always a tradeoff between sensor's ability to measure close range versus long range. We selected relatively long range sensors as our car's dimensions were large enough for us not to need the very short range distance measurement

- There was also a tradeoff between using more number of sensors versus using a rotating sensor. We chose rotating sensor as a cost effective solution as each sensor costs around Rs. _____. The single rotating sensor can be used to measure distances from all the surrounding walls as well as getting the global angle of the vehicle with respect to the surroundings.

- Using rotating sensor costs us significant amount of time. Also using these accurate sensors costs us a delay of 40 ms for each reading. This introduces a hint of inaccuracy in the final position of the car which can of course be removed by taking readings when the car is completely stationary and adjusting the car's final position.

Motor Drivers

- The selection of motor drivers was the trickiest part of all. There were a lot of factors to be considered while keeping an eye on the cost of components at the same time. These tradeoffs are explained in great detail in the Hardware Description section of the report.

Batteries

- There was a tradeoff between battery weight, power and cost. We chose Li-Ion batteries as they seem to perfectly match our requirement. These batteries give a high open circuit voltage as compared to their aqueous counterparts (Lead Acid, Ni-Cd, Ni-MH). These batteries are also lighter than the lead acid batteries and cheaper compared to Ni-Cd and Ni-MH batteries. We thus used a set of 4 Li-Ion batteries of 4.2 Volts each and _____mAh giving us a total 16.8 Volts power supply onboard the vehicle.

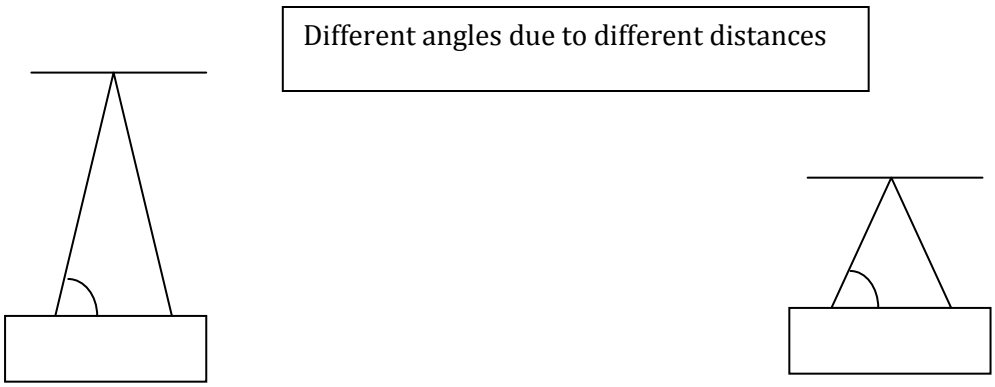
- As the batteries wore out, they supplied less and less current to the motors. This made calibrating the velocity of the car very difficult. This wasn't a problem for stepper motors as we used a constant current drive for that purpose. But DC motors get heavily affected by battery drainout.

HARDWARE DESCRIPTION

The hardware of the car is divided into 4 major parts.

1. DC Motor Driver Board
2. Stepper Motor Driver Board
3. Rotating Sensor Board
4. Main Control Board

We are using SHARP GP2D12 sensor for distance measurement. The GP2D12 is a distance measuring sensor with integrated signal processing and analog voltage output. The sensor uses triangulation and a small linear CCD array to compute the distance and/or presence of objects in the field of view. The basic idea is that a pulse of IR light is emitted by the emitter. This light travels out in the field of view and either hits an object or just keeps on going. In the case of no object, the light is never reflected and the reading shows no object. If the light reflects off an object, it returns to the detector and creates a triangle between the point of reflection, the emitter, and the detector.

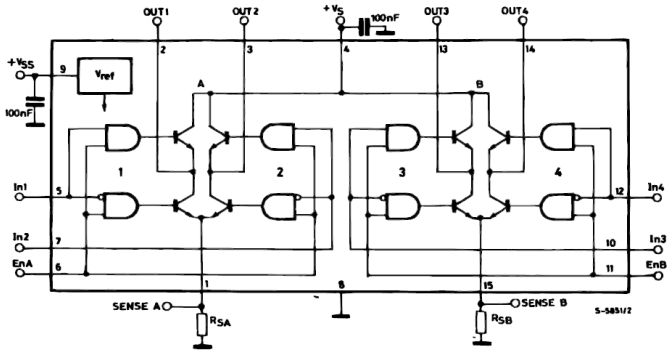


Using this triangle the signal processing circuit inside the sensor calculates the analog voltage corresponding to the distance between the object and the sensor and we get the relevant voltage at the output.

Now, let us have a look at the hardware design

DC MOTOR DRIVER BOARD

To drive DC motors we used a single L298N (Dual Full Bridge Driver). The L298 is an integrated monolithic circuit in a 15-lead Multiwatt package. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads. The internal circuitry is as follows

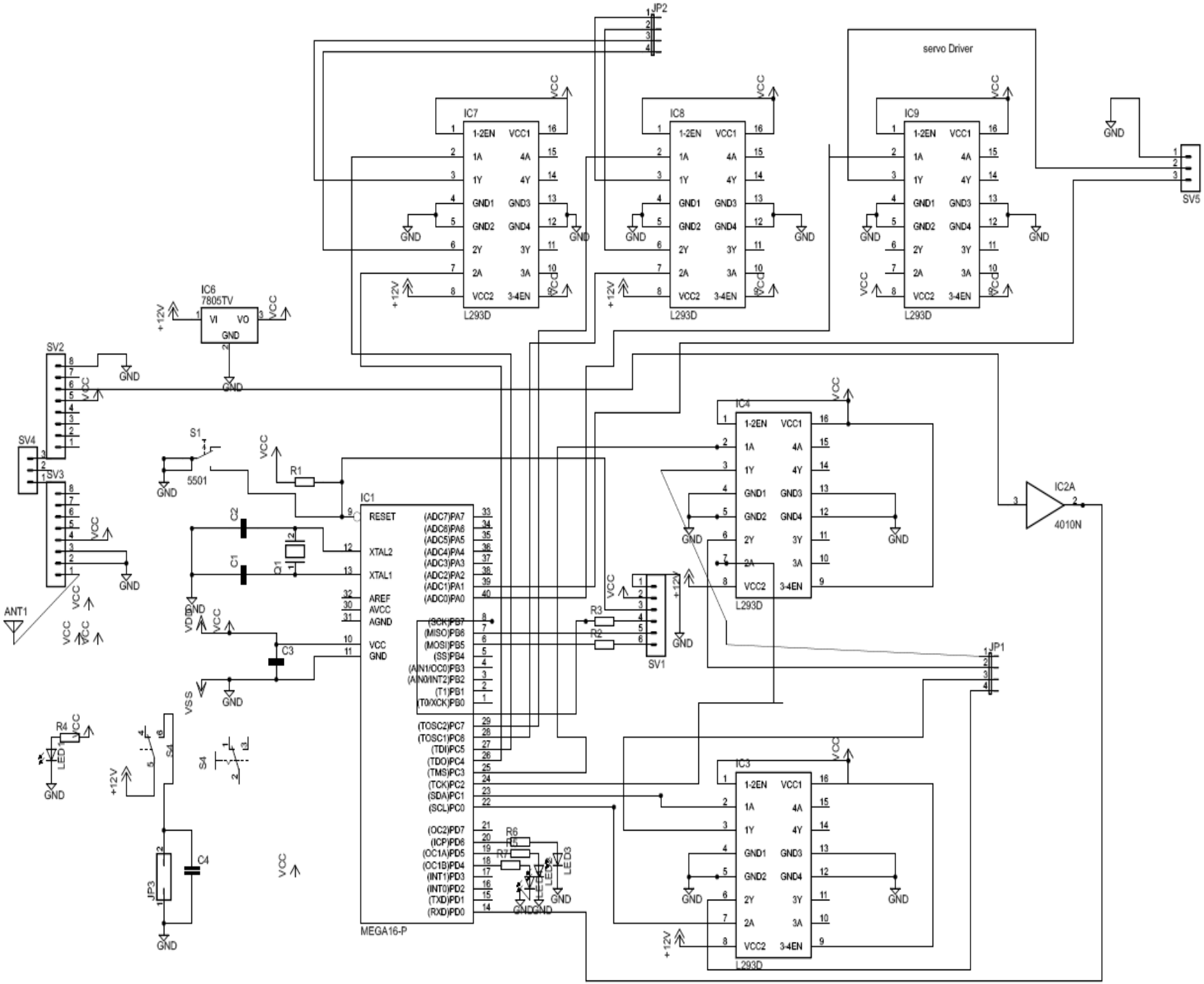


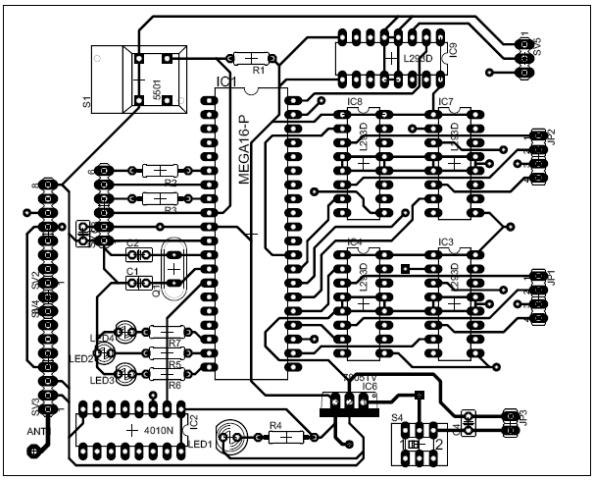
STEPPER MOTOR DRIVER BOARD

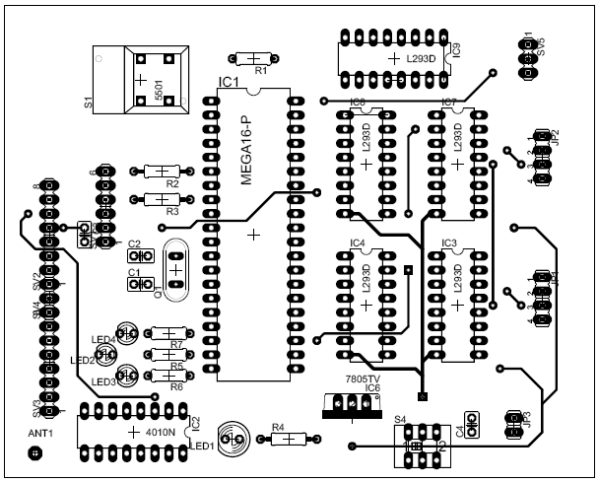
Stepper motors offer very precise way to control the position and velocity of the device in use. The stepper motors were tested with following drivers

L293D (ST Microelectronics)

Each L293D is a simple half bridge. The schematic and the PCB are as follows


servo Driver
JP2
IC7
IC8
IC9
L293D
SV5
IC6
7805TV
SV2
SV4
SV3
S1
5501
R1
IC1
MEGA16-P
RESET
XTAL2
XTAL1
AREF
AVCC
AGND
VCC
GND
C1
C2
C3
C4
Q1
ANT1
R2
R3
R4
R5
R6
R7
LED1
LED2
LED3
S4
JP3
SV1
IC4
IC3
IC2A
4010N
JP1
+12V
VCC
VSS



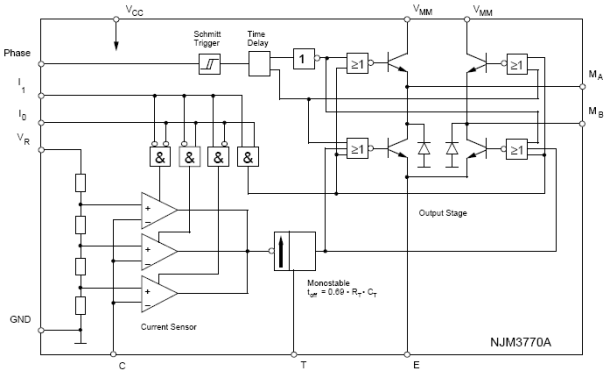


Problems Encountered

To drive each motor, we require 2 L293D ICs. Also the drivers heat up very quickly and at times burn. There is no easy heat sinking capability as opposed to L298N. The only way for heat sinking was to use ground planes which did not prove to be very efficient. Also the drivers are simple half H-Bridges, hence cannot provide constant current drive and any other advanced functionalities available in some other drivers (PWM current Control, Chopper Current Control).

UC3770A/B (Texas Instruments)

NJM3770A is a stepper motor driver, which consists of a LS-TTL compatible logic input stage, a current sensor, a monostable multivibrator and a high power H-bridge output stage. The IC works on the basis of Chopper Based Constant Current Control which is one of the best techniques available for stepper motor driving purposes. The internal circuitry is shown in the figure.



The control circuitry was very complicated. To drive each motor, we require 2 UC3770A/Bs. To control each motor individually, 6 pins from microcontroller are required per motor. This in turn results in software complications and motor is harder control.

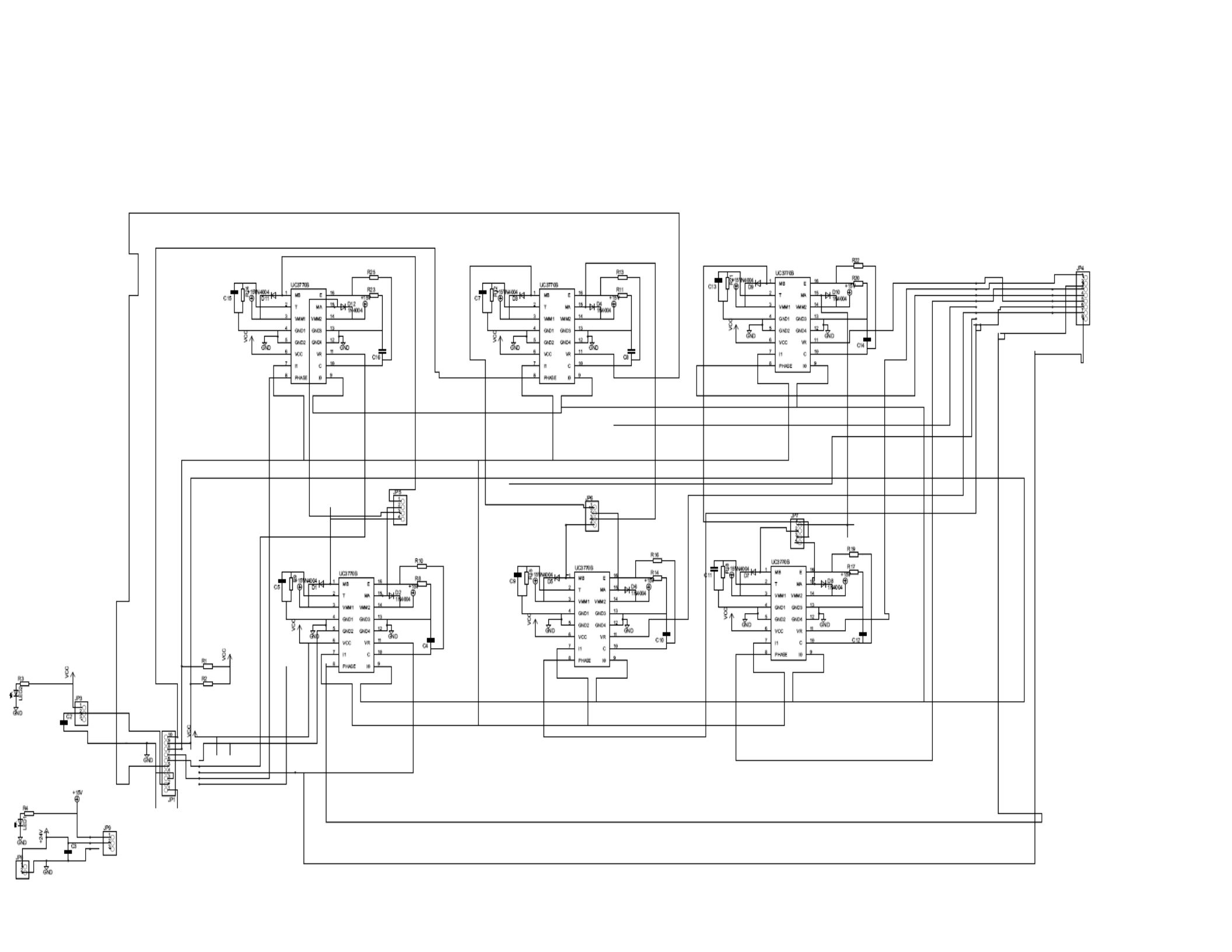

UC3770B
1N4004
VCC
GND
JP4
JP5
JP6
JP7
JP8
JP9
JP3
JP2
LED1
LED2
+15V
+24V

Figure. Schematic

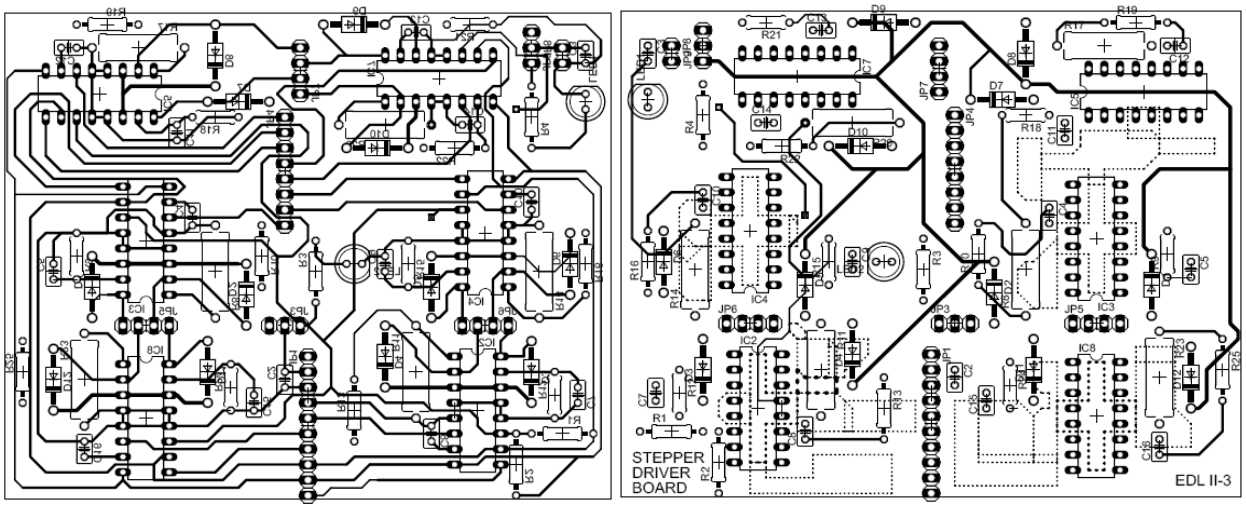


Figure. PCB

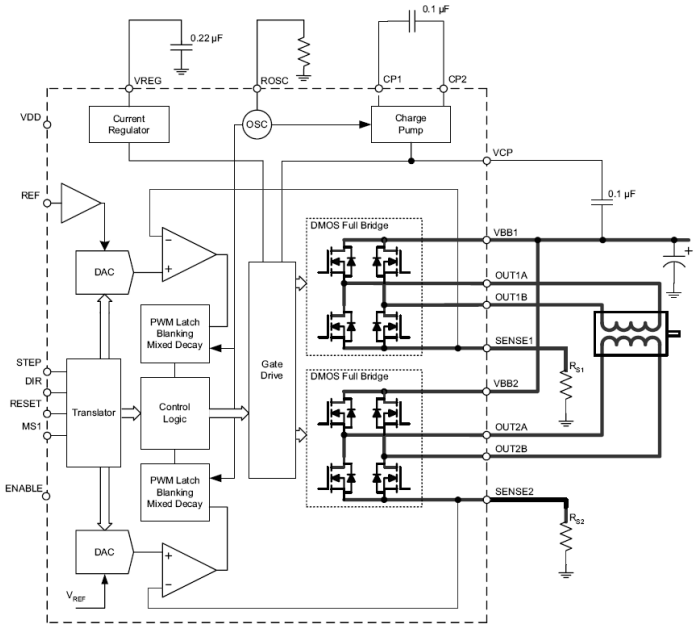
Problems Encountered

To drive a single motor we required 2 UC3770Bs. Also the external circuitry was very complex. The IC used Chopper current control and most of it had to be done by an external circuit. The software was fairly complicated and although we were able to drive the motors very precisely by using these ICs; the heat sinking issues still remained and the at times ICs could not handle large amount of current required by the motors.

A3982 (Allegro)

The A3982 is a complete stepper motor driver with built-in translator. It is designed to operate bipolar stepper motors in full- and half-step modes, with an output drive capacity of up to 35 V and ±2 A. The A3982 includes a fixed off-time current regulator which has the ability to operate in Slow or Mixed decay modes.

The Internal Circuitry looks as follows



The calculations for the functioning of driver to meet our requirements are as follows

ITripMAX = VREF / (8 × RS)

Itrip = (%ITripMAX / 100) × ITripMAX

tOFF = ROSC / 825

tBLANK ≈ 1 µs

For not exceeding 0.5 V on the Sense Pin,

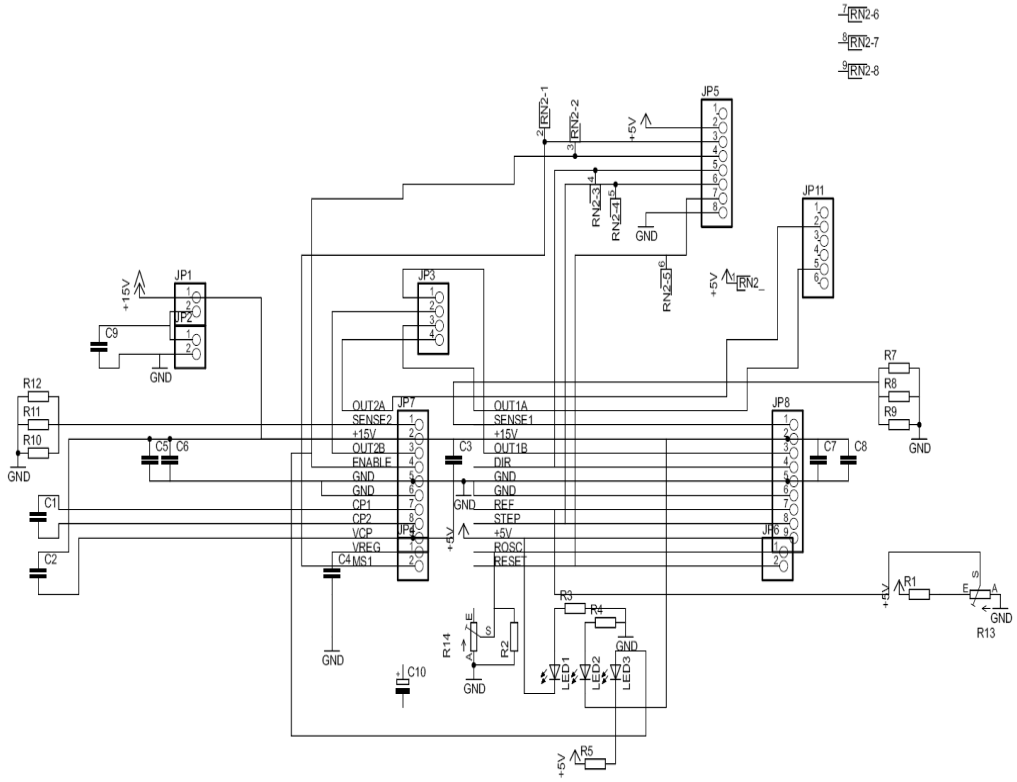
RS = 0.5 / ITRIP(max),

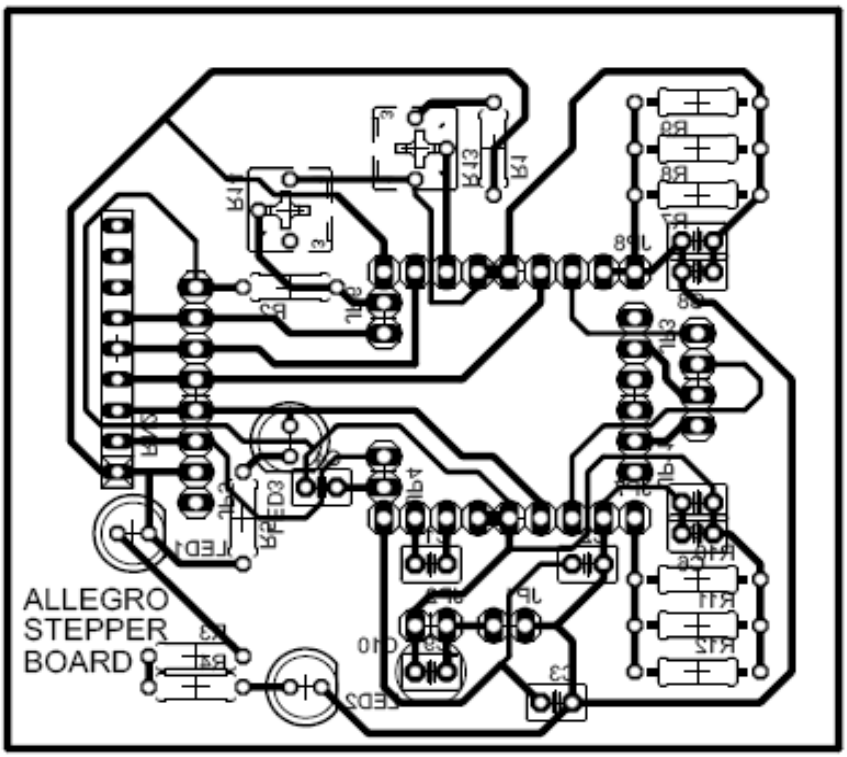
We decided according to motor requirements,
ITRIP(max) = 0.8 A
tOFF = 30 us
VREF = 2.3 V

Thus the values that we get on calculation are,
ROSC = 27 kOhm (Rounded off to the nearest standard value)
Rs = 0.35 ohm (Achieved by using 3 resistors of 1 ohm in parallel)

The circuit designed with the above design constraints showed performance meeting our requirements. The driver was interfaced with the microcontroller with 5 pins namely STEP, DIR, MS1, RESET, ENABLE. STEP and DIR decide the speed and direction respectively while RESET and ENABLE are for controlling purposes. MS1 decides the stepping mode of the circuit (Full Step/Half Step).

The schematic and layout are as shown below





THE CHOICE OF THE STEPPER MOTOR DRIVER

The allegro A3982 was the best choice available mainly due to the following reasons.

- It used internal PWM current control. The maximum allowed current can be set very precisely by perfectly designing the hardware.
- The software is very simple. Thus it is very convenient to keep track of the motion of the stepper.
- It dissipates relatively lower amount of power.
- The driver is also equipped with internal UVLO (Under Voltage Lockout) and TSD (Thermal Shutdown) circuitry which makes it safer to use.
- It automatically detects and selects the current decay mode according to stepping direction.
- It has an internal translator which minimizes the number of control lines to just 2 as opposed to 6 or 8 in other motor drivers.

All these features result in reduced audible motor noise, increased step accuracy and reduced power dissipation. Thus the stepper motor driver A3982 was used.

ROTATING SENSOR BOARD

This board contains an ATMEGA-8 microcontroller interfaced with the distance measurement sensor. The data from the distance measurement sensor (GP2D12) is sampled by ADC of ATMEGA 16. The sampled data is then sent serially to the transmitter using USART communication protocol. The board is equipped with In System Programming feature thus enabling easy debugging interface.

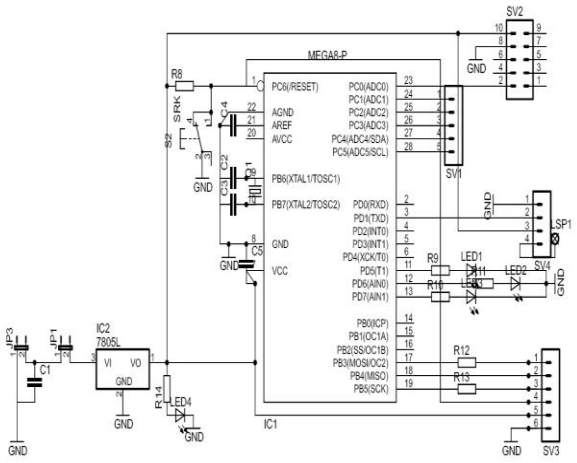


Figure. Schematic

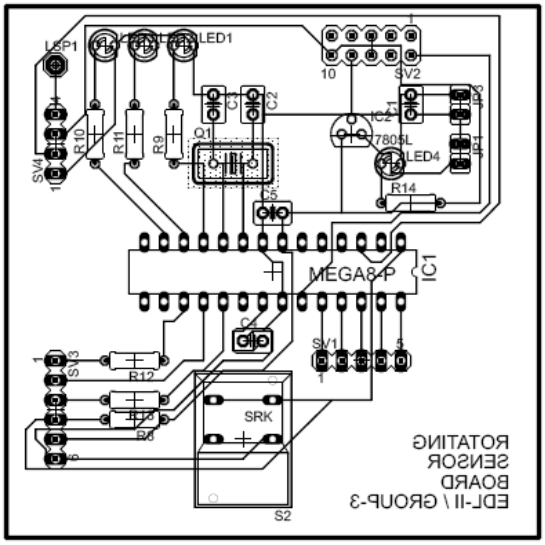


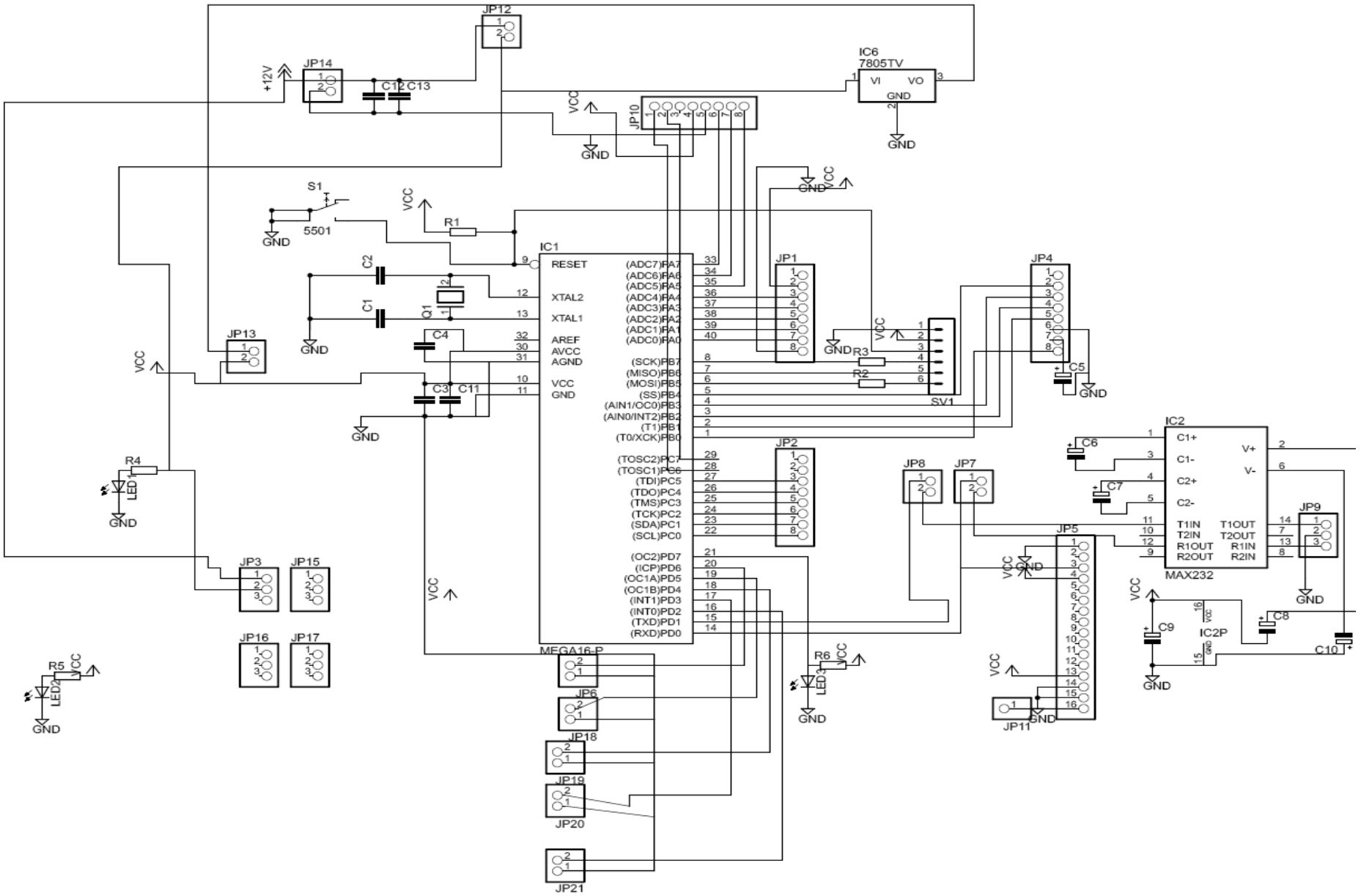
Figure. PCB

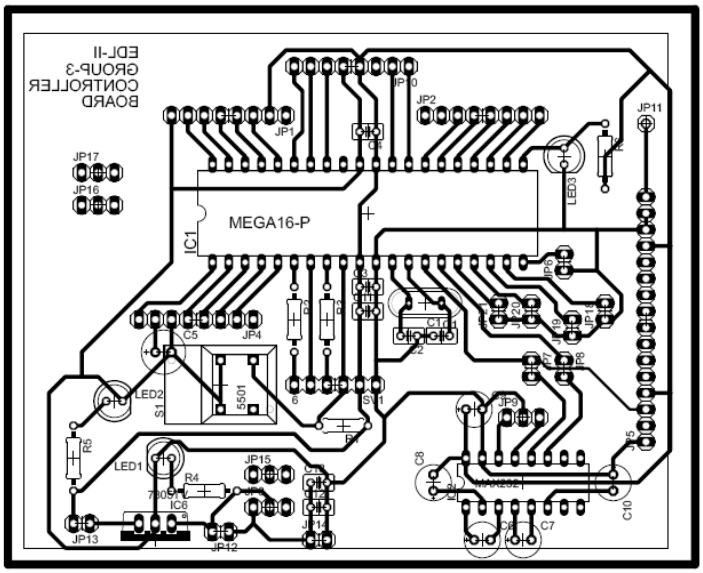
THE CONTROLLER BOARD

The controller board accomplishes following different tasks

1. Reception of data from the sensor
2. Drive the DC motors for movement of the car
3. Drive the stepper motor for the rotation of the sensor
4. Drive the indication LEDs
5. Serial communication with the Computer for testing and debugging purposes

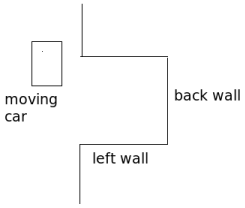
We chose ATMEGA 16 as it can handle all the above functions optimally and also has a feature of In System Programming for easy debugging and testing purposes.


JP12
JP14
+12V
C12 C13
IC6
7805TV
VI
VO
GND
JP10
VCC
S1
5501
R1
IC1
RESET
XTAL2
XTAL1
AREF
AVCC
AGND
VCC
GND
C2
C1
Q1
C4
C3
C11
JP13
JP1
JP4
R3
R2
SV1
C5
IC2
C1+
C1-
C2+
C2-
V+
V-
T1IN
T2IN
R1OUT
R2OUT
T1OUT
T2OUT
R1IN
R2IN
MAX232
C6
C7
C8
C9
C10
IC2P
JP9
R4
LED1
JP3
JP15
JP16
JP17
R5
LED2
JP8
JP7
JP5
JP11
R6
LED3
MEGA16-P
JP6
JP18
JP19
JP20
JP21
JP2
(ADC7)PA7
(ADC6)PA6
(ADC5)PA5
(ADC4)PA4
(ADC3)PA3
(ADC2)PA2
(ADC1)PA1
(ADC0)PA0
(SCK)PB7
(MISO)PB6
(MOSI)PB5
(SS)PB4
(AIN1/OC0)PB3
(AIN0/INT2)PB2
(T1)PB1
(T0/XCK)PB0
(TOSC2)PC7
(TOSC1)PC6
(TDI)PC5
(TDO)PC4
(TMS)PC3
(TCK)PC2
(SDA)PC1
(SCL)PC0
(OC2)PD7
(ICP)PD6
(OC1A)PD5
(OC1B)PD4
(INT1)PD3
(INT0)PD2
(TXD)PD1
(RXD)PD0

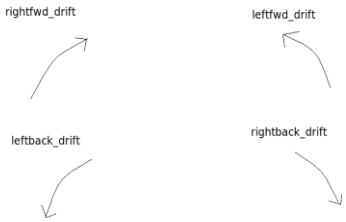


SOFTWARE DESCRIPTION

The parking arena and parking functions for the car are described in following figures.



Parking Arena

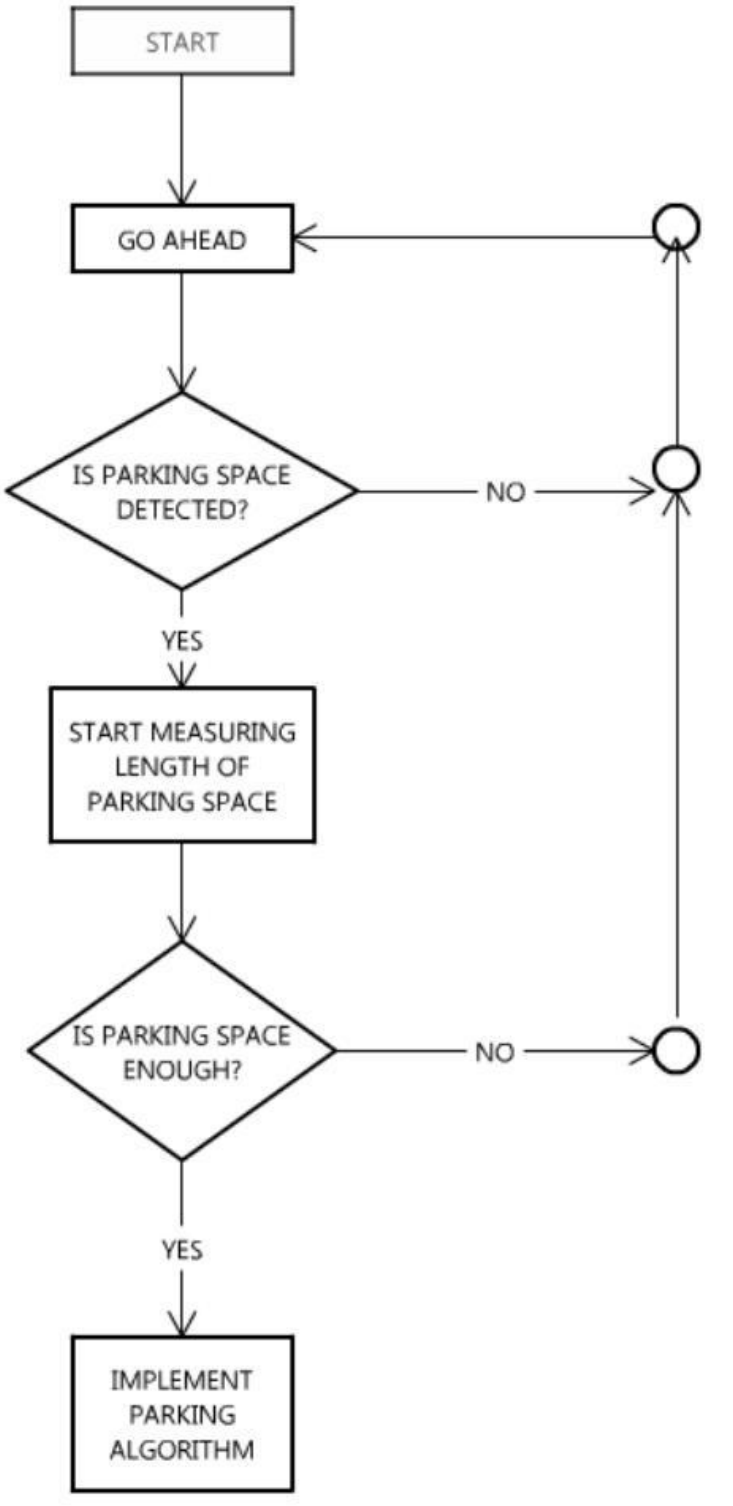


Drive functions for the car

The software is mainly divided into 2 parts: Parking Space detection algorithm and parking algorithm. The algorithms for perpendicular and parallel parking are described separately.

1. The Parking Space Detection Algorithm:

   Flowchart:



2. Perpendicular Parking Algorithm:

Flowchart: (Please see next page)

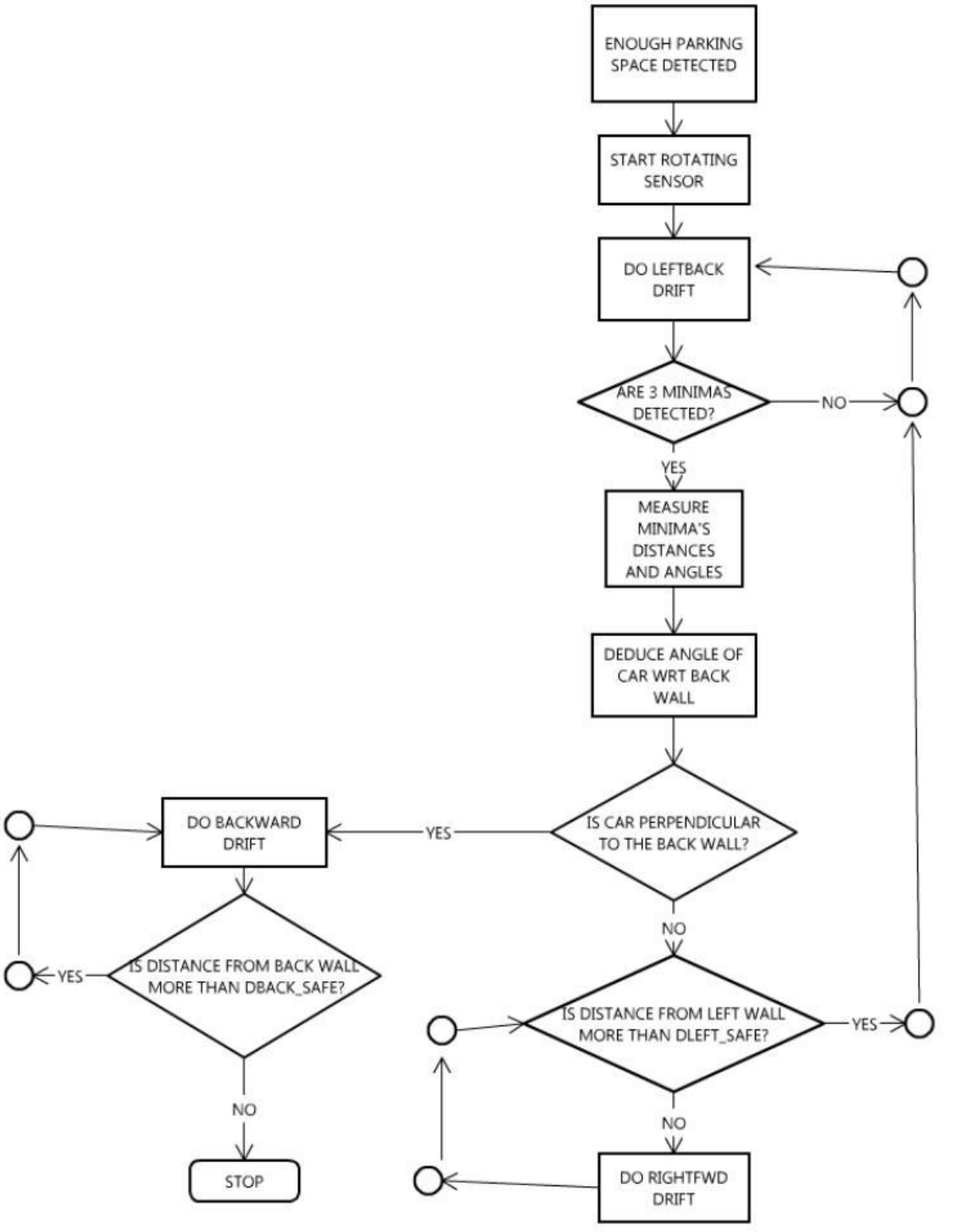
Description in Words:

Car is moving in a straight line. After every 10ms a function, chk_groove_perp is called which checks for the presence of a groove. If the groove is struck, a count is kept of the number of consecutive function calls, n_grooves, of the function chk_groove which return groove_present = 1. If (n_grooves == n_reqd_grooves), then the car starts parking.

While parking, we start rotating the overhead sensor and also start the leftback_drift and check for minimas. If good_position == 1 (ie, when the number of minimas is equal to 3), we watch out for the distances of the three minimas.

If we never get too close to the side wall, then continue leftback_drift until we get aligned to the back wall. Once the angle is aligned, then we reverse the car until minima2 lies at a distace of min2_safe.

If however, minima 1 is too close (we're in the danger of hitting the left wall), then start rightfwd_drift. In the new position, again check for minima 1, if we're still too close to the wall then again rightfwd_drift and so on until minima 1 is at a distance greater than min1_safe. When min1_safe is achieved, do a leftback_drfit. Now, keep polling on the location of minima2. When minima 2 starts lying in the range of 80-100 degrees, start doing reverse. Keep polling on the distance of minima2. Stop doing reverse when minima 2's distance equals to min2_safe.


ENOUGH PARKING SPACE DETECTED
START ROTATING SENSOR
DO LEFTBACK DRIFT
ARE 3 MINIMAS DETECTED?
NO
YES
MEASURE MINIMA'S DISTANCES AND ANGLES
DEDUCE ANGLE OF CAR WRT BACK WALL
IS CAR PERPENDICULAR TO THE BACK WALL?
YES
DO BACKWARD DRIFT
IS DISTANCE FROM BACK WALL MORE THAN DBACK_SAFE?
YES
NO
STOP
NO
IS DISTANCE FROM LEFT WALL MORE THAN DLEFT_SAFE?
YES
NO
DO RIGHTFWD DRIFT
Made with lovelycharts.com

3. Parallel Parking:

Flowchart: (Please see next page)

Description in Words:

Car is moving in a straight line. After every 10ms a function, chk_groove_parallel is called which checks for the presence of a groove. If the groove is struck, a count is kept of the number of consecutive function calls, n_grooves, of the function chk_groove_parallel , which return groove_present = 1. if (n_grooves == n_reqd_grooves), then the car starts parking.

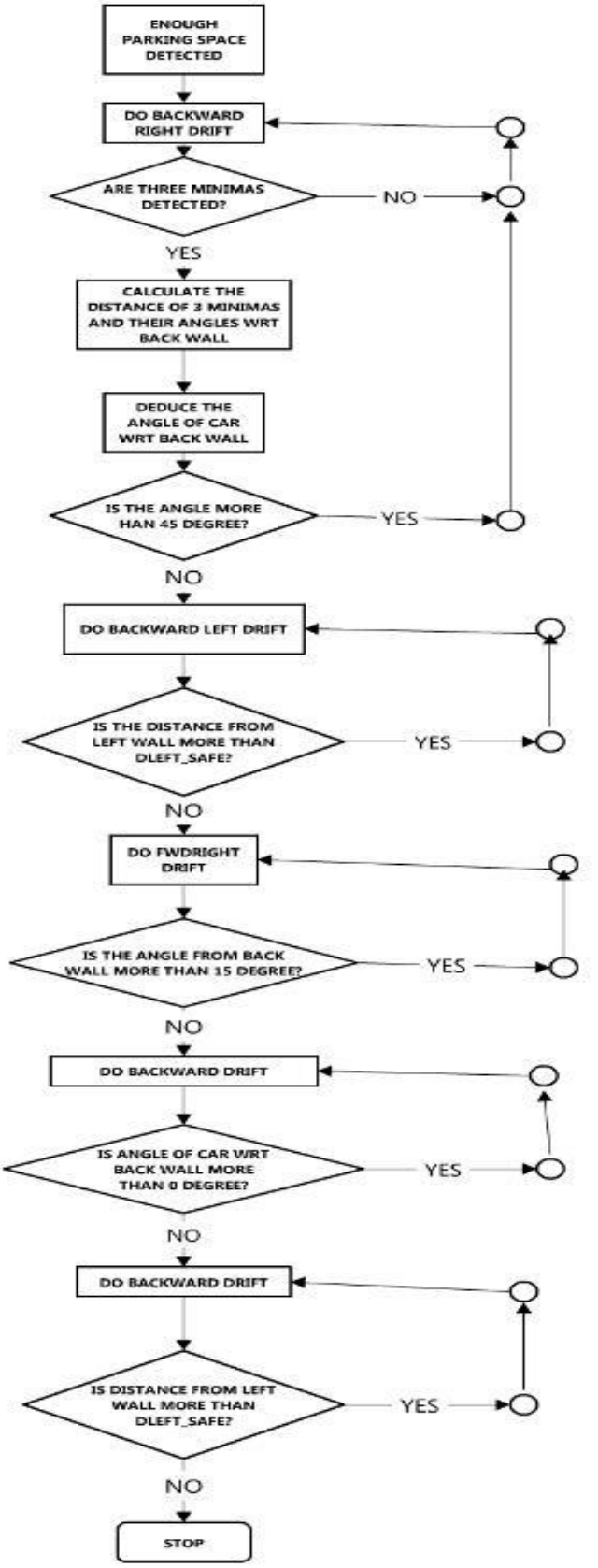
While parking, we start rotating the overhead sensor and also start the leftback_drift. the leftback_drift continues till we get in the good position (number of minimas equal to 3).

Thereafter, while continuing the rightback_drift, we start checking on the minimas' distance and angles. When the angle from back wall is more than 45 degree, we start leftback_drift.

This is continued till the distance from the left wall is greater than some threshold distance. After which, we do forwardright_drift, till the angle of the car wrt back wall is less than 15 degree.

Then we do bakwardleft_drift till the angle of the car wrt back wall is less than or equal to zero degree. (car becomes parallel to the back wall).

Then we drift the car backwards till it reaches within the safe distance from the left wall.



Code in C++

```
int chk_groove_perp()
{
int groove_present=1;
for(i=0;i<num_read_1rev;i++)
if(store_1rev[i] < groove_perp_depth)
groove_present = 0;
return groove_present;
}


void perp_parking()


{

rotate_sensor = 1;
while(good_position != 1)


        {
          leftback_drift();

            }




        while( (min[0][1] > min1_safe) && (min[1][0] > 150) )          //150 corresponds to 90
degrees


        {
          leftback_drift();

            }


        if(min[0][1] < min1_safe)
{
while( (min[0][1] < min1_safe + delta) && (min[1][0] > 150) )
rightfwd_drift();

if(min[1][0] > 150)
```

```
                {
                  while(min[1][1] > min2_safe)
                  reverse();
                 }

                else


                {
                     while(min[1][0] > 150)
                    leftback_drift();


                   while(min[1][1] > min2_safe)
                    reverse();
                       }

        }


        else


                while(min[1][1] > min2_safe)


                        reverse();

}


int chk_groove_parallel()


{
      int groove_present=1;
      for(i=0;i<num_read_1rev;i++)
      if(store_1rev[i] < groove_depth_parallel)
        groove_present = 0;
      return groove_present;

}


void parallel_parking()


{
      rotate_sensor = 1;
         while(angle != 45)
```

```
          {
            rightback_drift();

          }


          while (min[1][1] > min2_safe + delta1)
          {
         leftback_drift();
           }

          while( angle!=15 )
          {
           rightfwd_drift();
           }

       while (angle!= 0)

{

        Leftbak_drift();

}

          While (min[1][0] > dleft_safe)


          {

                              reverse();

          }
stop();

}
```

TESTING AND CALIBRATION PROCEDURES

To test and debug the codes, the following procedures were used

1. Testing of Wireless Data Transmission and Reception
2. Calibrating the Distance Measurement Sensor

TESTING OF WIRELESS DATA TRANSMISSION AND RECEPTION

There is one way communication of data between the rotating sensor board and the main controller board. The data is sent using a transmitter and receiver module which do ASK modulation of the frequebcy at 315 MHz.

Data was transmitted at a baud rate of 1200 bits/sec using the UART encoding scheme.

The transmission of data between the transmitter and the receiver was tested for the following:

1. Error rate in sent and received data
2. Number of data points successfully transmitted in a second

Encoding of the data:

The data was sent in the UART format using the Rx and Tx pins of the microcontroller. The Tx pin was directly connected to the transmitter and the receiver was connected to a buffer(Schmidt trigger) to amplify and strengthen the signal and then connected to the Rx pin of the other microcontroller.

However sending and receiving data in the UART format leads to many erroneous reception of data. The receiver absorbs random signals from the air and noise is received. To filter out this noise, an encoding scheme was employed to the data to ensure that only valid data is received.

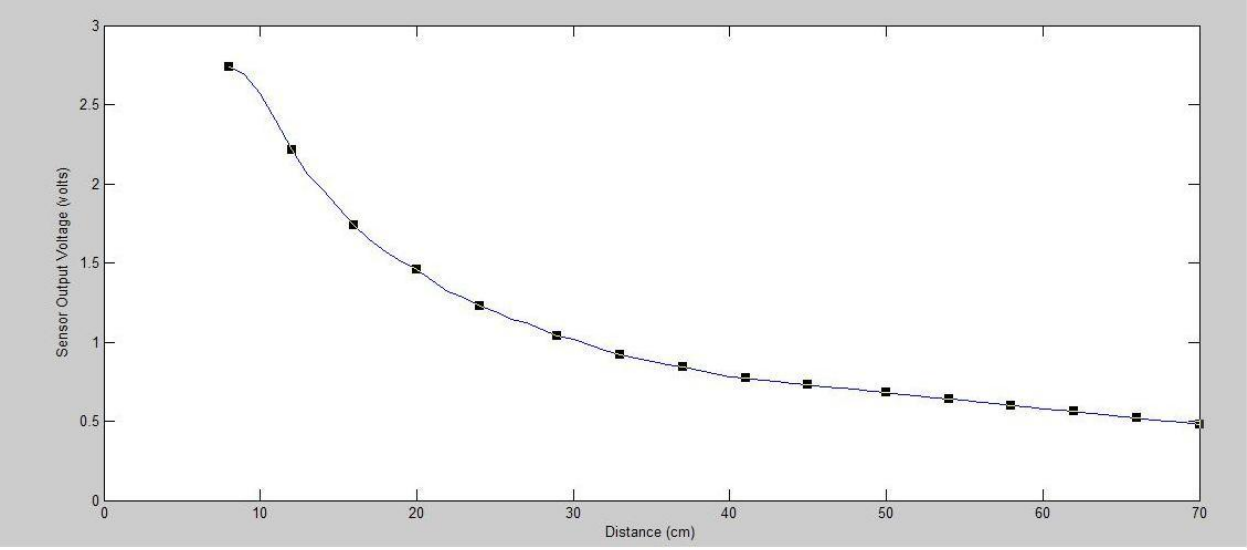
This encoding is done by adding start and stop bits to the data. The following packet of data was sent and received.

START BYTE1
START BYTE1
ACTUAL DATA
STOP BYTE1

Only when this sequence of bytes is received at the receiver, the data received is considered as valid. This encoding scheme was able to remove 100% noise at the receiver end. However it leaded to bloating of the data by 4 times.

CALIBRATING THE DISTANCE MEASUREMENT SENSOR

Following figure shows the results of characterization of the sensor.



DESIGN RESULTS

Execution Speed

The response time of the sensor was 40ms. This time was fast enough for the signals to be processed in real time. There was a tradeoff between speed of the car and the torque available. We chose the 60 RPM DC motors for the optimum performance.

Accuracy

Distance Sensors
Distance sensors were fairly accurate to +/- 2cm.

Parking Space Detection
Parking space can be very accurately detected by our algorithm.

Parking Algorithm
Parking Algorithm for both parallel and perpendicular parking was working successfully with some minor ambiguities.