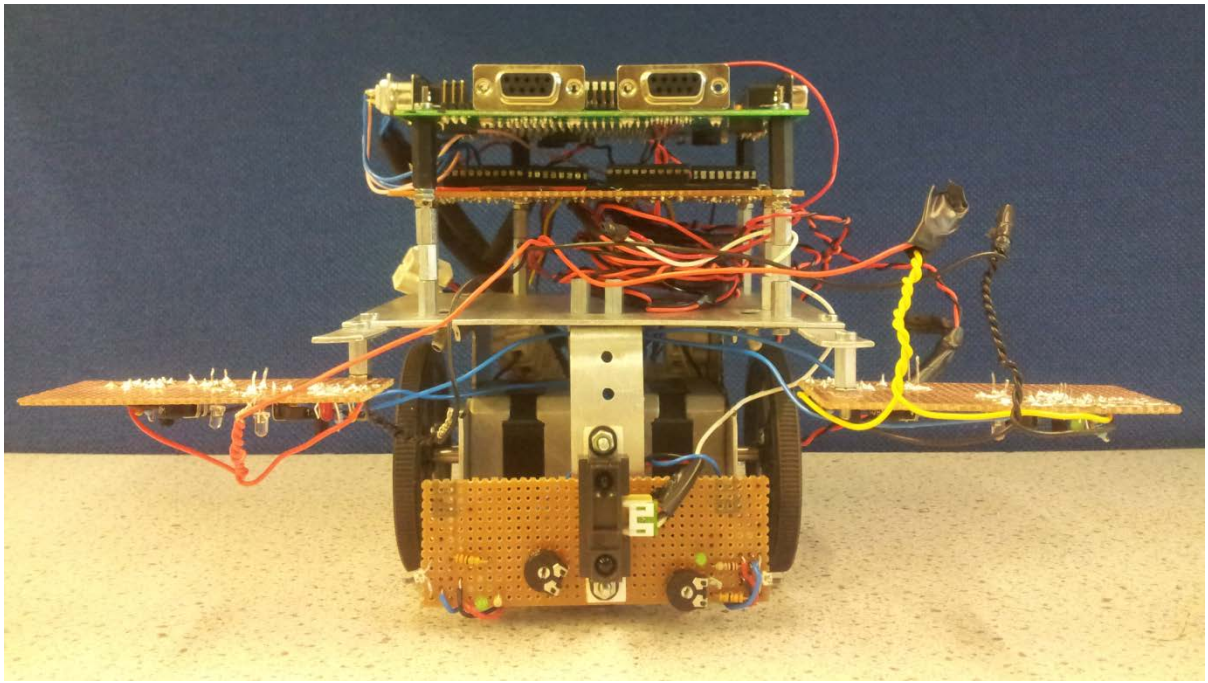
Echelon-V: Micromouse-2012 Report

University of Westminster

Module: EECT510-Embedded Microprocessor System Project

Module Leader: Dr. Martin Giles



Group Members:

- **Sameer Sunani**
- **Anjani Maddala**
- **Jules Pinto**
- **Naeem Hosany**
- **Deujean Cabey**

Contents

Introduction

This report explains the efforts put in by our group Echelon-V to create a micro mouse. At the end of the semester our mouse has taken part in the university micro mouse competition and ended up in the second spot. Our approach was to keep things simple and execute plans with a high level of engineering aptitude.

The micro mouse is an autonomous robot which is used to solve a maze using an algorithm (Lee Algorithm, Left Hand Algorithm etc...). The mouse only has the knowledge of the end square.

The international micro mouse challenge began back in the 70's and takes place across the world including countries such as the UK, U.S.A, Japan and Singapore.

Micro mouse competition: The international micro mouse competition requires the mouse to solve a maze of the length 16x16 with each square being 17cm. The base of the maze is painted black to absorb any ambient noise. The walls are painted white with their top being red to allow good reflection of the sensors. Also the target square is the centre square of the maze.

Modifications made: In the university competition our mouse took part in, the maze was reduced to 8x8. The target square is the first square in the top right hand corner of the maze and the start square is the last square of the bottom left hand side of the maze.

Our mouse

The figure on the right is our mouse.

The following are the features of our mouse:

- Has 9 sensors
- 1 distance sensor, 8 proximity sensors
- Two unipolar motors
- Uses a PIC16f877A microcontroller
- Uses Left Hand Algorithm to solve the maze

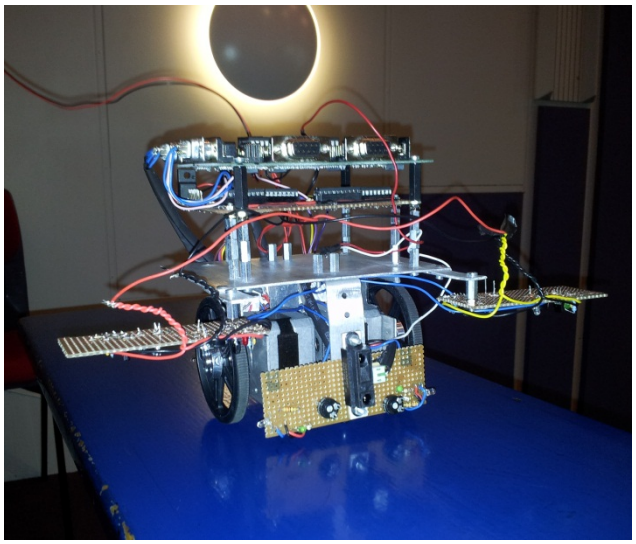


Figure-1: Our Mouse

1- Specifications

1.1- Maze

The maze given to solve measures 9 squares wide and 9 squares in height, each square measures 17cm by 17cm and has walls approximately 4.5cm wide. Each wall has a red painted top that can be used with infrared emitters and the sides of the walls are painted white. The surface of the maze is coated in black paint, this helps to minimise noise cause by stray reflections. Apart from navigating the maze walls, the mouse has to move over the surface competently to avoid collision.

1.2- Chassis and Wheels

At first, we used the wheels and chassis that were given to us, shown in Fig 1.2.1.We decided to avoid building a custom chassis in the interest of time, as you can probably notice, the wheels did not come with any sort of tyres or grip. This is not a good design, as the mouse will be very likely to skid when stopping or if there's enough torque being supplied to the wheels; it can cause the wheels to spin!

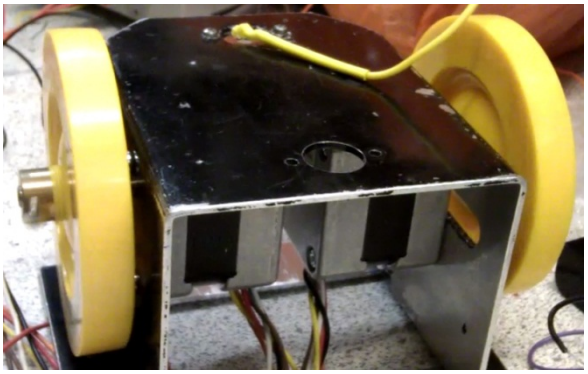


Figure 1.2.1 - Yellow wheels without grip

Thus the mouse probably would not move at all, or when it does manage to grip on the maze surface, it may move uncontrollably! Therefore, we shopped around for wheels primarily with good grip, and came up with a new set of wheels; very sturdy, providing very good grip made by Pololu.These wheels are 70mm in diameter and 8mm in width and come with silicone rubber tyres for very good traction.

Figure 1.2.2 - Pololu 7cm wheels with tyre removed

Throughout our project, we made amends to our chassis; mainly because our experts in sensor design thought that moving and adding new sensors to the mouse would benefit it. The diagrams below show the original blueprints for our mouse.

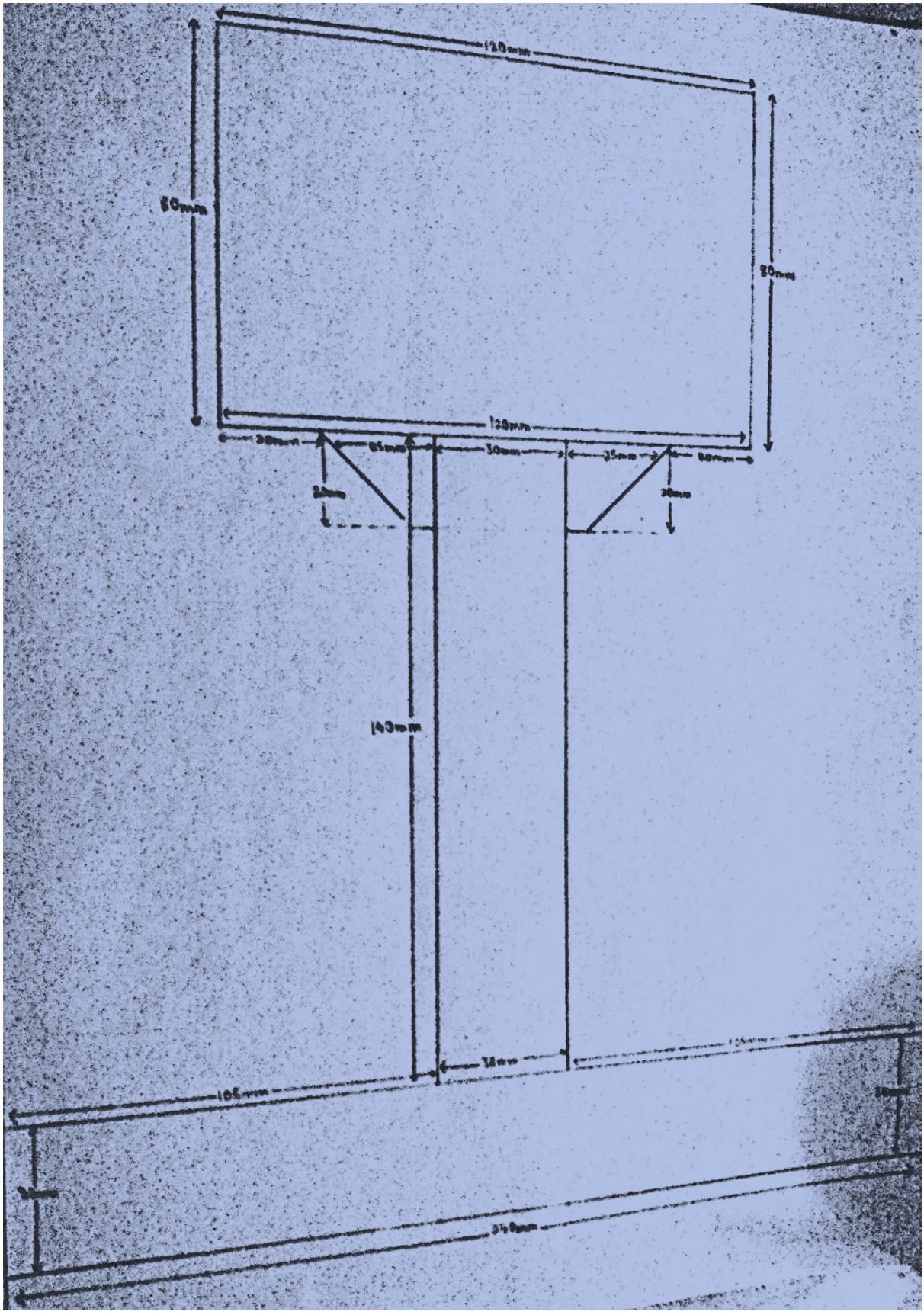


Figure 1.2.3 - Chassis with wing - Top view

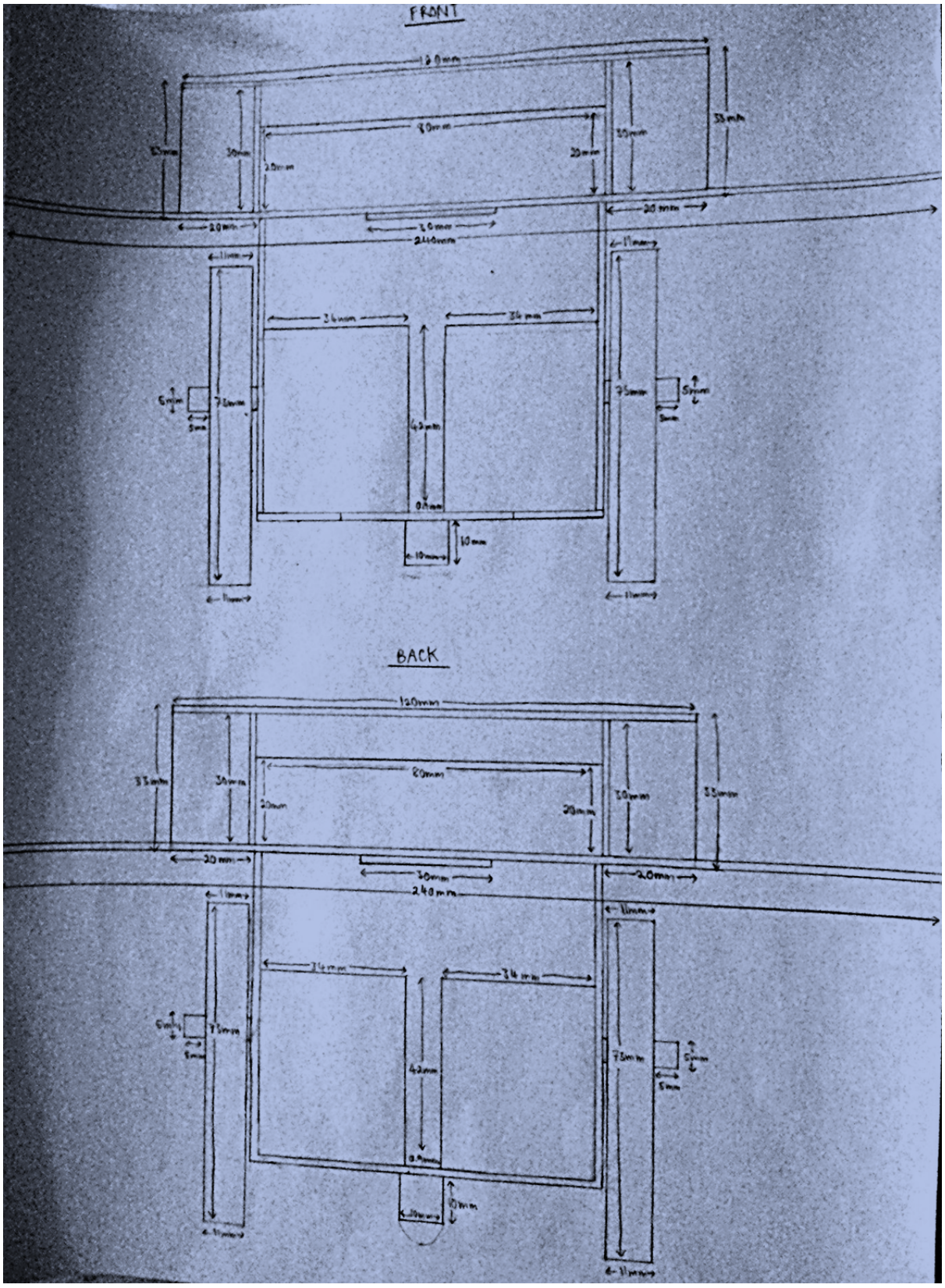


Figure1.2.4 - Chassis - Front and Back View

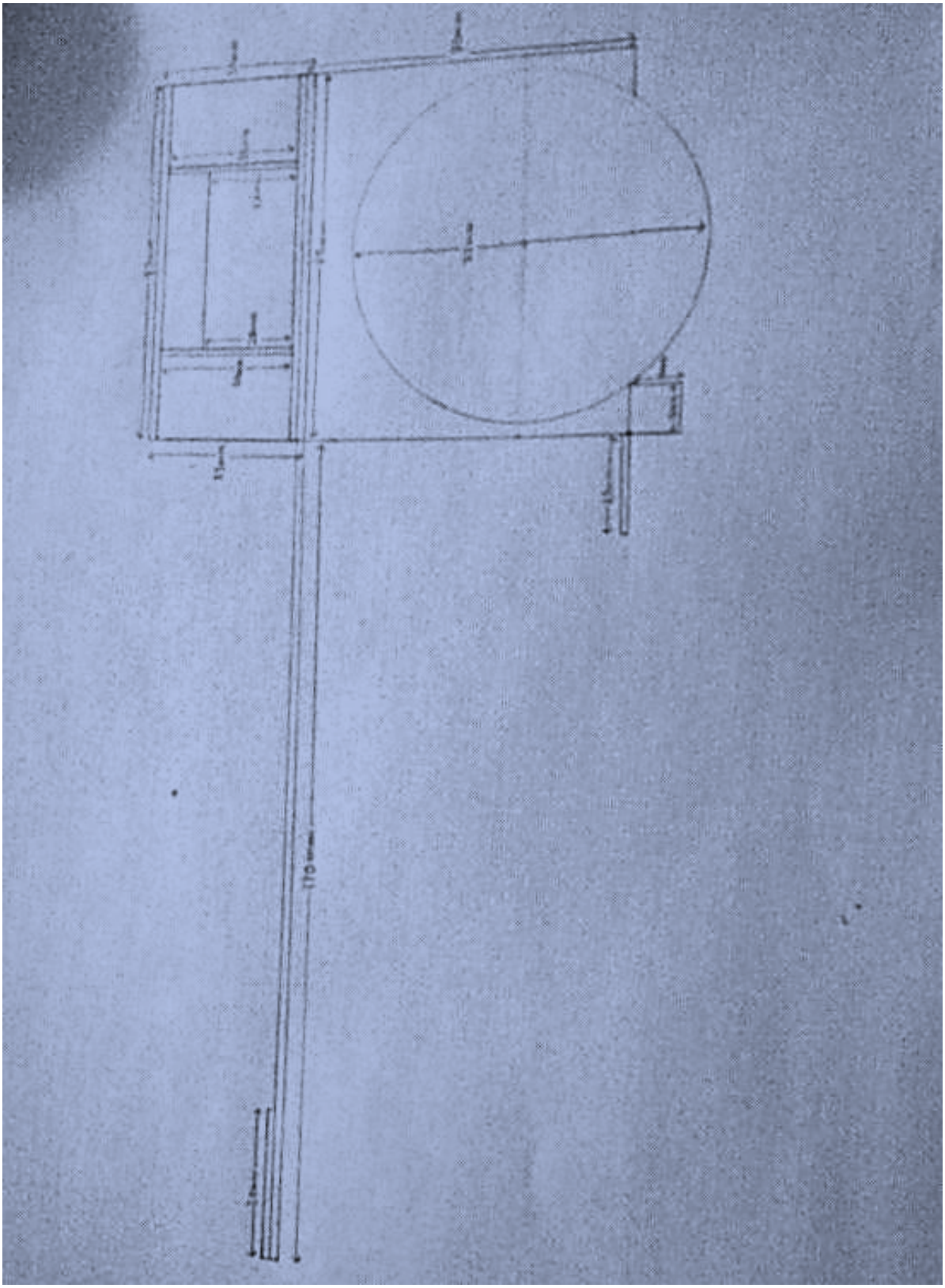


Figure1.2.5 - Chassis - Side View

Due to the sensor redesign, we had to make certain changes to the chassis. First of all, the wing design was abandoned and 'collision'/wall detection sensors were installed at the front of the mouse. Fig 1.2.8 shows the new design.

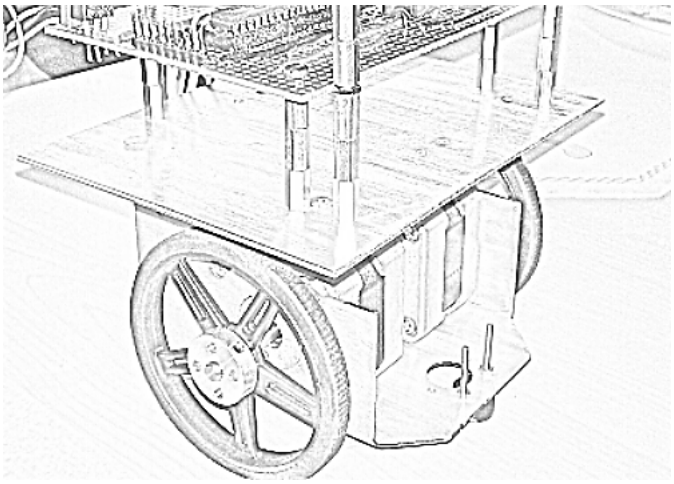


Figure 1.2.6 - Chassis with wing removed

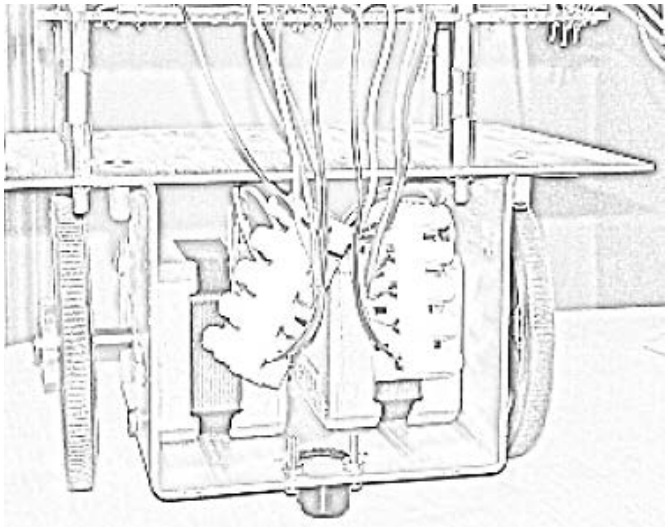


Figure 1.2.7 - Chassis - Back View

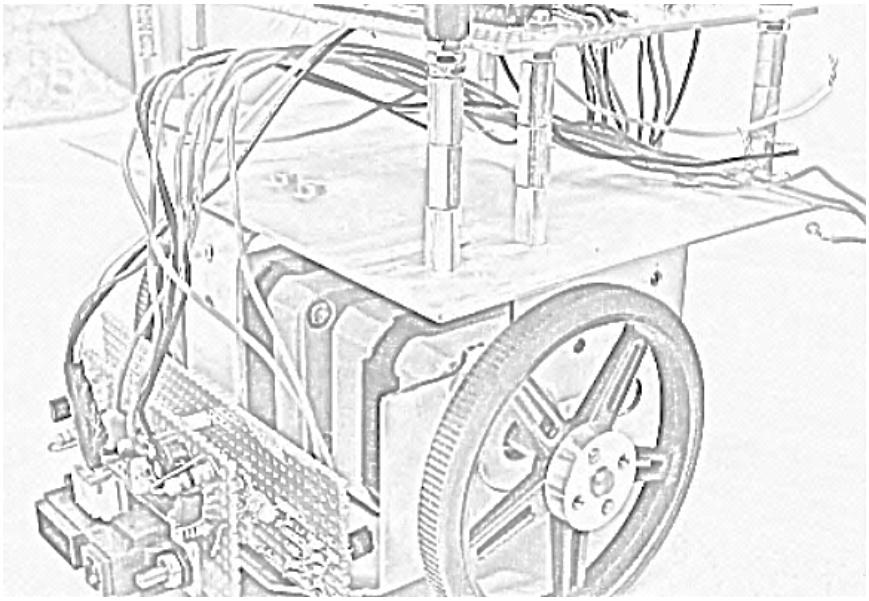


Figure 1.2.8–Front View showing wall and collision sensors and support to chassis

After many hours of testing, we realised that we had to re-design the sensors again and therefore amend the chassis one more time to accommodate the new sensors which will be placed at the back of the mouse. We removed the wall detection sensors at the front and cut a slot in the back of the chassis in the process. Fig 1.2.9 and 10 shows the new look.

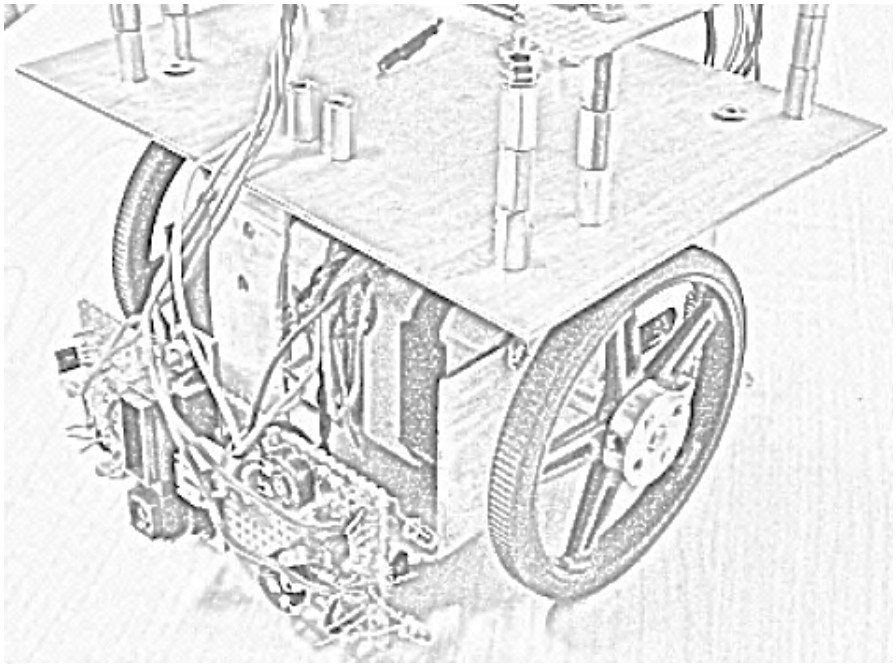


Figure 1.2.9 - Chassis - Front View showing 'collision' sensors, wall detection sensors have been removed

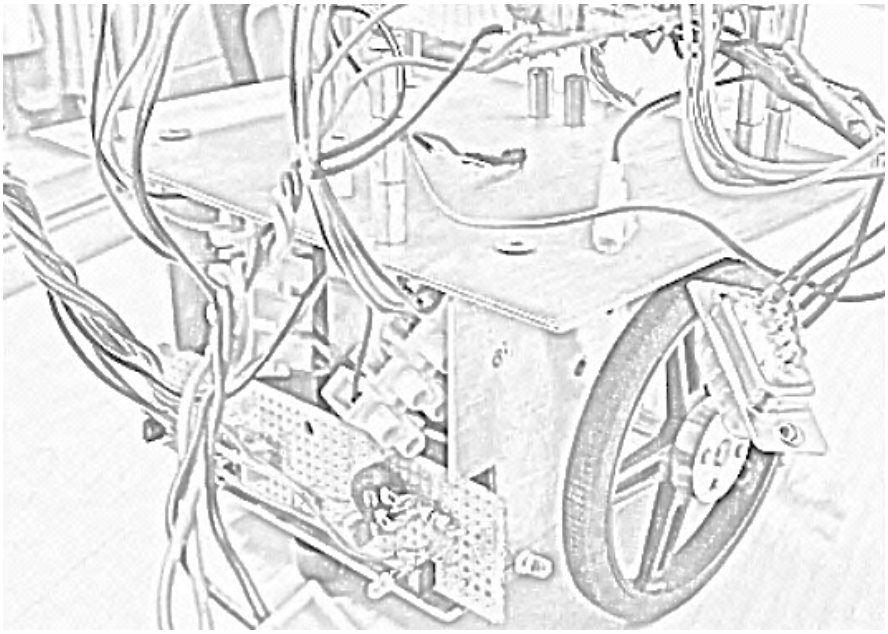


Figure 1.2.10 - Chassis - Back View showing new wall sensors

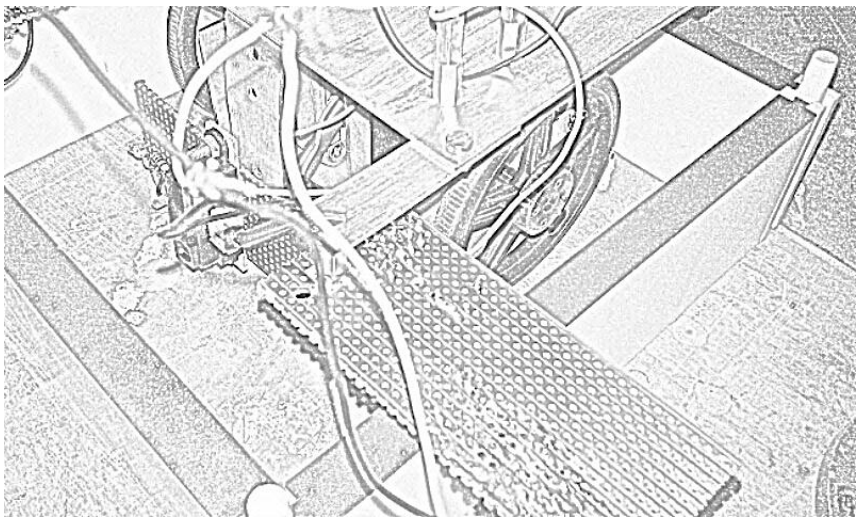


Figure 1.2.11 - Close up shot of the left wing

Towards the end of the project, we realised that we needed more sensors and therefore we implemented a unique wing design whereby we used the circuit board itself as the wing. This is was the last change we made to the chassis.

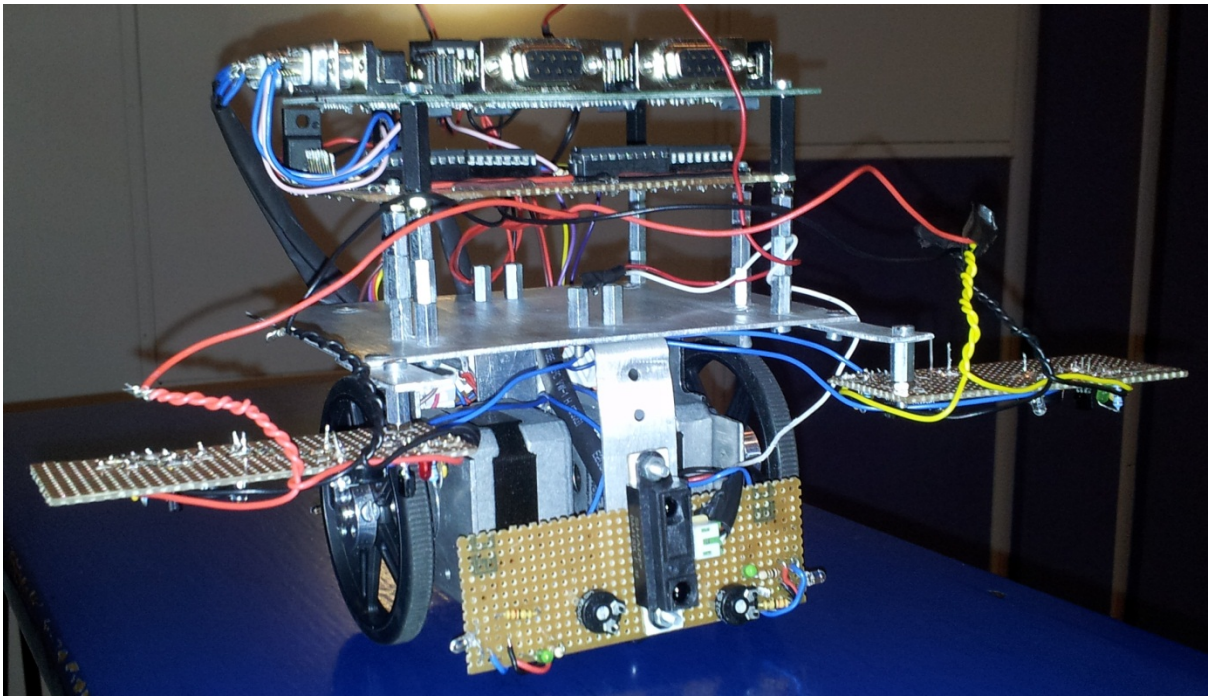


Figure 1.2.12 - Shows wing design and front sensor

1.3- Sensors

Our mouse uses nine (9) individual sensors for its 'eyes' in the maze, keeping in mind that the sensors need to be used as wall detectors, detect proximity to wall and distance measuring.

Sensor circuits consist of IS471 receiver and Op165d emitter used for Proximity, and wall detection, both receiver and emitter work in union and have a variable resistor for adjusting the range of about 4cm max, also a Sharp GP2D120XJ00F used as a forward distance sensor and has a range of 4-30cm.

Figure 1.3.1 - left emitter, right receiver

Figure 1.3.2 - Sharp Distance Sensor

1.4- Batteries

After an in-depth group discussion, we decided that we will power up our mouse with a Lithium Polymer battery. Initially, we couldn't decide between the AA Nickel Metal Hydride (Ni-MH) rechargeableand the Lithium Polymer battery, so we put the specifications of each side by side and decided which one would be more suitable for our mouse.

Figure 1.4.1 - OverLander Lithium Polymer Battery

Below are some comparisons between the two, which eventually lead us to using the Lithium Polymer.

Specifications	Energizer AA Ni-MH x 12	OverLanderLithium Polymer
Maximum discharge current (C)	2 @ 4.8A	**25(continuous)** **40(in bursts)**
Weight (grams)	360	**217**
Dimensions(mm) – Volume(cm^3)	**50.5x7x7(each) – 297**	104x35x30 – 1092
Total Voltage (V)	1.2 x 12 = 14.4	**14.8**
Capacity (mAh)	**2300**	2200

Figure 1.4.2 - Battery specifications

Assuming a worst case scenario, we obtained all the **peak** currents from the datasheets of the major components that we'll be using, below are the results:

Component	Amount	Avg req.	Max Rating	Total(avg)	Total(max)
PIC	1	220uA	220uA	220uA	220uA
MOTORS	2	0.16A	0.16A	0.32A	0.32A
SENSORS (RECIEVERS)	8	3.5mA	50mA	28mA	400mA
EMITTERS	8	50mA	3A	400mA	24A
TOTAL				**748.22mA**	**24.72A**

Figure 1.4.3 - Component current usages

Even though the AA was the winner in certain aspects that we looked at, including volume and capacity, we believed that it may not be able to handle the amount of current we have just looked at in the above table - 24.72A, even if it is just a short burst. So instead, we went for the Lithium Polymer.

The LiPo battery worked very well, and lasted over 30mins while testing. However, very unfortunate for us, the battery swelled and blew up while charging, and since the catastrophe, we opted to use the DC power supply instead.

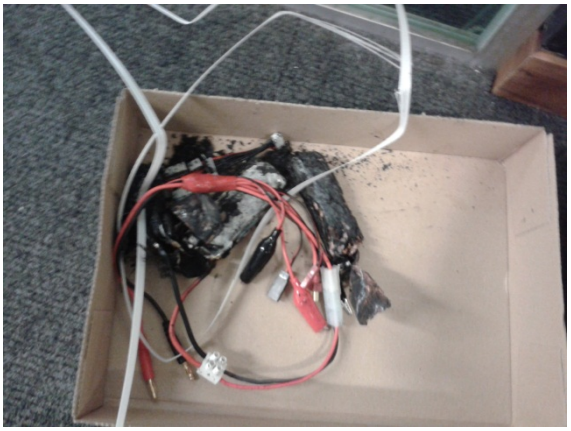


Figure 2.4.4 - Exploded battery

1.5- Motors

The motor used in the design is that Y129-5. Y129-5 is a two phase unipolar stepper motor. The figure below shows the motor used in the design.

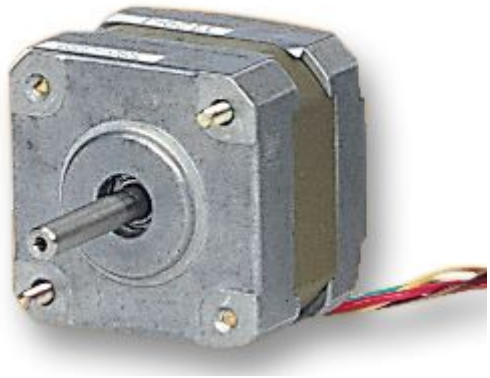


Figure 1.5.1: Y129-5 Stepper Motor [1]

The specifications for the motor are given in the table given below:

Stepping angle	**1.8°**
Holding torque	**9Ncm**
Mass	**0.22Kg**
Current specification	**0.16A**

Table 1: Motor Specifications

1.6- Calculations

By taking the total mass of the mouse we can calculate the force which is required to move the mouse and the torque that needs to be generated for this mass to be moved in the maze.

We first start off by finding the total mass of the mouse. This is done by adding up the mass contribution of each individual component of the mouse which is as follows:

1. Contribution from the motors, M_m= 2*0.22=0.440kg
2. Contribution from the chassis, C_m = 0.100kg
3. Contribution from the rest of the components, O_m = 0.050kg

Total Mass, M_T = 0.440+ 0.100+0.050 =0.59 kg.

Total force $F_T = M_T*a + M_T*\mu*g$

Where a is the acceleration, g=9.81 ms^{-2},μ is the coefficient of friction which is assumed to be 0.3 for our maze.

For a realistic estimate of a = 0.15ms^{-2}

Total force, F_T = (0.59)*0.15+(0.59)*0.3*9.81=[(0.59)*3.093] N.

Therefore, force per wheel=F_T/2=[(0.59)*3.093]/2==[(0.59)*1.5465] N.

The on load Torque, T_T =F_T*r

where r is the radius of the wheel. For our mouse ,r = 7.5 cm/2 =3.75 cm

Therefore T_T=[(0.59)*3.093]*3.75 =[(0.59)*11.60] N-cm.

Therefore torque per wheel = T_T/2=[(0.59)*5.80]N-cm.

1.7- Microcontroller Used

The microcontroller used is a Microchip 8-bit CMOS FLASH programmable chip. It has the following features:

- 40 pins, spilt up into 5 ports
- operating voltage of 2 - 5.5 volts
- operational temperature range -40 to 125 deg(C)
- 3 timers, 2x8-bits, 1x16bits
- 2 Capture and compare peripheral
- 2 comparators
- 8-bit and 16-bit ADC
- 368 RAM BYTES
- 256 Data EEPROM BYTES

We opted to use the Matrix Eblock which allowed for easy development, using the eblock we had better access to pins on the pic, we could easily change connections if needed. The eblock also had a access to 12 volt

DC and 5 volt DC built in and other features include on-board resetting, and icd2 support.

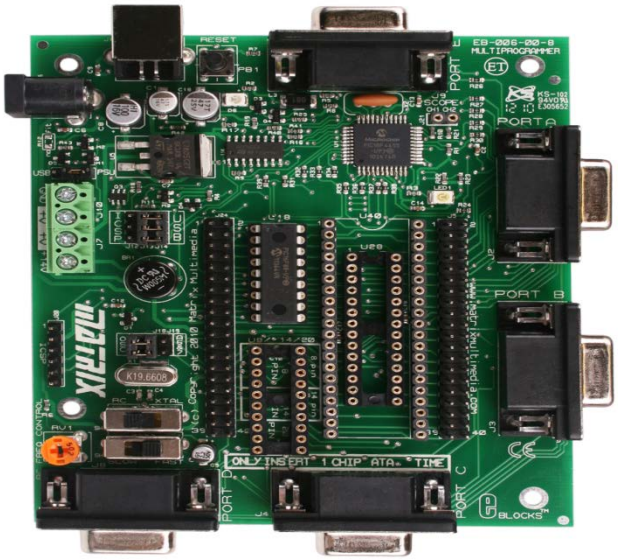


Figure 1.7 eblock used[2]

2- Design Approach

2.1- Chassis Design Implementation

In developing the mouse, precautions had to be taken to reduce problems in the electronics and momentum. The following is a list of additional precautions taken:

- Replacing metal mounts – the metal mounts in the chassis caused electrical shorts in the electronics we had to remove the appropriate mounts, mainly the ones around the eblock, and motor circuit and replace them with non-conducting mounts.
- Ball casters – the ball bearings initially were set with spacers to balance the height of the chassis with wheels but this reduced the rocking and overall stability of the mouse movement but due to the surface of the maze having slight elevation in some areas the mouse did not have enough momentum to overcome them, a change of the heights of the ball bearings helped solve this issue, removing the was spacer in the front gave the mouse a slight unevenrocking to overcome the surface. We chose to lower the front spacer because of the forward momentum and centre of gravity, the front ball bearing will be slightly raised over the surface.

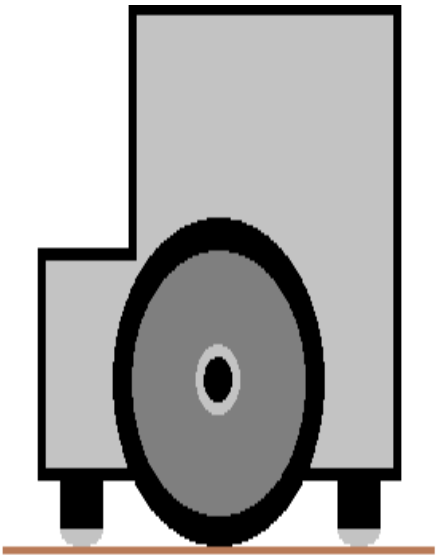


Figure 2.1. Balanced Mouse has problems with surface

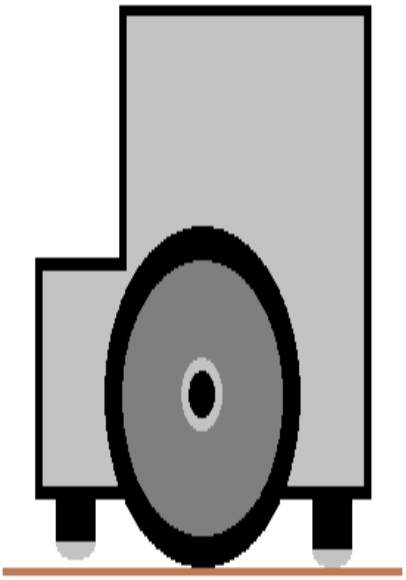


Figure 2.2.2 Unbalanced Mouse “rocks” over bumps

2.2- Final Hardware Schematic

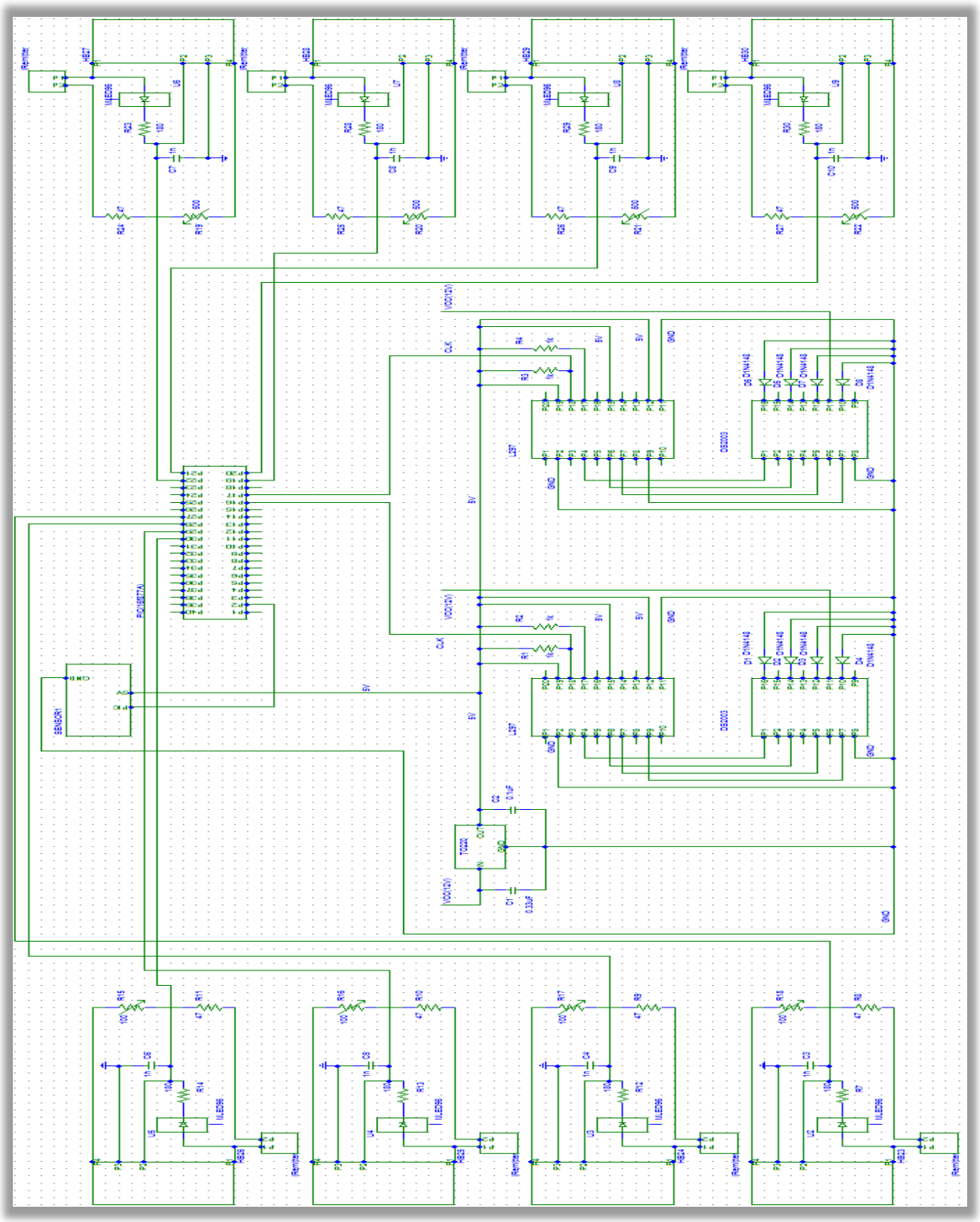


Figure-2.2.1: Final Schematic of the mouse

3- Hardware Diagnostics: Power Electronics and Control Systems

3.1- Motor Driver Circuitry

Using the Y129-5, we had the added benefits of precision movement, equality between similar models and they are brushless. To effectively turn these motors a driver circuit is need to control the motors steps (increments), the steps are cause by the phase changes of the internal wiring of the motor, as each phase is powered it caused the shaft to rotate. The motor circuit has three (3) main sections, pattern(wave) generator, power generator and the actual motors.

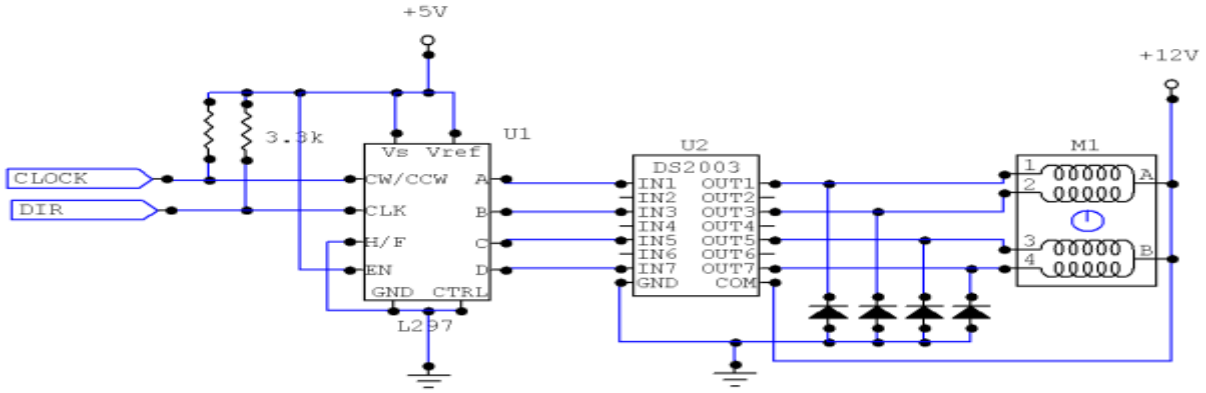


Figure 3.1.1 Motor driver Circuit[3]

- Pattern Generator – using the L297, Stepper Motor controller we supplied a single square wave to the IC's Clock pin and produced a configuration that can drive four outputs, in this case our phases of the stepper motor. The stepper motor can be set to Full/Half Step were each step angle can be bigger or smaller depending on the orientation of the Half/Full Pin, smaller steps can lead to better precision for turning or general movement.
  We can also control the direction of using the controller, simply supplying 5 volts or grounding the clock/clockwise pin can control direction of the steps.
- Power generator – using a DS2003 chip we ensure the motor receives a strong/constant current to sustain continuous movement, since the DS2003 is connected to pattern generator the power generated is in the pattern of the input. The DS2003 uses Darlington complimentary transistors to maintain the high power requirements, also implemented are four diodes to prevent backwards current.
- Motors – Y129, Unipolar stepper motor, using two of these motors movement of the mouse can be achieved. Each motor has its own dedicated circuit i.e. L297 and DS2003. The motors have max torque stationary hence we needed to ensure we make the most of the initial moment when the shaft moves by attaching wheels with superb traction and little weight. Each motor requires a supply of 12volts which was supplied directly.

Since different parts of the circuit required different voltages a DC-DC circuit was needed to bring down the 12 volts needed for the LS2003 and the Motors to 5 volts needed for the L297.

3.2- Sensor Circuitry Development &Testing

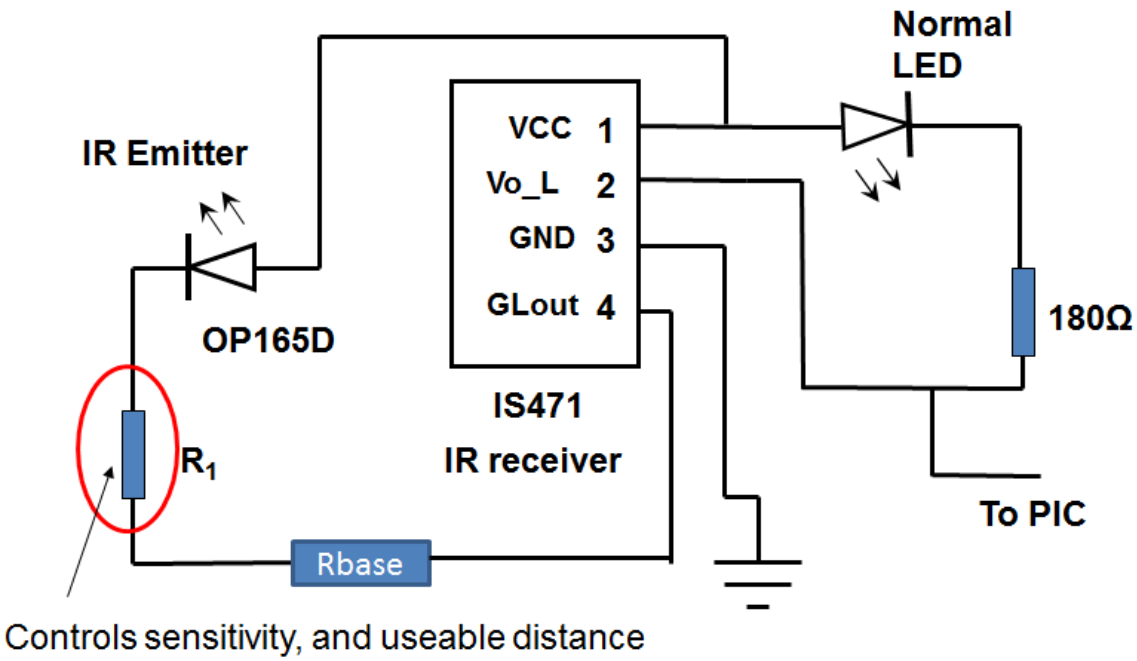


Figure 3.2.1 Sensor Circuit for receiver and emitter [4]

The IS471(receiver) and Op165d(emitter)work using 5v supplied to the first pin of the receiver and also the anode of the emitter, the emitter is connected in series to the 500 ohm variable resistor to control the emission range, then to pin 4 of the receiver. For diagnostics we added LEDs to the circuit to see real time detection during testing. The LED is connected between pin 1 a 180 Ohm resistor and pin 2 of the receiver, pin 2 is also used with the pic, in the software we test whether this pin is high, meaning no detection or low meaning the receivers have detected a wall. Pin 3 is grounded.

Each set of circuit, 3 in total has a specific job and needs to be placed in a specific location to be effective. Below is a breakdown of the entire sensor circuit divided into the 3 main parts.

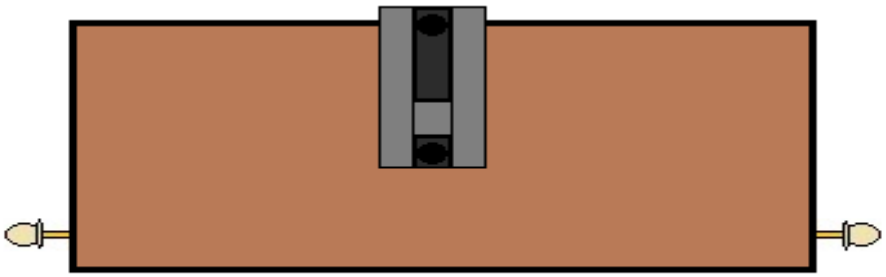


Figure 3.2.1 Forward distance and proximity/collision detector sensors

The front sensors main job is to avoid collision both with side walls and any wall that might be in front of the mouse. We adjusted the Proximity sensors to detect from a range of 1.5cm, with a maze of 17cm and wheel to wheel width of 14cm the mouse has a zone of around 3cm when it's going straight or in a wondering zone during testing we realised we

needed extra sensors for precision to get rid of the wondering zone. Although for just navigating the maze the forward distance and proximity sensor worked fine if we wanted to accurately map maze we needed more precision. This was the moment that we decided to add more sensors.

- Extra Sensors for precision centring

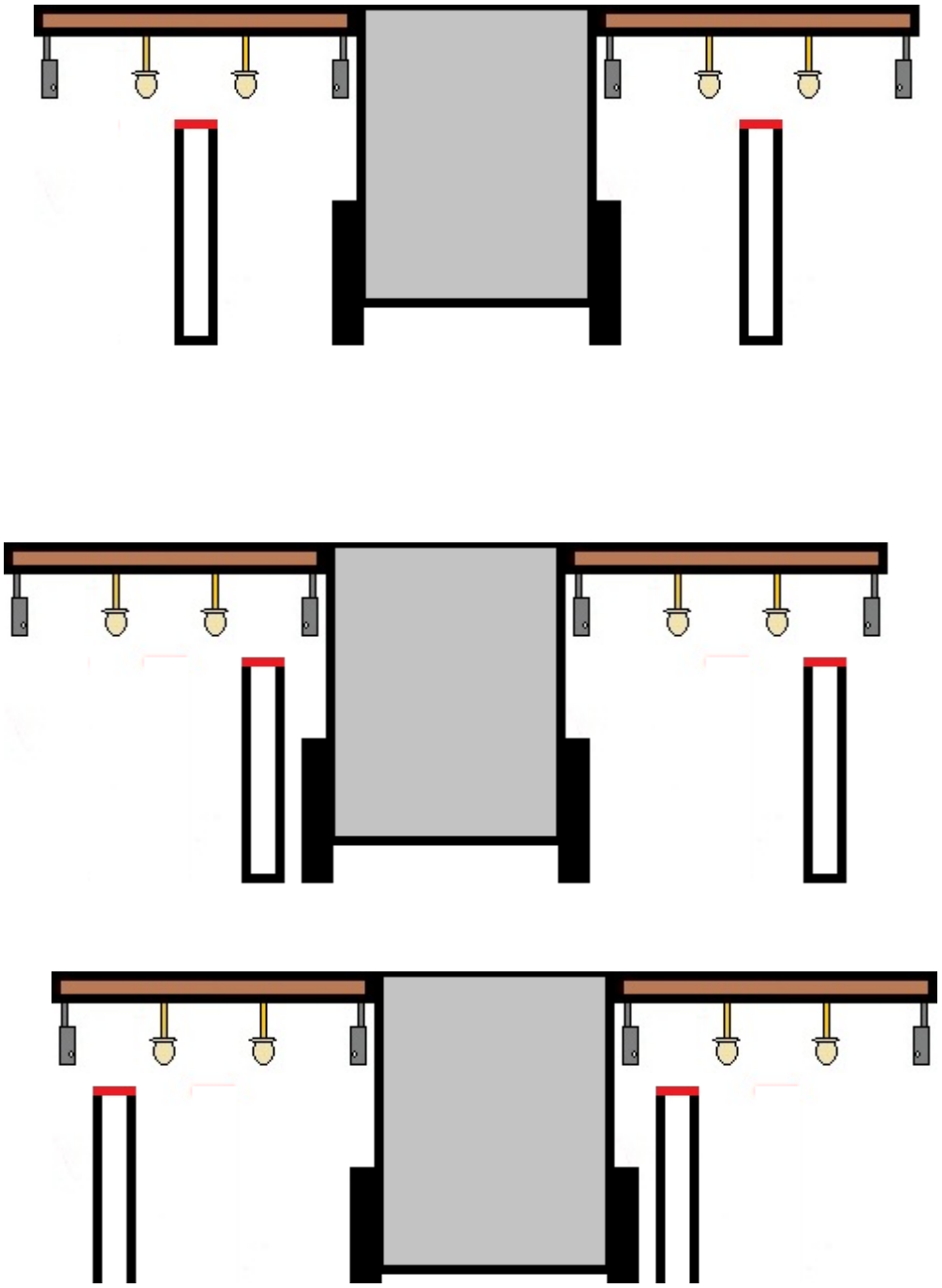


Figure 3.2.2 image of wing sensor in various positions, target, left correction needed, and right correction needed

We added these sensors to create better precision for navigation, the sensors work by correcting left if ever L2, the left outer sensor and R1, the right most inner sensor were ever on and correction right if L1, left inner and R2 right outer sensors are ever on, i.e. detect something. In tests we have seen that these extra sensors help to reduce the wondering zone greatly. We could not however remove the other proximity sensors in fear of these wing sensors ever getting into a situation where they are not over a wall. Hence these sensors work with the front proximity for greater precision in avoiding collisions.

- Wall detector

Figure 3.2.3 *image back sensor, front view, Back Left1(BL1), Back Right1(BR1)*

These sensors have the job of wall locator, the mouse cannot navigate solely on proximity, and this would be time consuming. We can tell if a wall is present if either BL1 or BR1 have zero volts i.e. the LED light up. The sensors were placed at the back to avoid interference between other emitters.

4-Software Integration

4.1- Software Implementation of the Motor

The motors act as the legs of our mouse, and are used for the following applications:

1. Mobility (so the mouse can traverse various parts of the maze).
2. Centring (ensures mouse traverses accurately within the confinement of the walls)
3. Avoid Collisions (to ensure mouse does not take any damage by unnecessary collisions)
4. Turns-(to ensure that the mouse can complete accurate 90 degree turns)

The software implementation will be divided for the above four cases:

Mobility

The motors being unipolar stepper motors, require a square wave of about 50-800Hz to operate. Due to the our mouse having a bulky chassis, and keeping the sensor delay time under consideration our mouse optimally performed at a square wave frequency of 220Hz.

At this speed we ensured that the mouse did not traverse too much distance in a given frame of time, providing ideal time for the mouse to read the correct sensor values and correct itself if necessary.

Timer0 and Timer1 calculations

Timer0 controls the actual period of the generated square wave and timer1 controls the duty cycle of the square wave.

The pre-scalar of Timer0 was assigned to 32. It was initialised with a value of 113, so that in total there will be

255-113=142 cycles between time-outs.

This makes the total number of cycles be:

```
timer_setup
    bcf     INTCON,TOIF ;clear timer overflow flag
                        ;always do this to reset the timer
    bsf     STATUS,RP0  ;set page to bank1
    movlw   b'11010100'
    movwf   OPTION_REG  ;set timer to be clocked by fosc/4
                        ;set prescalar to 2:1
                        ;1XXXXXXX=pull-ups disabled
                        ;x1xxxxxx=interrupt on rising edge
                        ;xx0xxxxx=timer source clock internal
                        ;xxxx0xxx=prescalar assigned to timer
                        ;xxxxx100=32:1
    bcf     STATUS,RP0  ;set page 0
    movlw   d'113'      ;255-113=142 cycles between time-outs approx
                        ;therefor the period of the waveform becomes
                        ;142*32=4545uS or 4.545msmS which is 220Hz
    movwf   TMR0        ;initialise timer value
    bsf     INTCON,TOIE ;enable timer overflow interrupt
```

142*32=4545uS or 4.545ms, which is 220Hz.

```
;############    INCLUSIVE TIMER1 SETUP
    movlw   b'00110100'
    movwf   T1CON       ;00XX XXXX=we dont care what values they hold
                        ;XX11 XXXX=Timer1 Prescaler 11=1:8(ideal)
                        ;XXXX 0XXX=oscillator is disabled(internal RC)
                        ;XXXX X1XX=Do not Synchronize clock external input
                        ;XXXX XX0X=Timer1 Clock Source Select Bit=>Use internal Clock
                        ;XXXX XXX0=Timer1 ON BIT
    bsf     T1CON,0     ;TURN timer1 on
    bcf     INTCON,TOIE ;for now, turn timer0 off
```

The pre-scalar of Timer1 has been set to 8. This means that timer1 will count after skipping every 8 instruction cycles.

```
clrf    TMR1L
clrf    TMR1H
movlw   b'00011100' ;this makes 284*8=2272(half period)
movwf   CCPR2L      ;move this value into both registers for CCP module
movwf   CCPR1L
movlw   b'00000001'
movwf   CCPR2H      ;set the higher registers
movwf   CCPR1H
```

The Timer1 splits the period of Timer0 by a half.

The way the output signal is set high or low, is achieved by CCP module, and we use the Compare module on the CCP to ensure we have a high or low on the CCP1 and CCP2 pins. We upload a counter value in the CCP registers so that once there is a match between the Timer1L and Timer1H registers and the CCP registers, the CCP pin will go high or low, depending on what we have configured the CCP pin to be.

FORWARD MOVEMENT: This is achieved by setting the direction pins opposite each other i.e. pin RC3 is low and RC4 is high. And motors are pulsed at 220Hz.

Centring: Centring is achieved with the feedback from the wing sensors.

```
left_wall_centering

    lft_crct_setter d'2'
    bcf     PORTC,RC3
    bsf     PORTC,RC4

    goto    turn_set

right_wall_centering

    rt_crct_setter  d'2'

    bcf     PORTC,RC3
    bsf     PORTC,RC4

    goto    turn_set
```

We stop one of the motors, and pulse the other motor to align the mouse in such a way that it becomes equidistant from both the walls. We apply a maximum of two pulses as this ensures that the mouse corrects itself by a distance of 5-7mm.

Left Centring: Left motor is pulsed and right motor is stopped. Two pulses are applied

Right Centring: Right motor is pulsed and left motor is stopped. Two pulses are applied.

Avoid Collisions: We check the feedback from the front proximity sensors.

It is similar to centring, except the correction of the mouse requires more severe alignment. We apply maximum of 6 pulses, so that mouse corrects itself by a distance of 1-1.5cm.

```
left_collision_correct
    lft_crct_setter d'6'
    bcf     PORTC,RC3
    bsf     PORTC,RC4

    goto    turn_set

right_collision_correct
    rt_crct_setter d'6'

    bcf     PORTC,RC3
    bsf     PORTC,RC4

    goto    turn_set
```

Turns:

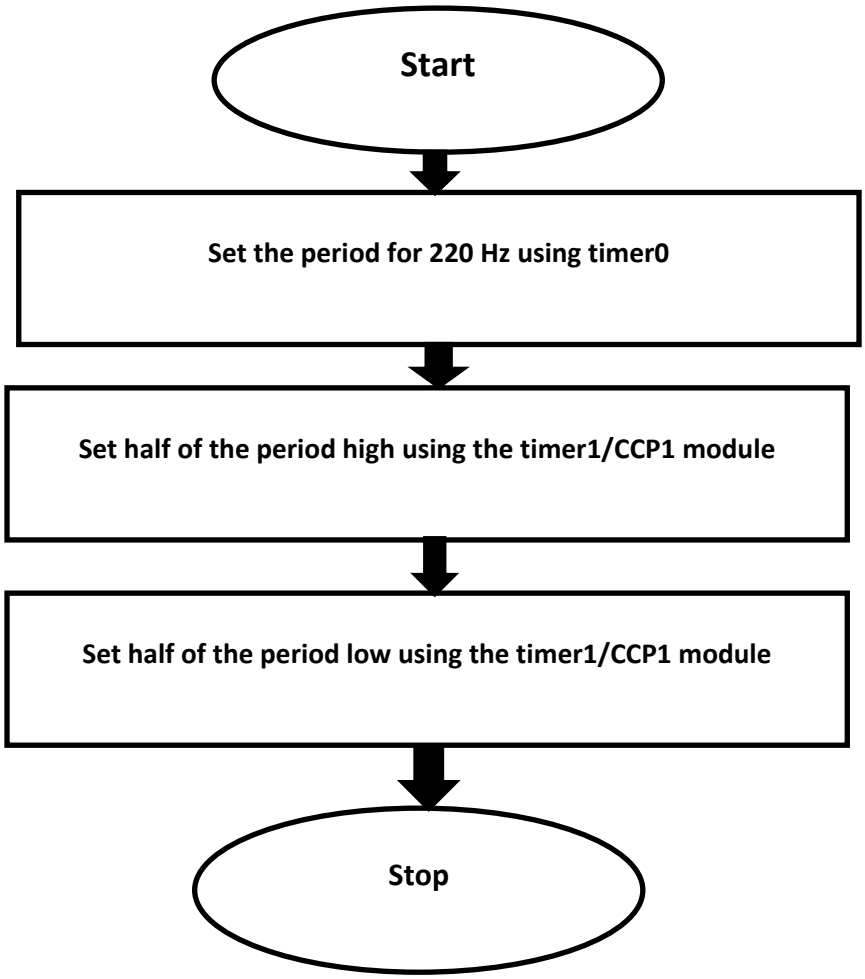
This is achieved by making the mouse turn clockwise for a right turn and anticlockwise for a left turn. We pulse the motor a fixed number of times, to ensure that the mouse rotates at a full 90

Left Turn: We pulse the motor 153 times and set RC3 and RC4 high for an anticlockwise turn.

Right Turn: We pulse the motor 153 times and set RC3 and RC4 low for an clockwise turn.

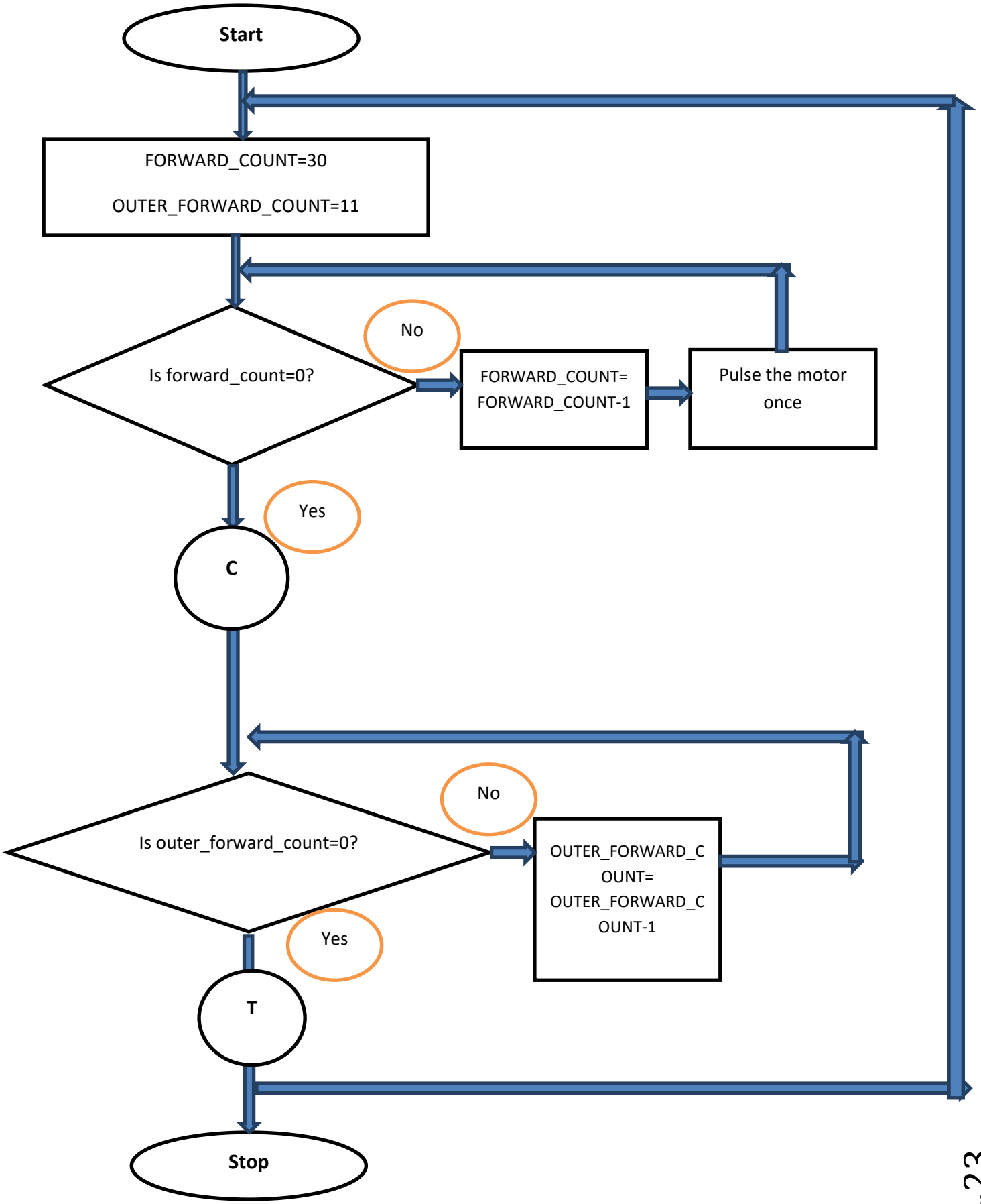
The pulsing of the motor is done in the interrupt service routine (ISR). The ISR represents one single pulse of the motor. Therefore, the number of times the motor is pulsed is the number of times we send the program into interrupt mode. This is done by setting up an infinite loop where every time the program goes into interrupt mode a counter is decremented (where the counter represents the number of pulses) and when returned from the interrupt mode the counter value is check if the motor is pulsed the required number of times. This was we can control the number from the main routine

The flowchart below shows the interrupt service routine of the part of the program that controls the motors.

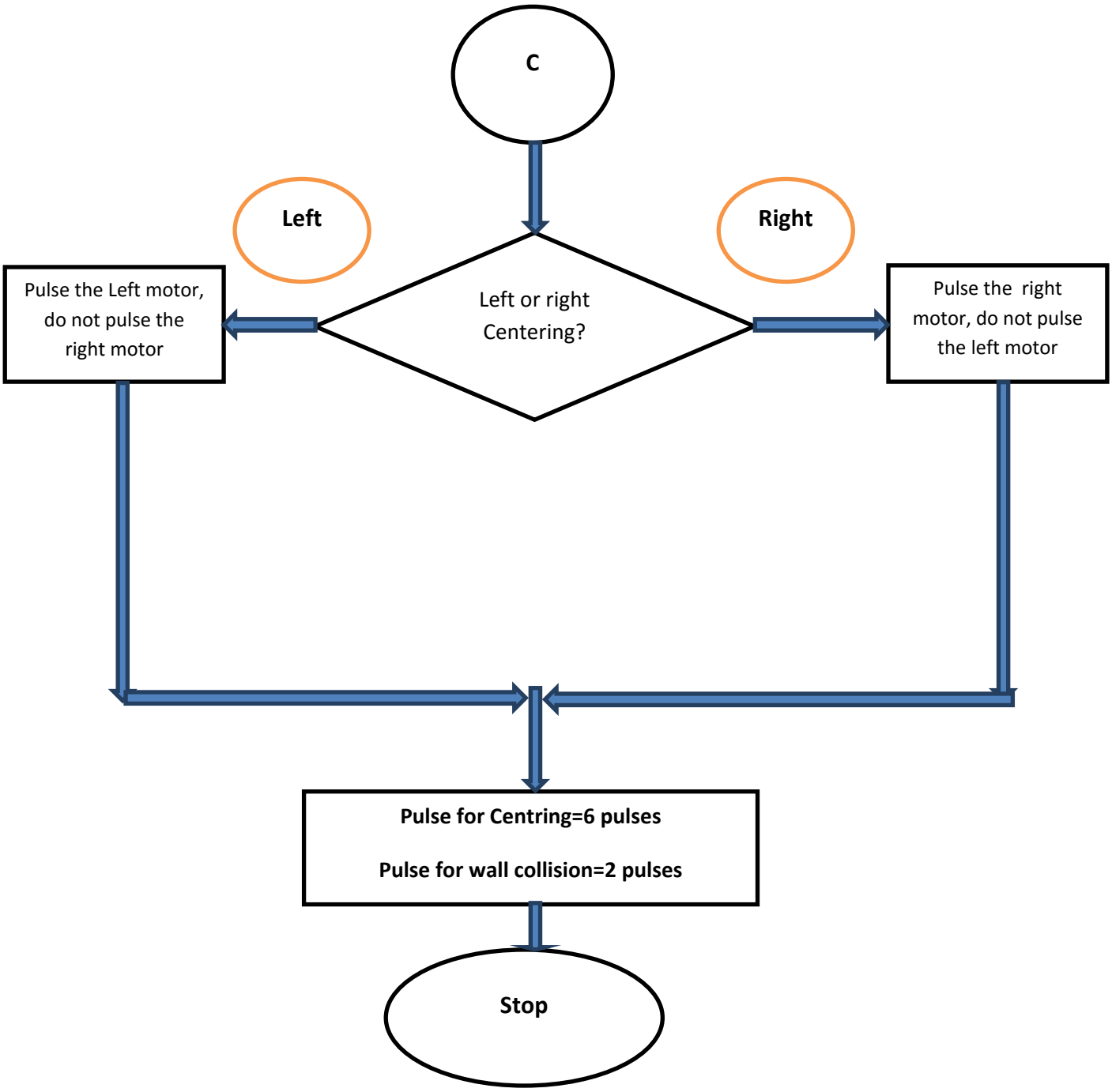


Flowchart-1: Flowchart of the Interrupt Service Routine

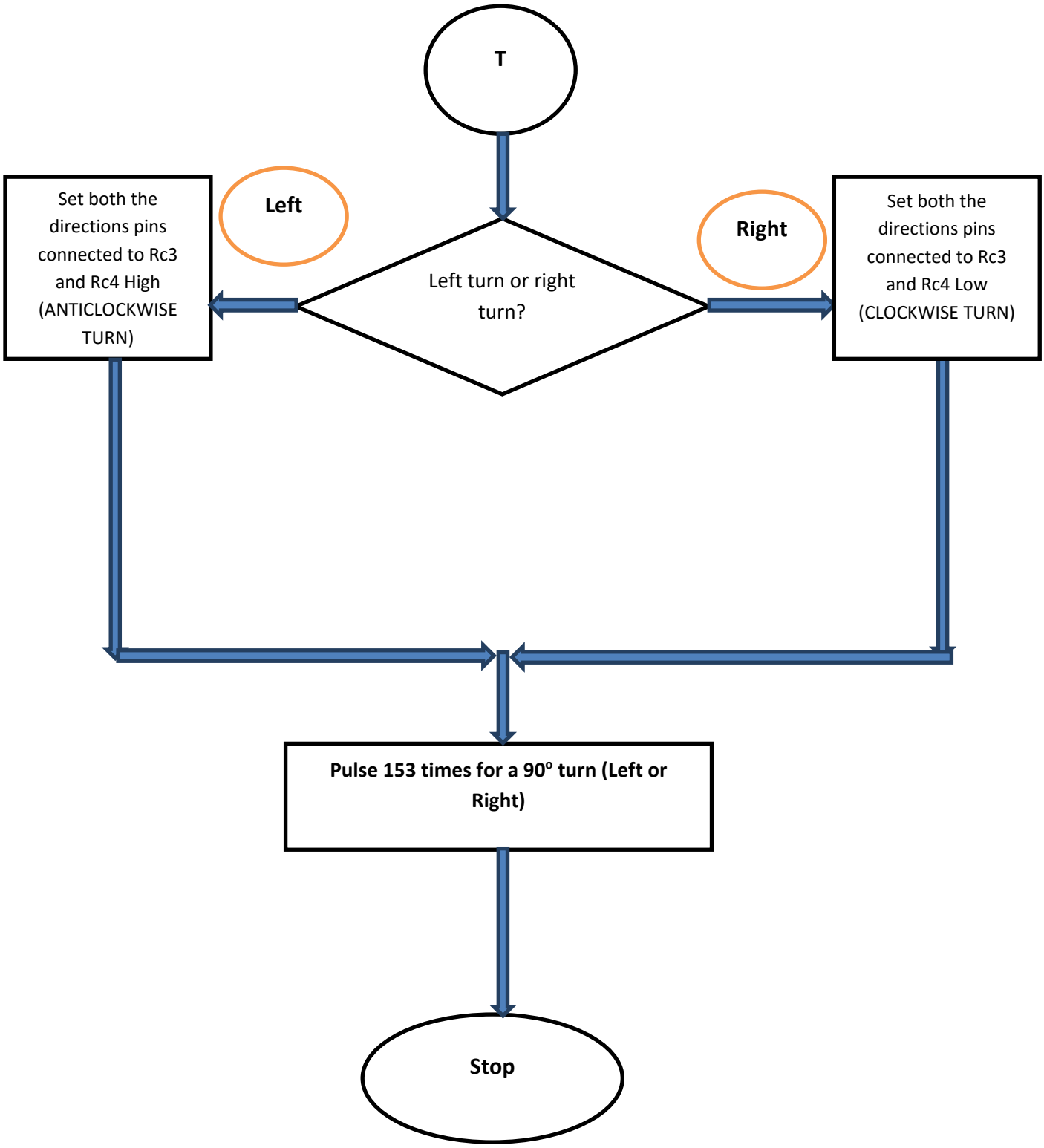
The next flowchart explains the overall control of the motors. The value 30 is loaded into the forward_count_ variable which represents the number of pulses. And the variable outer_forward_count represents number of times we repeat the '30 times' pulsing of the motor. This variable is loaded with a value of 11. Therefor the motor is pulsed for 30*11=330 pulses which is the number of pulses required for each square and the flowchart is repeated. Therefore for every square this flowchart is executed. The centring and turning parts of the flowchart are done in separate flowcharts.


Start
FORWARD_COUNT=30
OUTER_FORWARD_COUNT=11
Is forward_count=0?
No
FORWARD_COUNT= FORWARD_COUNT-1
Pulse the motor once
Yes
C
Is outer_forward_count=0?
No
OUTER_FORWARD_COUNT= OUTER_FORWARD_COUNT-1
Yes
T
Stop

Flowchart-2:Flowchart explaining the overall control of the motors(previous page)



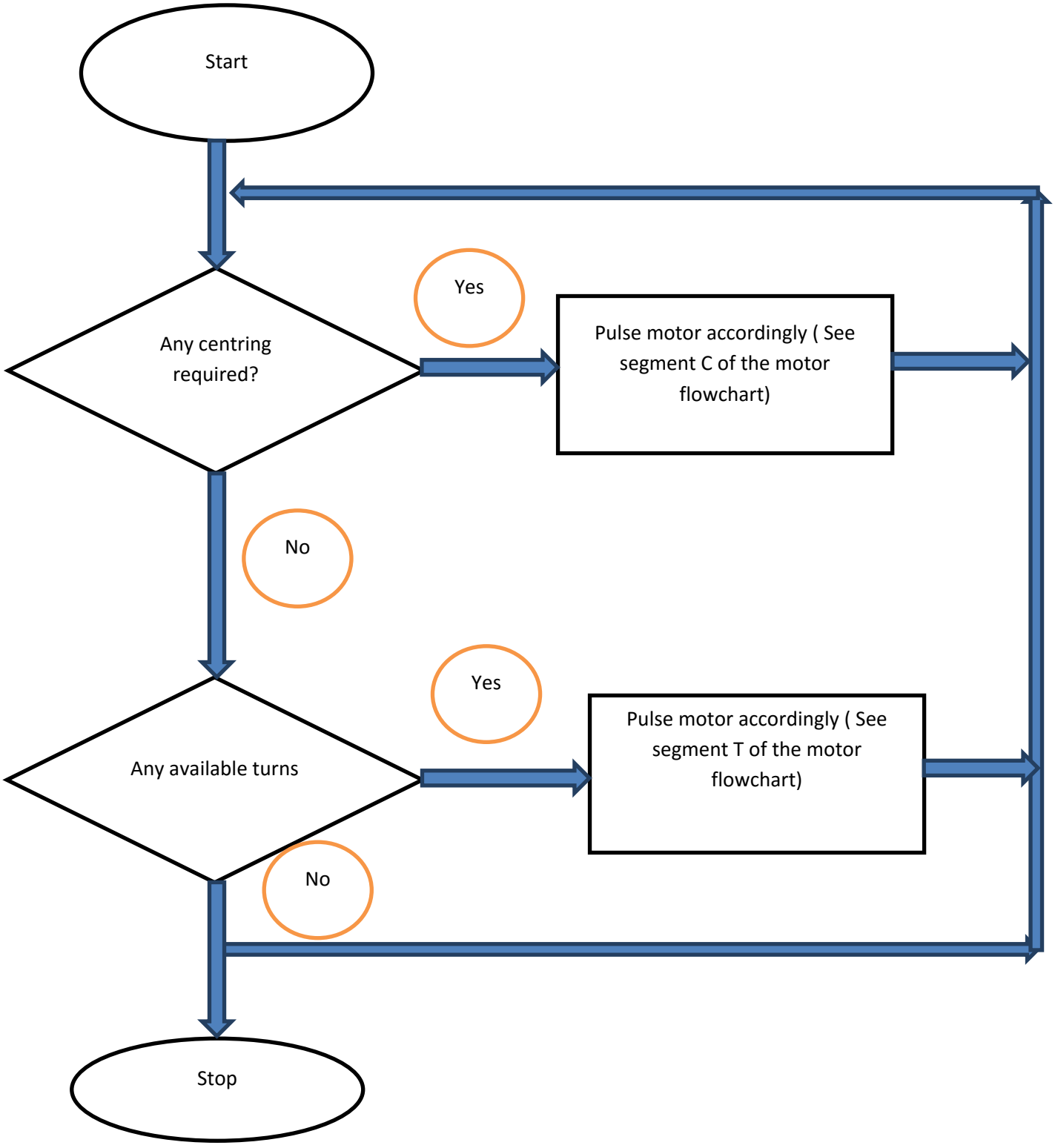
Flowchart-3:Flowchart explaining the pulsing of the motors when performing centring(above)



Flowchart-4:Flowchart explaining the pulsing of the motors when turning

4.2- Software Implementation of the Sensors

At the software level, the proximity sensors were pulsed periodically. The pulsing of the sensors was done in sequence, i.e., the pulsing of the sensors was done in a loop where each set of sensors were pulsed at different. The flowchart explains the sequence of steps that were followed when pulsing the sensors.



Flowchart-5: Sensor check flowchart

The reason why the sensors are checked the way shown above because the way used above gives us a way to minimising the number of errors when navigating through the maze. The sensor check can be explained the following way:

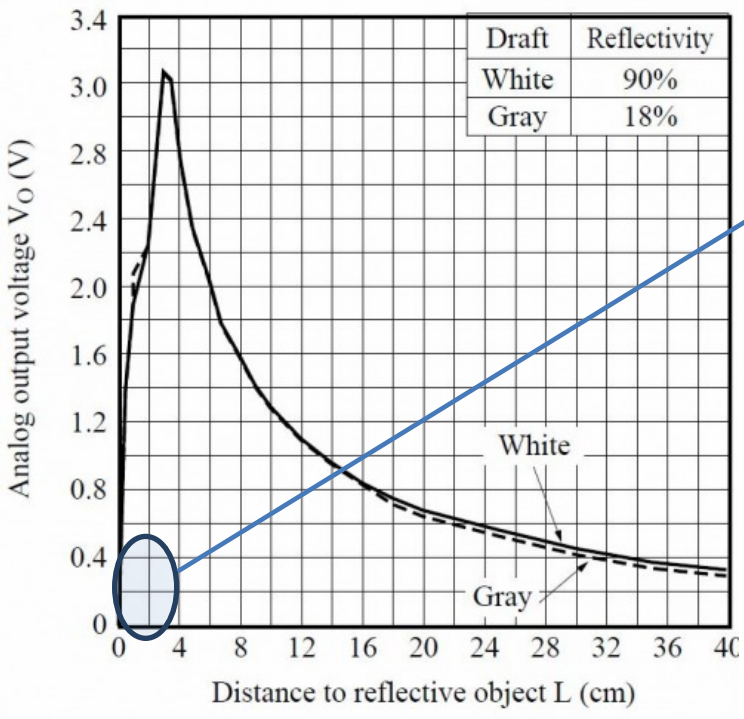
1. **Centring:** The centring of the mouse is performed by two sets of sensors
   - i- **Collision sensors:** The collision sensors are prioritised the most because collision with any walls throws the mouse off by a few steps which results in the mouse crashing and going off track frequently.
   - ii- **Wing Sensors:** The wing sensors work hand in hand with collision sensors. The information provided by the sensors is used to make the mouse exactly at the centre of the square when traversing i.e. these are used for the fine tuning of the centring provided by the collision sensors.
2. **Turning:** The check for turning is performed for every square. Therefore the information for the turning is taken in the end because this check is performed at the end of each square.

```
;--------------------------------------------------------------------------------
;               CENTERING
;--------------------------------------------------------------------------------

check_centering_sensors
;################ Checking for possible collisions###################

    movf    PORTD,0
    andlw   b'00000110'         ;mask collision sensors
    movwf   PORTD_TEMP          ;store in temporary variable

    movlw   b'00000100'         ;mouse is deviated towards left(about to collide)
    xorwf   PORTD_TEMP,0
    BTFSC   STATUS,Z
    goto    left_collision_correct


    movlw   b'00000010'         ;mouse has deviated towards right(about to collide
    xorwf   PORTD_TEMP,0
    BTFSC   STATUS,Z
    goto    right_collision_correct

;;############### check for right wing ############################

    movf    PORTD,0
    andlw   b'00110000'         ;mask sensors for right wing
    movwf   PORTD_TEMP          ;store in temporary variable

    movlw   b'00100000'         ;mouse needs slight adjustment on right
    xorwf   PORTD_TEMP,0
    BTFSC   STATUS,Z
    goto    right_wall_centering

    movlw   b'00010000'         ;mouse needs slight adjustment on left side
    xorwf   PORTD_TEMP,0
    BTFSC   STATUS,Z
    goto    left_wall_centering
;################## check for left wing ########################

    movf    PORTD,0
    andlw   b'11000000'     ;mask sensors for left wing
    movwf   PORTD_TEMP      ;store in temporary variable

    movlw   b'10000000'     ;mouse needs slight adjustment on right
    xorwf   PORTD_TEMP,0
    BTFSC   STATUS,Z
    goto    right_wall_centering

    movlw   b'01000000'     ;mouse needs slight adjustment on left
    xorwf   PORTD_TEMP,0
    BTFSC   STATUS,Z
```

```
;-------------------------------------------------------------------------------
;           SENSOR CHECKS
;ZERO SIGNIFIES PRESENCE OF WALL
;ONE SIGNIFIES ABSENCE OF WALL
;-------------------------------------------------------------------------------
sensor_check

    movlw   d'11'
    movwf   LEFT_COUNT              ;reset left check counter, so that it restarts per square checl

    movf    PORTD,0
    andlw   b'00001001'             ;sensor mask for wall detection checks
    movwf   PORTD_TEMP              ;store readings in temporary register


    movlw   b'00001000' ;left wall is absent AND RIGHT WALL IS PRESENT therefore turn left
    xorwf   PORTD_TEMP,0
    BTFSC   STATUS,Z
    goto    left_turn


    movlw   b'00000001' ;Right wall is absent AND LEFT WALL IS ABSENT therefore turn right
    xorwf   PORTD_TEMP,0
    BTFSC   STATUS,Z
    goto    right_turn

;*******************************************************************************
;U-TURN
;***********************************************

    movlw   b'00000000' ;if there are walls present on both sides then turn left, and will turn left again
                        ;to complete u turn
    xorwf   PORTD_TEMP,0
    btfsc   STATUS,Z
    goto    left_turn


    goto    sensor_check ;keep checking sensors




  ;-------------------------------------------------------------------------------

  left_check
      movlw   d'11'             ;reset the LEFT checking counter
      movwf   LEFT_COUNT

      movf    PORTD,0
      andlw   b'00001001'       ;masking for wall detection sensors
      movwf   PORTD_TEMP        ;temporary register

      movlw   b'00001000' ;left wall is absent AND RIGHT WALL IS PRESENT therefore turn left
      xorwf   PORTD_TEMP,0
      BTFSC   STATUS,Z
      goto    left_turn

      movlw   b'00001001'       ;walls are absent on both sides of the maze, so turn left (priority
      xorwf   PORTD_TEMP,0
      BTFSC   STATUS,Z
      goto    left_turn

      goto    read_distance     ;no turns available, so continue moving forward for 330 pulses
```

Software Implementation of the Distance sensor:

The distance sensor works on the principle that the output voltage is inversely proportional to the distance detected by it. That can be clearly observed from the following graph:



This point here marks the Dead Zone of the distance sensor (one of the flaws it has)

Figure-4.2.1: Graph showing the working of a distance sensor

Our approach was based on using the Output voltage values that the distance sensor gives us in such a way that we could establish a relation between the voltages and distance. We simply couldn't write an equation saying that this voltage comes out to be this distance. Thus, we measured each voltage and checked the corresponding distance.

However the voltage given by the distance sensor is in the analogue form, therefore we had to make

```
;*********************************************************************************
;   ANALOG TO DIGITAL PROCEDURE
;*********************************************************************************
;INIT A/D MODULE
;CONFIGURE PIC I/O LINES
;2. SELECT PARTS TO BE USED BY SETTING THE PCFGX BITS IN ADCON1 SELECT ADFM AS WELL
;3. SLEECT ANALOG CHANNELS A/D CONVERSION CLOCK AND ENABLE A/D MODULE
;4. WAIT ACQUISITION TIME
;5. INITIATE ONVERSION BY SETTING GO/DONE BIT IN ADCON0
;6. WAIT FOR CONVERSION TO COMPLETE
;7. READ AND STORE DIGITAL VALUE

Init_A2D
    BANKSEL TRISA
    MOVLW   b'00000001' ;RA0 AS INPUT
    MOVWF   TRISA
    MOVLW   b'00001110'
    MOVWF   ADCON1      ;ADFM=0 HENCE LEFT JUSTIFIED
                        ;0011-RA0 AS ANALOG AND RA3 IS VREF+ REST ARE DIGITAL
                        ;SET ADCON0 THE MAIN ADC CONVERTOR ROUTINE
    BANKSEL ADCON0
    MOVLW   b'01000000' ;FOSC/8
                        ;CHANNELS AS 000=RA0
                        ;ADON=1 HENCE TURN ADC MODULE ON
    MOVWF   ADCON0

    BCF     INTCON,ADIE
    BSF     ADCON0,ADON
    CALL    DELAY_20uS

    RETURN
```

use of the ADC in the PIC to convert the analogue waveform into a digital value.

The Subroutine Init_A2D is responsible for setting up the ADC format. Details have been provided in the snippet of the code above.

```
;*****************************************************************************
;   READ A/D LINE
;*****************************************************************************
;READ VALUE AND CONVERT TO DIGITAL, MAIN ADC ROUTINE FOR CONVERSION

Read_A2D
    CALL    DELAY_20uS
    BANKSEL ADCON0
    BSF     ADCON0,GO   ;SET GO BIT TO START ADC CONVERSION...
                        ;AUTOMATICALLY GETS CLEARED WHEN CONVERSION IS COMLPLETE

convWAIT
    BTFSC   ADCON0,GO
    GOTO    convWAIT
;READ MSBs
    MOVF    ADRESH,0
    BANKSEL PORTB
    RETURN
```

The subroutine Read_A2D is what actually converts the analogue value into a digital value (binary to be specific). Since the PIC is incapable of using information that's analogue, it becomes quite essential to convert something that may be analogue to a digital value.

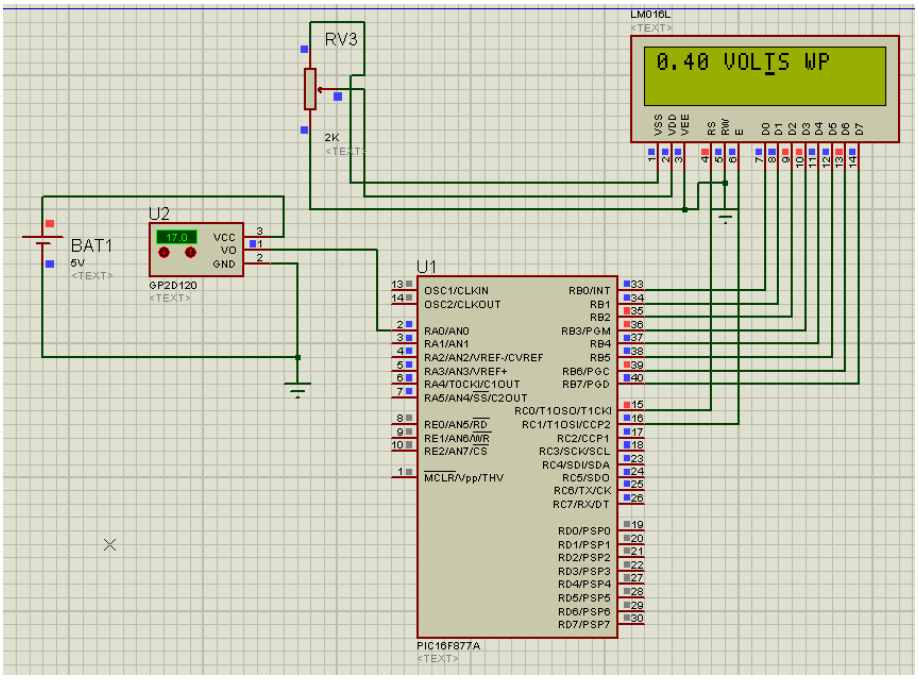
However, this binary value is pretty arbitrary as we have no idea as to what the binary value may stand for. So conversion of this binary value was done to the ASCII character set, to ensure that the values we have can be easily used. The conversion divides the voltage by two and gives a voltage output in the form of decimal values. Thus, for an input voltage of 5.00V the ASCII converter will give the reading to be 2.50V. Therefore we had created a digital Voltmeter, for the distance sensor. The Binary to ASCII converter used an algorithm, and results of the conversions (decimal places), were stored in different registers (ones, tenths and hundredths), to ease debugging. Below is the snippet of the conversion algorithm.Conversion is done by the bin_2ASC subroutine.

```
;*****************************************************************
;   BINARY TO ASCII VALUE DECIMAL CONVERSION
;*****************************************************************
 ;ON ENTRY:
;   W HAS BINARY VALUE IN RANGE 0 TO 255
;ON EXIT:
;   OUTPUT VARIABLE asc100, asc10 AND asc1 HAVE THREE DECIMAL UNITS
;VALUE 100 IS SUBTRACTED FROM SOURCE OPERAND UNTIL 0 REMAINS
;NUMBER OF SUBTRACTIONS IS THE DECIMAL HUNDREDS RESULT.
;100 IS ADDED TO COMPENSATE FOR LAST SUBTRACTION
;10 IS SUBTRACTED IN SAME MANNER
; VARIABLES
;inNUM  -STORAGE SPACE FOR OPERAND
;asc100
;asc10
;asc1
;thisDIG
bin_2ASC

    MOVWF   inNUM           ;COPY OF SOURCE VALUE
    CLRF    asc100
    CLRF    asc10
    CLRF    asc1
    CLRF    thisDIG
sub100
    MOVLW   d'100'
    SUBWF   inNUM,1
    BTFSC   STATUS,C        ;DID SUBTRACT OVERFLOW
    GOTO    bump100         ;NO COUNT OVERFLOW
    GOTO    end100
bump100
    INCF    thisDIG,1
    GOTO    sub100
;STORE 100TH DIGIT
end100
    MOVF    thisDIG,0       ;DIGIT COUNTER
    ADDLW   0x30            ;CONVERT TO ASCII
    MOVWF   asc100          ;STORE THE VALUE
```

```
;CALCULATE THE 10S POSITION
    CLRF    thisDIG
;adjust value
    MOVLW   d'100'
    ADDWF   inNUM,1
sub10
    MOVLW   d'10'
    SUBWF   inNUM,1         ;SUBTRACT 10
    BTFSC   STATUS,C        ;CHECK FOR OVERFLOW
    GOTO    bump10
    GOTO    end10
bump10
    INCF    thisDIG,1       ;INCREMENT DIGIT COUNTER
    GOTO    sub10
end10
    MOVLW   d'10'
    ADDWF   inNUM,1         ;ADJUST FOR LAST SUBTRACTION
    MOVF    thisDIG,0
    ADDLW   0x30            ;CONVERT TO ASCII
    MOVWF   asc10           ;STORE THE DIGITAL VALUE
;CALCULATE AND STORE UNITS DIGIT
    MOVF    inNUM,0         ;STORE UNITS VALUE
    ADDLW   0x30
    MOVWF   asc1
    RETURN
```

Thus we have done the binary to decimal value conversion.

We debugged the distance and voltage values using Proteus to measure the voltage values given for each particular distance. We deduced that between the voltage ranges of 1.25V and 0.4V, the mouse measures distances less than 17cm.



We had to program an LCD to measure the output voltages and to verify if the simulation results corresponded to the real life results.

Thus to observe the output voltage of the distance sensor, we monitored the changes in the LCD and measured the corresponding distance.

Figure-4.2.2 Proteus schematic for debugging the distance sensor

Countering the Dead Zone:

We noticed that the distance sensor values would fluctuate for distances less than 4cm. To avoid such fluctuations the dead zone was “disabled” or all voltages corresponding to distances less than 16cm implied

that the mouse had a wall right in front of it. Thus if the mouse had a wall in front of it at a distance of 12cm or 6cm, it meant the same thing i.e. that there was a wall in front of it.

Because we had disabled measurements of distances less than 16cm, we had to make sure that the mouse picked up an oncoming wall 17cm (1 square in advance). We used the voltage value of 0.5V (measured) to specify that the mouse had detected a wall in front of it, one square in advance. After the mouse had detected, we made sure, that the mouse reached the centre of the square in front to ensure precise turning was possible.

To summarise, we used the distance sensor to check how far an oncoming wall is and after wall detection not measuring the distance, but only checking if wall is in front or not.

CODE IMPLEMENTATION

And finally if-else statements were used to check the values received by the distance sensor.

The following table summarises our method of debugging the received distance sensor values

Output Voltage (as displayed on LCD)	Measured distance(s)	Command to mouse
1.25 V to 1.00V	Less than 6 cm	Don't move, wall is at front
0.9V to 0.5V	For 0.9v =7cm to 0.6v=12cm-14cm(varies)	Don't move, wall at front
0.5V	14cm to 16cm(varies)	Wall detected a square ahead
0.4v to 0V	From 17cm to large distances	No oncoming walls, keep moving forward

Table-2-Table of values used as reference when coding the distance sensor

We checked the ones place of the decimal value. If there was a 1 we knew measured distances would be less than 6cm. If the ones place had a 0, we would check the tenths place of the decimal value and find the ideal match, as per the table readings. Anything less than 4 at the ones place meant no oncoming walls. Complete code of how this was possible is given in the code snippet below:

```
read_distance
    ;part of the code that gives the forward movement priority
;the code first checks the distance sensor to see if the mouse is able to move.
;Two cases under the distance sensor:

    call    Read_A2D    ;now the w register has the ADRESH value, retrieves the Voltage
    call    bin_2ASC    ;converts the analog value to an ascii value, thus voltage in ASCII format

    decfsz  LEFT_COUNT,1    ;decrement it 11 times for periodic checks (per square)
    goto    left_check      ;check for availability of left turn

distance
;Here we interpret the voltage values from 1.25V to 0.40V
;   1.25 V to 0.40 V implies distance ranging from 4 cm to 15 cm
;   0.5V corresponds to exact 17cm distance
;checking is done systematically to check every number before and after decimal point.
;priority is given to see if front collision is possible (tackling the dead zone),
; If no values match, then that means
; that no oncoming obstacles are present and mouse continues to move forward
```

Continued…

```
movlw   '0'         ;is voltage value 1.00??
xorwf   asc100,0
btfss   STATUS,Z
goto    dont_move   ;if yes, goto suspend movement, if no, check for value after
                    ;decimal point

movlw   '9'         ;is voltage value 0.9V?
xorwf   asc10,0
BTFSC   STATUS,Z
goto    dont_move   ;if yes, suspend movement, otheriwise check next value

movlw   '8'         ;is voltage value 0.8V?
xorwf   asc10,0
BTFSC   STATUS,Z
goto    dont_move   ;if yes suspend movement, otherwise check next value

movlw   '7'         ;is voltage value 0.7V?
xorwf   asc10,0
BTFSC   STATUS,Z
goto    dont_move   ;if yes, suspend movement, otherwise check next value

movlw   '6'         ;is voltage value 0.6V?
xorwf   asc10,0
BTFSC   STATUS,Z
goto    dont_move   ;if yes, suspend movement, otherwise check next value

movlw   '5'         ;is voltage value 0.5V?
xorwf   asc10,0
BTFSC   STATUS,Z
goto    wall_detected   ;if yes, wall has been detected. goto subroutine
                        ;if no, proceed down to "forward"
```

5-Testing

The following tools have been used to simulate the mouse

1. Testing began in steps as the circuits were being built/ and phases completed; general test on every circuit includes visual inspection as well as testing for continuity, ensuring no shorts or broken tracks on the breadboard. General testing and building equipment included Feedback FS600 Function Generator, the Tektronix TDS 210 oscilloscope, GwinStek GSP-18500 DC power supply, soldering iron, pliers, cutters, wire strippers and multi-meter.

2. Proteus: Proteus was used for simulating the distance sensor design. We have connected up the PIC and the LCD in the software the way shown in the figure below.

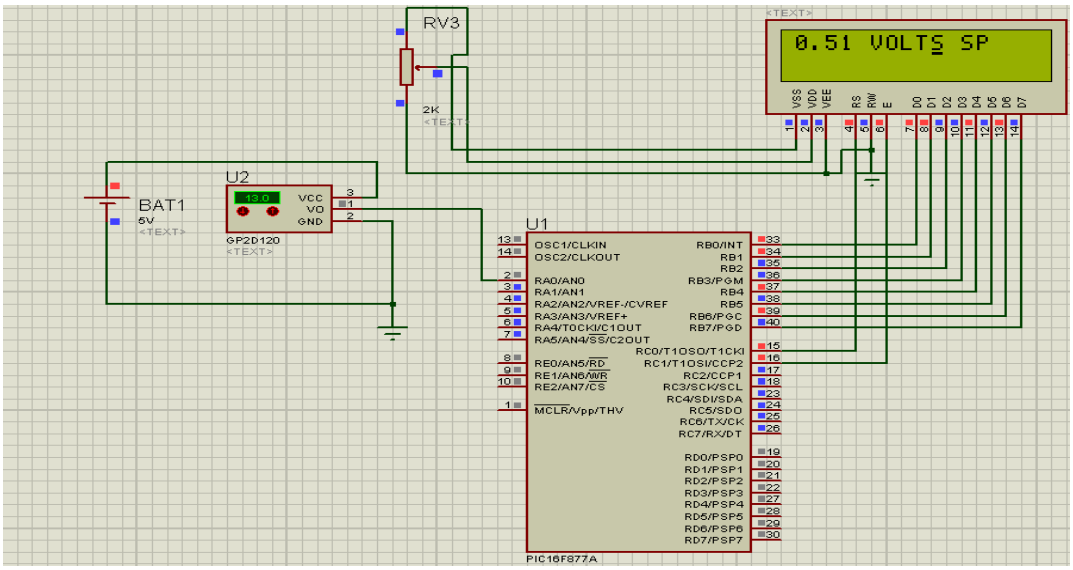


Figure-5.1.1: Simulation Circuit for the distance sensor

We can virtually program the PIC shown in the figure and get an idea of how our software is going to work. We have used Proteaus to simulate all the changes made to the distance sensor design.

Proteaus was also used to simulate the pulses that were used for the motor design. This was done by connecting a virtual oscilloscope to the PIC as shown below

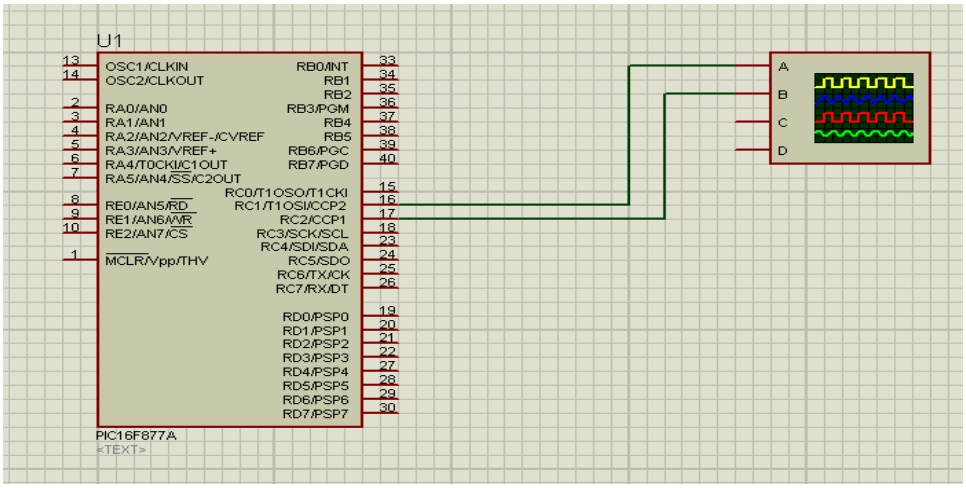


Figure-5.1.2: Simulation circuit for the motor pulses

3. ICD2: ICD-2 in the MPLAB PROGRAMMER environment was used for programming the PIC on the EBLOCK and the testing the actual mouse.

5.2- Tests Performed

Motor Movement –

Starting with the L297(Pattern Generator), we began by supplying a square wave using the signal generator and monitoring the output using the oscilloscope. We began by connecting the DC power supply to the relevant pins supplying five volts and ground where required, then we continued by putting the square wave in the input i.e. pin 18 and monitoring the output, pins 4,6,7 and 9 with the oscilloscope. We expected to see the input recreated at these pins and after some minor adjustments achieved this.

The next step we took was to build the driver circuit and DC to DC converter, we needed to build both of these together as we needed to test the DS2003 with the L297 and it was simpler to build both than connecting dual power supplies and connecting for common ground. After completing the circuits we retested using the same method for the L297 expecting to see the waves at the output on the oscilloscope. Using pins 1, 3, 5 and 7 as input of the DS2003 and using the adjacent pin as output i.e. 16, 14, 12 and 10 respectively. After clarifying that our input was being transmitted effectively and no heating issues were occurring we moved on to the next testing phase, using the actual motor for testing. Testing each motor independently then both at the same time with its significant circuit we began again using the same testing method this time we had to adjust the signal generator for the motor as speeds that are too fast or slow can cause the motor to stall. After calculating the relevant information found the best frequencies for smooth operation with the motors. We began testing other features of the L297 such as direction to ensure we can get both motors to turn same direction as well as opposite. After minor adjustments we achieved stable motor movement with the signal generator.

Mouse Movement –

Using the codes mentioned in the coding sections we began testing the micromouse in motion, after playing around with suitable speeds we settled on 330 pulses at the rate of 220Hz (required number of pulses for one square) the motion of the mouse was further assisted with wheels that added better grip.

In this testing phase we realised the wheels need to perfectly round else a wobble will occur that cause the mouse to veer left or right when moving straight, a vibration would also cause the circuits and sensors to malfunction, we solve this issue by attaching hubs with tighter grips on shaft and wheel, ensuring not to deform the wheel structure during the process.

Software-Modifications

The initial mouse movement was done at a very high speed of 400Hz. Observations showed that at that speed, the mouse would gather a lot of momentum and could easily be diverted. The mouse at this stage also showed signs of slight “missing steps”, where it appeared the motor had skipped pulses, and it would not perform optimally. This was due to a coding error which shall be discussed later on.

Distance Sensor –

The distance sensor was needed after movement was achieved in the micromouse. We began measuring where we needed the mouse to stop in order to do a turn whether left, right or U-Turn without collision. Here we had major issues with dead zone, described in coding section as well as noise from sunlight, this caused

mouse to travel in further and stop further out than wanted. We had to do some tests to avoid this interference and made sure considering the environment under which tests were carried out. Other actions taken include lowering the sensor into the maze almost touching the ground. This meant that the walls would block a great amount of sunlight. Also, cleaning the lens of the receiver of the distance sensor helped as well.

Software Modifications:

The countering of the dead zone has been discussed already. But to counter the false measurements we had to do few things:

1. We reduced the number of pulses for the forward movement, so that at a time of 30 pulses, the distance sensor values would get updated. This made sure that at a given time, we take more measurements of the values, and averaging the results would give us an optimum value to consider.
2. The wall detection voltage value was changed to keep it at a steady rate. Rather than making the mouse detect 10cm from the wall (where voltage values would heavily fluctuate due to the exponential graph), we made the mouse detect wall 16-17cm in advance. At this range, the output voltage was much more steady.

Tunnel Run –

Tunnel Run means the mouse should be able to travel down a straight path and turn left or right at the end of that path. Then continue until it encounters a wall in-front after which it does a complete U-Turn and returns to the start. A basic tunnel run layout is in the form of an L-Shaped path.

We needed the mouse to move dead straight and detect walls on both sides then make a decision.

Step one was figuring how many pulses were required to turn 90 degree in both directions (clockwise and anticlockwise turns) and for a full U-Turn.

Turning was approached in two ways. One was to stop one wheel and turn the other to make a turn. The other method involved rotating the wheels opposite each other.

The first method (stopping one wheel and rotating the other) gave a lot of friction while making a turn (due to the amount of weight being rested on one wheel and the other dragging that weight, giving unequal distribution of centre of mass).

The second method proved to be more sensible, as equal division of weight was done by both the rotating wheels. This made sure that centre of mass was stable and furthermore posed a lot less friction as both wheels had equal division of weight.

We found the issue was caused in turning because the momentum from movement(straight path movement) carried during the turn would affect the turns at times because at that time the centre of mass of the mouse was in motion(not fixed). In order to do a complete turn, the centre of mass had to be fixed, otherwise the mouse would skid. Therefore we implemented a delay before each turn to make each move separate from the last, ensuring the momentum disappears before making a turn. This helped as turns were more perfect than before.

Furthermore during this phase we realised that if the surface was not smooth at some instances the wheels would lift and the number of turns would be completed without the mouse actually turning. We decided to add suspension to the mouse, making the front/back uneven to add a slight rock, ensuring that the mouse

would go over some rough patches and would not slow down the front ball caster. This proved successful as the mouse would turn better more often although not perfect.

Step two: After the turning phase, sensors were looked upon to ensure centring as well as availability of turns.

Side sensors (two for wall-detection and two for centring) were implemented to help the mouse correct and check for availability of turns. The reason for adding the side sensors was simple. We noticed there was a flaw with the wing. If the wing went over the top of the wall, then there was no saving the mouse from collisions. Thus we had discarded the wing and had opted for the side proximities to check for presence of walls, both for turns as well as centring. The wall detection sensors would have the maximum sensitivity ensuring that walls would be detected. The centring proximities would work on the principle that if the mouse got near a wall then they would turn on and the respective action would be taken to correct it.

However the side sensors proved to be futile at first. Our initial prototype for side sensors involved placing them at the front, and to counter the interference we would pulse them at different times. Difficulties with this method have been discussed in the software part of this section.

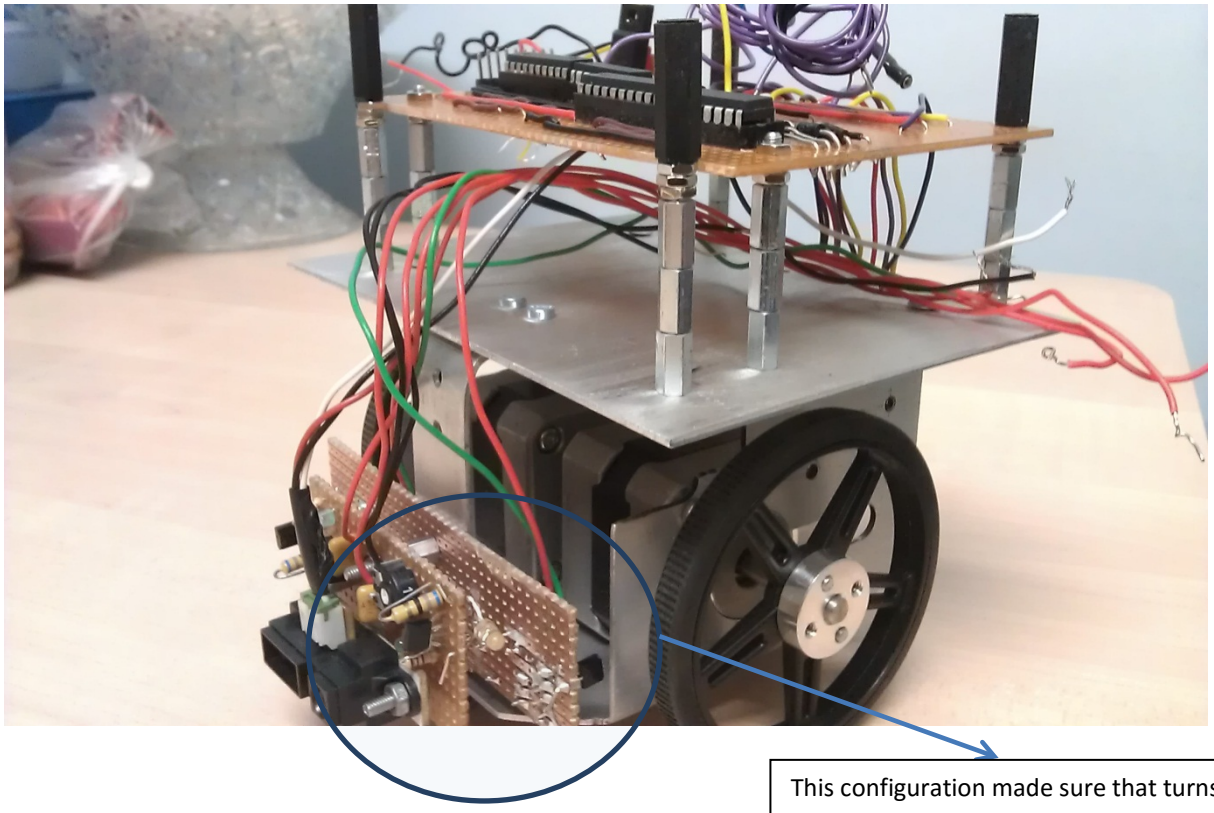


Figure 5.2.1Our mouse design before the front and back sensors and the wing sensors were used

Due to difficulties in pulsing them at different times, we opted for this configuration where the front sensors (centring) would be at a distance from wall detection (back sensors). And we put the receivers at opposite ends to ensure that the sensors would have as less interference possible. The above prototype was effective in detecting the presence/absence of walls however centring was not upto the mark.

The above prototype was able to avoid collisions, but took a long time to detect the presence of walls. The reason being the proximities had a lag if the sensitivity was increased too much. It worked perfectly for distances of 1-2cm from the wall. However we could not get the sensors to detect at a distance of 3cm from the wall. Even if we made it detect, there was a lag in the proximity going off even after correction had been made. This made the mouse overcorrect itself.

We concluded that the proximities were effective at detecting walls at a distance of 1-2cm, but anything more than that posed great lags in readings. Furthermore, the mouse had to correct even the slightest deviation (which was not possible without lags with the above configuration) as slight deviations, if left uncorrected, would gradually build up and cause problems.

Software Modifications:

Initially we wanted to pulse the sensors, to save power. Furthermore we could also reduce interference by turning them on at different times. We were unable to pulse the sensors however, as the readings would go down significantly (maybe due to lack of current). The other reason being is the use of potentiometers, which favoured steady voltages rather than pulsing voltages. Thus we discarded the idea of pulsing sensors and decided to supply them directly from mouse's main power.

During the tunnel run phase, the missing steps was taken care of as well. The problem was that the output waveform was heavily deformed and we had not checked it on the oscilloscope. This deformed waveform did not have a good duty cycle (had about 90%), which lead to the motors performing less smoothly and missing few steps. We verified the output waveform on Proteus and made sure 50% duty cycle was met.

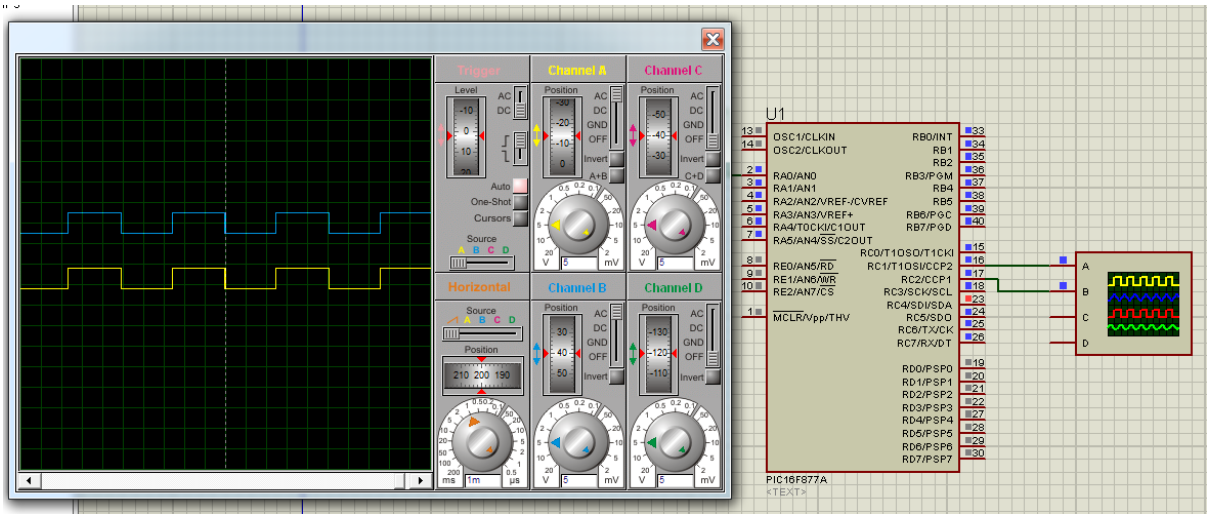


Figure 5.2.2 Proteus schematic for simulating the pulses of the motor

Lastly, the code was kept very simple. We did not check for walls every square, but only checked once the mouse had reached a wall. Only then the mouse would check for availability of turns and proceed. We mainly focused on sorting the centring issue out during this phase. This meant changing the rate at which centring checks were done (so that mouse doesn't spend too much time correcting). We did not use tables for sensor checks as we used a much simpler approach.

Prototype Maze(no left hand)

After ensuring that the mouse could do complete tunnel runs quite a few times without any problems, we decided to see how the mouse performed in a simple maze, requiring loads of turns and straight paths (basically a deformed tunnel run).

Here we realised that the previous side proximity configuration would not work, due to the lag in the proximity sensor detecting the deviation. Even after increased sensor sensitivity, the mouse needed to pick up slight deviations, which was proving impossible with the previous prototype.

Only way to compensate for correction was to physically bring the sensors closer to the wall. This resulted in widening of the Vero board for centring proximities and also to completely remove interference, placing the wall detection sensors and correction sensors at opposite ends of the mouse.

Therefore we split both the boards with one being placed in the front with the distance sensors used for centring and the back proximities on the sides only for checking for wall detection. The modified version of the mouse is shown below

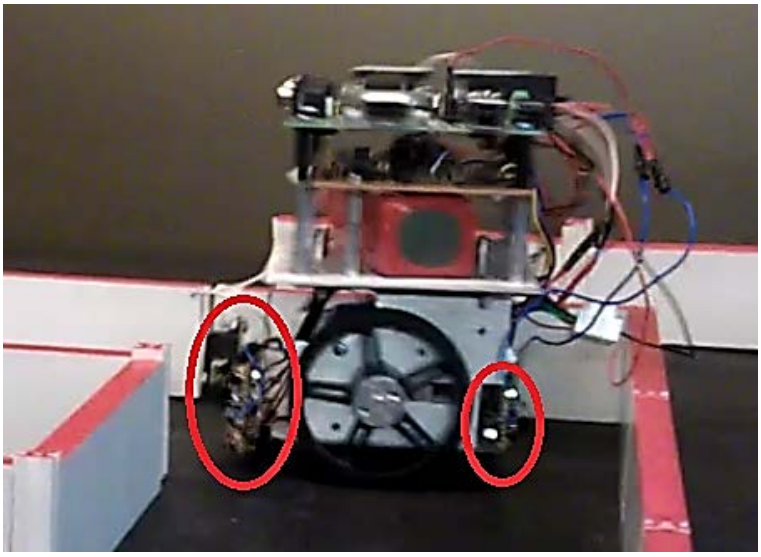


Figure-5.2.3:Picture of the mouse with the front and back sensors and the wing

This design really helped with the accuracy of corrections and has been able to solve most of the problems we encountered before. But once again, the distance of the proximities from the wall and their sensitivity proved to be a problem as the mouse was still deviating and lag of sensor detection meant that correction was not immediate, causing errors to gradually build up and making the mouse collide after half a run of the twisted tunnel.

We physically needed to move the proximities as close to the wall as possible, and reduce their sensitivity such that the mouse is able to detect even 5mm of deviations.

One way we could have countered this problem is by moving the proximities checking for wall collisions closer to the wall by increasing the size of the Vero board they were on. But this would cause problems while making a turn.

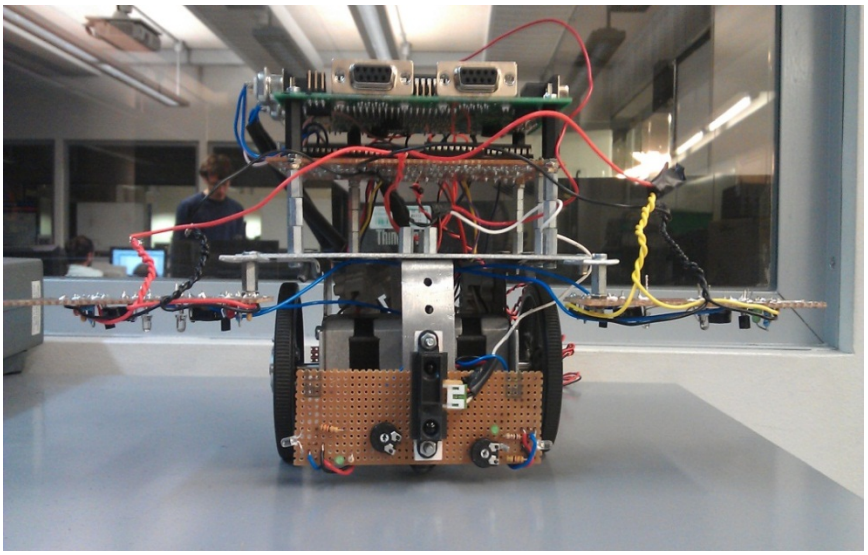


Figure-5.2.4-Final Design of the mouse

Therefore we adopted the wing design. With this design we were able to fine tune the corrections made by the collision sensors to execute perfect centering. The main reason this design worked, was because we now had in total, 6 sensors just to serve the purpose of correction. The collision sensors ensured mouse centred, even if the wing failed and the wing helped detect those slight deviations that the mouse would make which the side proximities had failed to do so.

The design is actually an "inverted wing design", as both sensors are off when mouse is in centre and only one came on, after deviation. Reason both were off is because, the wing was only used for one thing and that is to detect slight deviations. The initial wing design tried to do two things at once i.e. detect presence of walls and centre at the same time. However the current design made sure wall detection and centring were simplified and made independent of each other. We believe this simplification was a true benchmark for our mouse.

The mouse was able to perform very well, able to navigate through the twisted tunnel with ease and without any collisions. To incorporate the wing sensors, the following software modifications were made during this phase:

Software modifications:

1. No software modifications were made when the front proximities were detached and placed at the back.
2. After incorporating the wing design, sensor checks for centring required the checking of 6 sensors at a time. Furthermore, sensor checks employed a rule of priority, giving collision sensors higher priority than the wing sensors.
3. We had to ensure that amount of pulses required to correct the mouse was minimal and did not result in over correction. So a motor would skip two pulses to correct deviation.
4. Our right wing was much more sensitive than the left wing. We tried our best to ensure they both are the same, but was not possible. Thus due to sensitivity difference, we prioritised the right wing checks i.e. right wing was always checked first before checking the left wing. However the sensitivity was not that big of a deal.
5. There was an increase in waveform frequency from 200Hz to 220Hz. This was the maximum momentum our mouse could carry without any skidding of wheels.
6. The extra information from sensors ensured good operation and navigation of the mouse.

Left Hand

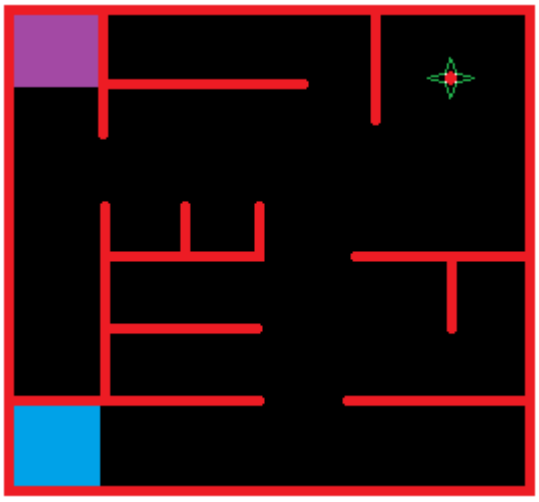
The Left Hand Algorithm works on the rule of following the left wall continuously until it leads to the centre. The Micromouse senses the wall on the left, and follows it to wherever it leads until the centre is reached. This simple logic used for solving the maze is demonstrated by the following algorithm.

1. First it checks if there is an available left turn. If so we execute a left turn.
2. If there isn't an available left turn it checks if there is a front wall. If so, it moves forward by one square and loops back to the first step.
3. If there is a front wall present, it checks if there is a right wall. If not, the mouse executes a right turn and loops back to step-1.
4. Finally, if does detect a right wall, then we have encountered a situation where there is a left wall, right wall, and a front wall. This means we have reached a dead end and therefore we execute a 180^{o} turn and loop back to step 1.

The flowchart for the algorithm implementation is shown in APPENDIX-1

Testing of the algorithm:

We have used the following maze to test the mouse with all the modifications made.



Figue-5.2.5:Maze used for the algorithm testing

In the maze we see that there is a pin, circled in green. The presence of this was very vital because the objective of the mouse was to so from the start square (painted in blue) to the end square (painted in purple) and on its way back to the start square, it performs a 360^0 turn around that pin and go back to the start square. Due to this we were able to fine tune the delays and the pulses in the code and obtain the final code shown in appendix-2.

Other issues -

- using DC meant we had extra-long wires that hand to be held to prevent tangle
- wires that on circuits sometimes touch the walls, cutting wires short and making the circuit higher help
- emitter failure, emitters would sometimes become faulty after extended use after doing some research we found the issue to be related to the fact that we used smaller resistors than recommended, we then added smaller resisters to the variable resistor to act as a base resistance
- broken emitters, emitters would sometimes break when the mouse had a misfortune of colliding, we had to unsolder some emitters and re-solder them further in towards the mouse centre to avoid collision, we initially had the idea of placing rams/bumpers towards the front and rear but when attached these caused vibrations, also when they collided they cause a severe jolt in the mouse motion that made movement harder.

Conclusions

Numerous conclusions can be made from the project. They range from general points like group co-operation, budgeting and time management to decision making and learning experiences.

- As a group we had to use our individual knowledge to accomplish tasks. We had to further strengthen our skills in areas where we were already competent in, at the same time attempting to strengthen our weak areas. Tasks had to be assigned while keeping this in mind. This showed us the power of team work and how we could depend on each other to complete a task.
- There was a budget linked with our design. During building we used more components than expected, we had to find solutions to resolve problems quickly, e.g. when the emitters kept getting destroyed, it was costing a lot to get them replaced with next day shipping. We had to find a solution in our design to help save emitters; we knew we had to be conscious about our design in all particular aspects.
- Time management – with exams and deadlines round the corner we had to find time to meet as a group. We made use of technology by using Skype via online video conferencing. Using Dropbox, a multimedia sharing website, we were able to instantly send each other media, datasheets and written reports. Furthermore we also used phone instant messaging applications such Whatsapp to create group conferences to communicate while we were on the go.
- Difficult decisions had to be made when thinking about what directions to take with certain suggested ideas. The best example worth mentioning was deciding against using Lee's algorithm to solve the maze and opting for the Left Hand algorithm. Here we had to discuss what was achievable and realistic given the up and coming deadline within the given time period. Although we attempted the Lee's algorithm, we had to ensure that we concluded having a mouse that could navigate the maze with some sort of algorithm first. This turned out to be a wise decision since we could not complete Lee's Algorithm in time for the demonstration. By having the Left Hand algorithms a fully functional backup we managed to gain 2nd place in the competition. We think given enough time we could have got Lees Algorithm to work we had already got to the point where we can know what square the mouse is in and had the initial Lees number stored to memory. Looking back we should have used this knowledge of learning which square the mouse was in to make the mouse recognise when it was at the centre using Left Hand.
- A lot of things were learnt while coding the micro-mouse using Assembly. We gained extensive knowledge using macros, as well as using timers to produce pulses for motors. We also learnt to manipulate physical information and use it to our advantage e.g. making the mouse detect walls.

Group Meetings

Member(s) present	Topic(s) discussed& Work done	Location	Time	Date
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Chassis	Home (Skype)	20:00-22:00	22 January 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Chassis	Home (Skype)	21:00-23:00	23 January 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Motors	Home (Skype)	20:00-23:00	26 January 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Motors	Home (Skype)	21:00-22:00	29 January 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Wheels • Hubs	Home (Skype)	20:00-22:00	5 February 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Wheels • Hubs	Classroom	18:00-19:00	9February 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Sensors	Classroom	18:00-19:00	16February 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Sensors	Classroom	18:00-19:00	23 February 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Battery • Battery Charger	Lab	13:00-22:00	29 February 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Battery Bag	Lab	11:00-22:00	7 March 2012

Member(s) present	Topic(s) discussed& Work Done	Location	Time	Date
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Coding	Lab	15:00-22:00	14 March 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Sensor Configuration	Lab	12:00-22:00	28 March 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Sensor Configuration	Lab	12:00-22:00	2 April 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Sensor Configuration	Lab	13.00-22:00	3 April 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Writing Code • Hardware	Lab	13:00-22:00	4 April 2012 (nearly every day up until May)
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Writing Code • Hardware	Lab	14:00-22:00	1 May 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Writing Code • Hardware	Lab	14:00-22:00	2 May 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Writing Code • Hardware	Lab	16:00-22:00	3May 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Writing Code • Hardware • Testing Micro-mouse on Maze	Lab	12:00-22:00	4 May 2012
Sam Sunani DeujeanCabey NaeemHosany AnjaniMaddala Jules Pinto	• Writing Code • Testing Micro-mouse on Maze	Lab	11:00-22:00	7 May 2012 (nearly every day up until 24th May)

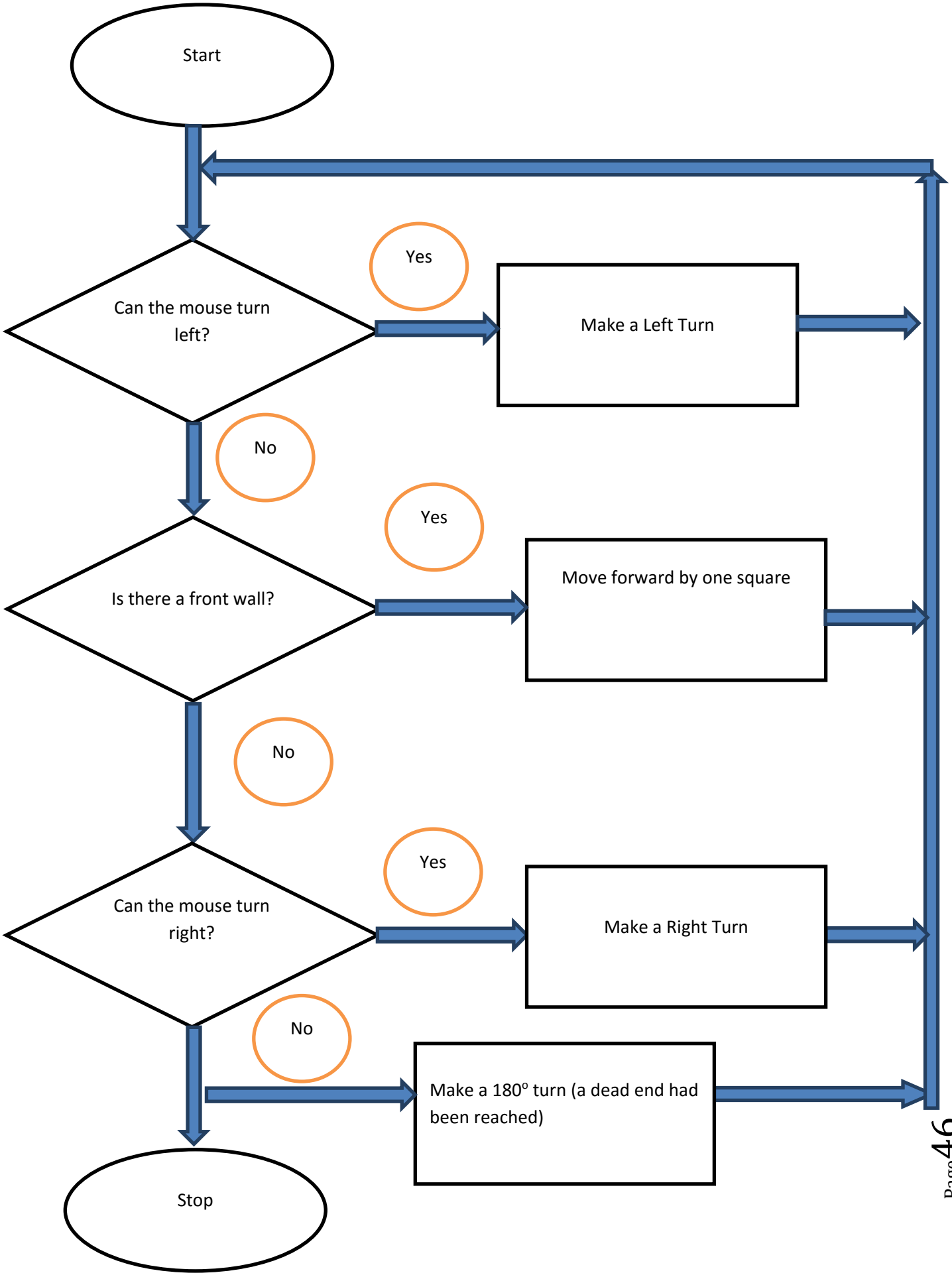
Expenditure

Item(s)	Cost
Wheels	**£5**
Hubs	**£5**
ICD2	**£32**
Lithium Polymer Battery	**£39**
Lithium Polymer Fire Proof Bag	**£6**
Lithium polymer Charger	**£40**
IR Emitters	**£15**
IR Receivers	**£30**
Sharp Distance Sensor	**£20**
Solder	**£15**
Padlock	**£15**
Chopblock	**£10**
Heatshrink tubing	**£2**
Wires (female and male)	**£4**
9pin D-Sub	**£7**
	Total: **£245**

Contributions

Name(s)	Percentage Contributed
Sameer Sunani	20%
Deujean Cabey	20%
Naeem Hosany	20%
Anjani Maddala	20%
Jules Pinto	20%

Appendix-1 Flowchart for the algorithm



Appendix-2: The Code

```
;*****************************************************************
;
;   Filename:                   Left_Mouse_Final.asm
;   Date:         20/05/2012
;   File Version:   Final Build
;
;   Author:      Echelon V
;   Company:     Echelon V
;
;
;*****************************************************************
;
;   Files Required: P16F877A.INC
;
;*****************************************************************
;
;   Notes: The Final Code for the Micromouse.
;               The code prioritises the distance sensor reading, and every 330 pulses of the motor
;               checks for an available left turn. 330 pulses is equivalent to 1 square distance
;               travelled by the mouse.
;               During the turn checking phase of the software, priority is changed
;               from the distance sensor to sensor wall detection and availability of a
;               left turn is checked.                   *
;
;               RA0= Distance Sensor input for ADC Conversion
;
;               PORTD= all bits have been used for proximity sensor readings
;                               PORTD Higher Nibble:
;                                       used for the wing design. Each Bit from left to right
;                                       is used to represent sensors from left to right on the wing.
;                               PORTD Lower Nibble
;                                       used for wall detection as wall as collision avoiding.
;                                       RD3 and RD0 used for wall detection
;                                       RD1 and RD2 used for collision checks


;               RC1=Right Motor
;               RC2=Left Motor
;               RC3=Right Motor Direction Pin
;               RC4=Left Motor Direction Pin

;               TABLES have not been used to check for sensors. Rather simple
;               xorwf scheme has been employed.
;
;               Timer interrupts are only enabled during counter check
;*****************************************************************



               list                    p=16f877A  ; list directive to define processor
               #include    <p16f877A.inc>          ; processor specific variable definitions

               __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _RC_OSC & _WRT_OFF & _LVP_ON & _CPD_OFF

;################### VARIABLE DEFINITIONS ########################################
w_temp                  EQU     0x7D                    ; variable used for context saving
status_tempEQU          0x7E                    ; variable used for context saving
pclath_temp             EQU     0x7F                    ; variable used for context saving
RP0                     equ                     0x05                                                    ;PAGE BIT FOR STATUS REGISTER
Z                       equ                     0x02                                                    ;ZERO BIT

STATUS                  equ                     0x03                                                    ;STATUS REGISTER
INTCON                  equ                     0x0B                                                    ;INTERRUPT REGISTER
TRISB                   equ                     0x86                                                    ;I/O REGISTER FOR PORTB
PORTB                   equ                     0x06                                                    ;PORT REGISTER FOR OUTPUT OR INPUT
PORTA                   equ                             0x05

TRISA                   equ                             0x85
Z                               equ                             0x02
GO                              equ                             0x02

RP0                     equ                     5
TOIF        equ                 2
TOIE        equ                 5
GIE         equ                 7
RB0                             equ                             0
RB1                             equ                     1
RB2                             equ                             2
RB3                             equ                             3
```

```
RC1                     equ             1
LSB                     equ             0
RA7                     equ             7
RA6                     equ             6

;Registers used for delays, counters and memory management

        CBLOCK          0x20
                        inNUM
                        asc100
                        asc10
                        asc1
                        thisDIG
                        count
                        count1
                        count2
                        COUNTA
                        COUNTB
                        COUNT20
                        COUNTA5
                        COUNTB5
                        countd
                        move
                        PORTD_TEMP
                        TURN_COUNT
                        LEFT_COUNT
                        FORWARD_COUNT
                        OUTER_FORWARD_COUNT
                        left_motor_control1
                        right_motor_control1
                        left_motor_control2
                        right_motor_control2
                        ENDC


;*****************************************************************
        ORG     0x000           ; processor reset vector

        gotomain_start          ; go to beginning of program

        ORG     0x004           ; interrupt vector location
        goto            isr_start

;-------------------------------------------------------------------------
;                                                       MACROS
;-------------------------------------------------------------------------
; basically used to assign counter numbers for making sure motors get pulsed a fixed number of times.
; CCP control has been further enhanced with the use of left_motor_control1 & 2 register and
;       right_motor_control1 & 2 registers. These allow to turn a motor ON or OFF by setting the
;       right values into them.

;*************************************************************************
;forward_move Macro
;*****************************************************
;       This macro is used to pulse the motor a fixed number of times. cnt1 is resposible
; for the inner loop in the counter check, and cnt2 controls the outer loop in the counter check
; The counter for interrupt will pulse only the number of times dictacted by cnt1, via using the
; counter Register FOWARD_COUNT. Here both the motors are turned ON


forward_move            MACRO   cnt1,cnt2
                                movlw   cnt1
                                movwf   FORWARD_COUNT

                                movlw   cnt2
                                movwf   OUTER_FORWARD_COUNT

                                bcf                     PORTC,RC3
                                bsf                     PORTC,RC4

                                movlw   b'00001001'
                                movwf   left_motor_control1
                                movwf   right_motor_control1

                                movlw   b'00001000'
                                movwf   left_motor_control2
                                movwf   right_motor_control2

                                endm

;*********************************
```

```
;turn_setter Macro
;*********************************
; This specifies the Number of pulses required for the motor to make a complete 90 degree turn.
; Number of pulses is set be TURN_COUNT register.
; here both motors are turned ON.

turn_setter             MACRO   cnt1

                                movlw   cnt1
                                movwf   TURN_COUNT

                                movlw   b'00001001'
                                movwf   left_motor_control1
                                movwf   right_motor_control1

                                movlw   b'00001000'
                                movwf   left_motor_control2
                                movwf   right_motor_control2

                                endm

;*********************************
;lft_crct_setter Macro
;*********************************
; TURN_COUNT =specifies the number of Pulses require to make a slight adjustment or avoid collision when
;               mouse is deviated towards the left
;Here right motor is turned OFF and left motor is TURNED ON
lft_crct_setter         MACRO   cnt1
                                movlw   cnt1
                                movwf   TURN_COUNT

                                movlw   b'00001001'
                                movwf   left_motor_control1     ;make sure that only left motor turns

                                movlw   b'00001000'
                                movwf   left_motor_control2

                                clrf    right_motor_control1    ;right motor doesnt move
                                clrf    right_motor_control2
                                endm

;*********************************
;rt_crct_setter Macro
;*********************************
; TURN_COUNT =specifies the number of Pulses require to make a slight adjustment or avoid collision when
;               mouse is deviated towards the left
;Here left motor is turned OFF and right motor is TURNED ON

rt_crct_setter          MACRO   cnt1
                                movlw   cnt1
                                movwf   TURN_COUNT



                                movlw   b'00001001';clears the pin, so you have a proper sine wave
                                movwf   right_motor_control1

                                movlw   b'00001000'
                                movwf   right_motor_control2    ;only right motor will turn

                                clrf    left_motor_control1             ;left motor wont move now.
                                clrf    left_motor_control2

                                endm
;=============================================================================
;                               INITIALIZERS
;=============================================================================

;*****************************************************************************
;               ANALOG TO DIGITAL PROCEDURE
;*****************************************************************************
;INIT A/D MODULE
;CONFIGURE PIC I/O LINES
;2. SELECT PARTS TO BE USED BY SETTING THE PCFGX BITS IN ADCON1 SELECT ADFM AS WELL
;3. SLEECT ANALOG CHANNELS A/D CONVERSION CLOCK AND ENABLE A/D MODULE
;4. WAIT ACQUISITION TIME
;5. INITIATE ONVERSION BY SETTING GO/DONE BIT IN ADCON0
;6. WAIT FOR CONVERSION TO COMPLETE
;7. READ AND STORE DIGITAL VALUE

Init_A2D
```

```
            BANKSEL     TRISA
            MOVLW       b'00000001' ;RA0 AS INPUT
            MOVWF       TRISA
            MOVLW       b'00001110'
            MOVWF       ADCON1                  ;ADFM=0 HENCE LEFT JUSTIFIED
                                                                    ;0011-RA0 AS ANALOG AND RA3 IS VREF+ REST ARE DIGITAL
                                                                    ;SET ADCON0 THE MAIN ADC CONVERTOR ROUTINE
            BANKSEL ADCON0
            MOVLW       b'01000000' ;FOSC/8
                                                                    ;CHANNELS AS 000=RA0
                                                                    ;ADON=1 HENCE TURN ADC MODULE ON
            MOVWF       ADCON0

            BCF                     INTCON,ADIE
            BSF                     ADCON0,ADON
            CALL        DELAY_20uS

            RETURN

;*****************************************************************************
;           READ A/D LINE
;*****************************************************************************
;READ VALUE AND CONVERT TO DIGITAL, MAIN ADC ROUTINE FOR CONVERSION

Read_A2D
            CALL        DELAY_20uS
            BANKSEL     ADCON0
            BSF                     ADCON0,GO;SET GO BIT TO START ADC CONVERSION...AUTOMATICALLY GETS CLEARED WHEN CONVERSION IS
COMLPLETE

convWAIT
            BTFSC       ADCON0,GO
            GOTO        convWAIT

;READ MSBS
            MOVF        ADRESH,0
            BANKSEL     PORTB
            RETURN
;*****************************************************************************
;           BINARY TO ASCII VALUE DECIMAL CONVERSION
;*****************************************************************************
 ;ON ENTRY:
;           W HAS BINARY VALUE IN RANGE 0 TO 255
;ON EXIT:
;           OUTPUT VARIABLE asc100, asc10 AND asc1 HAVE THREE DECIMAL UNITS
;VALUE 100 IS SUBTRACTED FROM SOURCE OPERAND UNTIL 0 REMAINS
;NUMBER OF SUBTRACTIONS IS THE DECIMAL HUNDREDS RESULT. 100 IS ADDED TO COMPENSATE FOR LAST SUBTRACTION
;10 IS SUBTRACTED IN SAME MANNER
; VARIABLES
;inNUM      -STORAGE SPACE FOR OPERAND
;asc100
;asc10
;asc1
;thisDIG
bin_2ASC

            MOVWF       inNUM                           ;COPY OF SOURCE VALUE
            CLRF        asc100
            CLRF        asc10
            CLRF        asc1
            CLRF        thisDIG

sub100
            MOVLW       d'100'
            SUBWF       inNUM,1
            BTFSC       STATUS,C                ;DID SUBTRACT OVERFLOW
            GOTO        bump100                         ;NO COUNT OVERFLOW
            GOTO        end100

bump100
            INCF        thisDIG,1
            GOTO        sub100
;STORE 100TH DIGIT

end100
            MOVF        thisDIG,0               ;DIGIT COUNTER
            ADDLW       0x30                            ;CONVERT TO ASCII
            MOVWF       asc100                          ;STORE THE VALUE
```

```
;CALCULATE THE 10S POSITION
            CLRF        thisDIG
;adjust value
            MOVLW       d'100'
            ADDWF       inNUM,1

sub10
            MOVLW       d'10'
            SUBWF       inNUM,1                 ;SUBTRACT 10
            BTFSC       STATUS,C        ;CHECK FOR OVERFLOW
            GOTO        bump10
            GOTO        end10

bump10
            INCF        thisDIG,1       ;INCREMENT DIGIT COUNTER
            GOTO        sub10

end10
            MOVLW       d'10'
            ADDWF       inNUM,1                 ;ADJUST FOR LAST SUBTRACTION
            MOVF        thisDIG,0
            ADDLW       0x30                    ;CONVERT TO ASCII
            MOVWF       asc10                   ;STORE THE DIGITAL VALUE
;CALCULATE AND STORE UNITS DIGIT
            MOVF        inNUM,0                 ;STORE UNITS VALUE
            ADDLW       0x30
            MOVWF       asc1
            RETURN
;************************
; DELAY FOR ABOUT 40uS
;************************

DELAY_40uS
            MOVLW       d'8'
            MOVWF       COUNTA

            NOP
            NOP
            NOP

LOOP_40
            NOP
            DECFSZ      COUNTA,1
            GOTO        LOOP_40
            RETURN

;************************
; DELAY FOR ABOUT 20uS
;************************
DELAY_20uS
            MOVLW       d'5'
            MOVWF       COUNT20
LOOP_20
            DECFSZ      COUNT20,1
            GOTO        LOOP_20
            RETURN


;##############################***DELAYS***########################################
;############################### WAIT SUBROUTINE = 1 SEC ########################################

DELAY_1S
                        movlw       d'24'                                   ;SET OUTERMOST LOOP TO 24

                        movwf       count                                   ;MOVE STATEMENTS=2
INSTRUCTION CYCLES
                        nop
                        nop
                        nop
                        nop
                        nop
                        nop
                        nop
                        nop
                        nop
                        nop
                        nop                                                                     ;11
NOPS=11 INSTRUCTION CYCLES

loop_ext1   movlw       d'254'                                  ;BEGINNING OF OUTERMOST LOOP
                        movwf       count1                                  ;2 INSTRUCTION CYCLES
```

```
                        nop
                        nop
                        nop
                        nop
                        nop
                        nop                                                                     ;6
NOPS=6 INSTRUCTION CYCLES

loop_ext    movlw       d'16'                                   ;BEGINNING OF MIDDLE LOOP
                        movwf       count2                              ;2 INSTRUCTION CYCLES

loop                nop                                                                     ;BEGINNING OF
INNERMOST LOOP
                        nop
                        nop
                        nop
                        nop
                        nop
                        nop                                                                     ;7
NOPS=7INSTRUCTION CYCLES

                        decfsz      count2,1                ;INNERMOST LOOP (10 CYCLES)-WILL
REPEAT COUNT2(16) TIMES
                        goto        loop                            ;((16x10)-1)x254x24=969,264
CYCLES

                        decfsz      count1,1                ;MIDDLE  LOOP (5 CYCLES)-WILL REPEAT
COUNT1(254) TIMES
                        goto        loop_ext                ;(5x254-1)x24=30,456

                        decfsz      count,1                         ;OUTER LOOP         (11
CYCLES)-WILL REPEAT COUNT(24) TIMES
                        goto        loop_ext1               ;11x24-1=263

                        return                                                                  ;2 INSTRUCTION
CYCLES


;*****************************************************************************************
; DELAY FOR 5msec; Delay equation : [50*([199*10-1]+10)-1]+2+2+2 = 10000cycles = 5msec (at Fosc=8MHz)
;*****************************************************************************************


delay_50ms

            MOVLW   D'35'
            MOVWF   COUNTA

OUTER_5
            MOVLW   D'199'
            MOVWF   COUNTB

            NOP
            NOP
            NOP
            NOP
            NOP


INNER_5
            NOP
            NOP
            NOP
            NOP
            NOP

            DECFSZ  COUNTB,1
            GOTO    INNER_5

            DECFSZ  COUNTA,1
            GOTO    OUTER_5

            RETURN
; ####################################################
; #########  set-up interrupts and ports  #############
; ####################################################
; set up port B pins to inputs

setup
            banksel TRISB
            clrf    TRISC               ; set all portc pins to outputs for motor PULSES
```

```
            clrf        TRISB                   ;set high for test LEDs
            movlw       0xFF
            movwf       TRISD                   ;PORTD all as inputs for sensor readings
            banksel     INTCON

; initialisation of interrupts
movlw   b'10000000'     ; enable interrupts globally and disable PIE
movwf   INTCON
            return


; ################################################
; #########                 set-up timer            ##########
; ################################################


timer_setup
            bcf                     INTCON,TOIF             ;clear timer overflow flag
                                                            ;always do this to reset the timer
            bsf                     STATUS,RP0 ;set page to bank1
            movlw       b'11010100'
            movwf       OPTION_REG              ;set timer to be clocked by fosc/4
                                                            ;set prescalar to 2:1
                                                            ;1XXXXXXX=pull-ups disabled
                                                            ;x1xxxxxx=interrupt on rising edge
                                                            ;xx0xxxxx=timer source clock internal
                                                            ;xxxx0xxx=prescalar assigned to timer
                                                            ;xxxxx100=32:1
            bcf                     STATUS,RP0 ;set page 0
            movlw       d'113'                  ;255-113=142 cycles between time-outs approx
                                                            ;therefor the period of the waveform becomes
                                                            ;142*32=4545uS or 4.545msmS which is 220Hz
            movwf       TMR0                    ;initialise timer value
            bsf                     INTCON,TOIE             ;enable timer overflow interrupt

;###########             INCLUSIVE TIMER1 SETUP
            movlw       b'00110100'
            movwf       T1CON                   ;00XX XXXX=we dont care what values they hold
                                                            ;XX11 XXXX=Timer1 Prescaler         11=1:8(ideal)
                                                            ;XXXX 0XXX=oscillator is disabled(internal RC)
                                                            ;XXXX X1XX=Do not Synchronize clock external input
                                                            ;XXXX XX0X=Timer1 Clock Source Select Bit=>Use internal Clock
                                                            ;XXXX XXX0=Timer1 ON BIT
            bsf                     T1CON,0                 ;TURN timer1 on
            bcf                     INTCON,TOIE             ;for now, turn timer0 off

            return


;################        MAIN PROGRAM ######################################
main_start

            clrf        STATUS
            clrf        PORTB
            clrf        PORTC
            clrf        PORTA
            call        setup
            movlw       d'11'
            movwf       LEFT_COUNT              ;Load initial counter for LEFT HAND Checking scheme
            call        timer_setup ;call timer setup
            call        Init_A2D    ;Initialse the ADC on PIC

read_distance
            ;part of the code that gives the forward movement priority
;the code first checks the distance sensor to see if the mouse is able to move.
;Two cases under the distance sensor:

            call        Read_A2D    ;now the w register has the ADRESH value, retrieves the Voltage
            call        bin_2ASC    ;converts the analog value to an ascii value, thus voltage in ASCII format

            decfsz      LEFT_COUNT,1            ;decrement it 11 times for periodic checks (per square)
            goto        left_check              ;check for availability of left turn

distance

;Here we interpret the voltage values from 1.25V to 0.40V
;           1.25 V to 0.40 V implies distance ranging from 4 cm to 15 cm
;           0.5V corresponds to exact 17cm distance
;checking is done systematically to check every number before and after decimal point.
;priority is given to see if front collision is possible (tackling the dead zone),
; If no values match, then that means
```

```
; that no oncoming obstacles are present and mouse continues to move forward

        movlw   '0'                     ;is voltage value 1.00??
        xorwf   asc100,0
        btfss   STATUS,Z
        goto    dont_move  ;if yes, goto suspend movement, if no, check for value after
                                                        ;decimal point

        movlw   '9'                     ;is voltage value 0.9V?
        xorwf   asc10,0
        BTFSC   STATUS,Z
        goto    dont_move  ;if yes, suspend movement, otheriwise check next value

        movlw   '8'                     ;is voltage value 0.8V?
        xorwf   asc10,0
        BTFSC   STATUS,Z
        goto    dont_move  ;if yes suspend movement, otherwise check next value

        movlw   '7'                     ;is voltage value 0.7V?
        xorwf   asc10,0
        BTFSC   STATUS,Z
        goto    dont_move  ;if yes, suspend movement, otherwise check next value

        movlw   '6'                     ;is voltage value 0.6V?
        xorwf   asc10,0
        BTFSC   STATUS,Z
        goto    dont_move  ;if yes, suspend movement, otherwise check next value

        movlw   '5'                     ;is voltage value 0.5V?
        xorwf   asc10,0
        BTFSC   STATUS,Z
        goto    wall_detected           ;if yes, wall has been detected. goto subroutine
                                                                ;if no, proceed down to "forward"

;-------------------------------------------------------------------------------
;none of the above values matched, so no oncoming front walls, hence prepare forward movement

forward

        forward_move            d'32',d'1'    ;pulse the motor 32 times


        bsf                     move,0                          ;set the move checking register high


        goto    test                                    ;test if no false readings are present
;-------------------------------------------------------------------------------
dont_move

        bcf                     move,0                  ;disable movement

        goto    sensor_check            ;mouse will check for available turns in the subroutine
;-------------------------------------------------------------------------------
wall_detected
        forward_move            d'170',d'1' ;pulse the motor 170 times once and check for possible collisions
                                                                        ;while it progresses to next square.
                                                                        ; check only once

        bsf                     move,0                  ;enable movement


        goto    test_center             ;check for false readings

;-------------------------------------------------------------------------------
test
        btfss   move,0                  ;check for false readings
        goto    read_distance           ;if there was false reading, check distance sensor
        goto    move_forward

move_forward

        bsf                     move,0
        bsf                     INTCON,TOIE             ;enable timer interrupt

        bcf                     PORTC,RC3               ;set direction pins for forward movement
        bsf                     PORTC,RC4
moving
        btfsc   move,0                  ;check if interrupt has been entered
        goto    moving
        bcf                     INTCON,TOIE             ;disable timer interrupt
        decfsz  FORWARD_COUNT,1 ;counter check
```

```
        goto    move_forward            ;pulses have not been completed, re enable timer
        goto    check_centering_sensors;check centering sensors after 30 pulses
;*******************************************************************************

;wall has already been detected a square in advance at this point of code
test_center
        btfss   move,0                  ;check for false readings
        goto    read_distance
        goto    move_forward_center_reset

move_forward_center_reset       ;reset counter value for forward movement
        movlw   d'170'
        movwf   FORWARD_COUNT

move_forward_center
        bsf             move,0
        bsf             INTCON,TOIE
        bcf             PORTC,RC3       ;direction pins for forward movement
        bsf             PORTC,RC4

moving_center
        btfsc   move,0
        goto    moving_center
        bcf             INTCON,TOIE
        decfsz  FORWARD_COUNT,1                 ;       inner loop for pulses
        goto    move_forward_center
        goto    wall_detect_centering
resume_forward
        decfsz  OUTER_FORWARD_COUNT,1           ;outer loop for pulses
        goto    move_forward_center_reset
        goto    sensor_check                            ;once the mouse has reached center of next square check for
available turns



;-------------------------------------------------------------------------------
;                               SENSOR CHECKS
;ZERO SIGNIFIES PRESENCE OF WALL
;ONE SIGNIFIES ABSENCE OF WALL
;-------------------------------------------------------------------------------
sensor_check

        movlw   d'11'
        movwf   LEFT_COUNT                              ;reset left check counter, so that it restarts per square checl

        movf    PORTD,0
        andlw   b'00001001'                     ;sensor mask for wall detection checks
        movwf   PORTD_TEMP                              ;store readings in temporary register


        movlw   b'00001000' ;left wall is absent AND RIGHT WALL IS PRESENT therefore turn left
        xorwf   PORTD_TEMP,0
        BTFSC   STATUS,Z
        goto    left_turn


        movlw   b'00000001' ;Right wall is absent AND LEFT WALL IS ABSENT therefore turn right
        xorwf   PORTD_TEMP,0
        BTFSC   STATUS,Z
        goto    right_turn

;*******************************************************************************
;U-TURN
;************************************************

        movlw   b'00000000' ;if there are walls present on both sides then turn left, and will turn left again
                                                                ;to complete u turn
        xorwf   PORTD_TEMP,0
        btfsc   STATUS,Z
        goto    left_turn


        goto    sensor_check ;keep checking sensors


;-------------------------------------------------------------------------------

left_check
        movlw   d'11'                   ;reset the LEFT checking counter
        movwf   LEFT_COUNT
```

```
        movf    PORTD,0
        andlw   b'00001001'     ;masking for wall detection sensors
        movwf   PORTD_TEMP              ;temporary register

        movlw   b'00001000' ;left wall is absent AND RIGHT WALL IS PRESENT therefore turn left
        xorwf   PORTD_TEMP,0
        BTFSC   STATUS,Z
        goto    left_turn

        movlw   b'00001001'     ;walls are absent on both sides of the maze, so turn left (priority)
        xorwf   PORTD_TEMP,0
        BTFSC   STATUS,Z
        goto    left_turn

        goto    read_distance   ;no turns available, so continue moving forward for 330 pulses


;-------------------------------------------------------------------------
;                                               TURNS
;-------------------------------------------------------------------------

left_turn
        turn_setter  d'153'     ;set turn pulses to be 153

        bsf             PORTC,RC3       ;direction pins toggled for anticlockwise turn
        bsf             PORTC,RC4

        call    DELAY_1S        ;delay is used to ensure stable turning of mouse

        goto    turn_set

right_turn
        turn_setter             d'153'  ;set turn pulses to be 153

        bcf             PORTC,RC3       ;direction pins toggled for clockwise turn
        bcf             PORTC,RC4

        call    DELAY_1S        ;delay is used to ensure stability after before turning

        goto    turn_set


turn_set
        bsf             move,0                  ;enable movement
        bsf             INTCON,TOIE                     ;enable timer overflow interrupt
turning
        btfsc   move,0                  ;check if interrupt has been entered
        goto    turning
        bcf     INTCON,TOIE
        decfsz  TURN_COUNT,1    ;keep pulsing until Turning has been complete
        goto    turn_set
        goto    read_distance   ;afer complete, check if mouse can move forward again

;-------------------------------------------------------------------------
;                                               CENTERING
;-------------------------------------------------------------------------

check_centering_sensors
;############### Checking for possible collisions###################

        movf    PORTD,0
        andlw   b'00000110'             ;mask collision sensors
        movwf   PORTD_TEMP                      ;store in temporary variable

        movlw   b'00000100'             ;mouse is deviated towards left(about to collide)
        xorwf   PORTD_TEMP,0
        BTFSC   STATUS,Z
        goto    left_collision_correct


        movlw   b'00000010'             ;mouse has deviated towards right(about to collide)
        xorwf   PORTD_TEMP,0
        BTFSC   STATUS,Z
        goto    right_collision_correct

;;############## check for right wing ############################

        movf    PORTD,0
        andlw   b'00110000'             ;mask sensors for right wing
        movwf   PORTD_TEMP                      ;store in temporary variable
```

```
        movlw   b'00100000'             ;mouse needs slight adjustment on right
        xorwf   PORTD_TEMP,0
        BTFSC   STATUS,Z
        goto    right_wall_centering

        movlw   b'00010000'             ;mouse needs slight adjustment on left side
        xorwf   PORTD_TEMP,0
        BTFSC   STATUS,Z
        goto    left_wall_centering

;################### check for left wing ########################

        movf    PORTD,0
        andlw   b'11000000'     ;mask sensors for left wing
        movwf   PORTD_TEMP              ;store in temporary variable

        movlw   b'10000000'     ;mouse needs slight adjustment on right
        xorwf   PORTD_TEMP,0
        BTFSC   STATUS,Z
        goto    right_wall_centering

        movlw   b'01000000'     ;mouse needs slight adjustment on left
        xorwf   PORTD_TEMP,0
        BTFSC   STATUS,Z
        goto    left_wall_centering


        goto    read_distance   ;if everything is centered then keep moving forward
;-------------------------------------------------------------------

left_wall_centering

        lft_crct_setter         d'2'    ;pulse motor 2 times for slight adjusment
        bcf                     PORTC,RC3       ;direction pins for forward movement
        bsf                     PORTC,RC4

        goto    turn_set

right_wall_centering

        rt_crct_setter          d'2'    ;pulse motor 2 times for slight adjustment

        bcf                     PORTC,RC3       ;direction pins for forward movement
        bsf                     PORTC,RC4

        goto    turn_set

left_collision_correct
        lft_crct_setter d'6'    ;pulse motors 6 times for collision correction
        bcf                     PORTC,RC3       ;direction pins set for forward movement
        bsf                     PORTC,RC4

        goto    turn_set

right_collision_correct
        rt_crct_setter d'6'             ;pulse motors 6 times for collision correction

        bcf                     PORTC,RC3       ;direction pins set for forward movement
        bsf                     PORTC,RC4

        goto    turn_set


;-------------------------------------------------------------------
;#####################################################################
wall_detect_centering
        movf    PORTD,0
        andlw   b'00000110'
        movwf   PORTD_TEMP              ;sensor masks for wall collision sensors

        movlw   b'00000100'     ;mouse is deviated towards left
        xorwf   PORTD_TEMP,0
        BTFSC   STATUS,Z
        goto    left_collision_WD

        movlw   b'00000010'     ;mouse is deviated towards right
        xorwf   PORTD_TEMP,0
        btfsc   STATUS,Z
        goto    right_collision_WD
```

```
        goto    resume_forward      ;goes back to where it came back from
;-------------------------------------------------------------------------

;centering after wall has been detected and mouse checks if it reaches next square precisely
left_collision_WD
        lft_crct_setter d'7'    ;correction for wall collision

        bcf             PORTC,RC3           ;direction pins for forward movement
        bsf             PORTC,RC4

        goto    turn_set_WD

right_collision_WD

        rt_crct_setter      d'7'    ;pulse 7 times to avoid wall collision

        bcf             PORTC,RC3           ;direction pins for forward movement
        bsf             PORTC,RC4
        goto    turn_set_WD

left_wall_centering_WD

        lft_crct_setter     d'3'    ;pulse 3 times for wall correction
        bcf             PORTC,RC3
        bsf             PORTC,RC4           ;direction pins for forward movement

        goto    turn_set_WD

right_wall_centering_WD

        rt_crct_setter      d'3'    ;pulse 3 times for wall correction

        bcf             PORTC,RC3           ;direction pins for forward movement
        bsf             PORTC,RC4

        goto    turn_set_WD
;-------------------------------------------------------------------------
;turn scheme after wall detection has been done and centering needs to be done
;to ensure mouse moves into the next square with as much precision as possible


turn_set_WD
        bsf             move,0
        bsf             INTCON,TOIE                 ;set timer interrupt

turning_wd
        btfsc   move,0                      ;check if interrupt has been entered
        goto    turning_wd
        bcf             INTCON,TOIE                 ;disable interrupt once it has come out of interrupt
        decfsz  TURN_COUNT,1
        goto    turn_set_WD                 ;decrement counter for turns(correction)
        goto    resume_forward      ;resume pulsing to ensure mouse reaches next square


; ###########################################################
; #############             interrupt service routine  ###############
; ###########################################################
isr_start
;context saving save
        movwfw_temp     ; save off current W register contents
        movf    STATUS,w    ; move status register into W register
        movwf   status_temp     ; save off contents of STATUS register
        movf    PCLATH,w    ; move pclath register into w register
        movwf   pclath_temp         ; save off contents of PCLATH register

;actual ISR routine
        call    timer_setup         ;set timer again, (makes TOIF clear)

        bsf             PORTC,RC1           ;set pulsing of both motors high(motor initiation)
        bsf             PORTC,RC2

        clrf    TMR1L               ;start TMR1 scheme
        clrf    TMR1H

        movlw   b'00011100'         ;this makes 284*8=2272(half period)
        movwf   CCPR2L              ;move this value into both registers for CCP module
        movwf   CCPR1L
        movlw   b'00000001'
        movwf   CCPR2H              ;set the higher registers
        movwf   CCPR1H
```

```
            movf        right_motor_control1,0  ;tells whether to turn right motor ON or OFF
            movwf       CCP2CON
            movf        left_motor_control1,0   ;tells whether to turn left motor ON or OFF
            movwf       CCP1CON

            clrf        TMR1L
            clrf        TMR1H
            movlw       b'00011100' ;this makes 284*8=2272(half period)
            movwf       CCPR2L                  ;move this value into both registers for CCP module
            movwf       CCPR1L
            movlw       b'00000001'
            movwf       CCPR2H                  ;set the higher registers
            movwf       CCPR1H

            movf        right_motor_control2,0  ;tells whether to turn right motor ON or OFF
            movwf       CCP2CON
            movf        left_motor_control2,0   ;tells whether to turn left motor ON or OFF
            movwf       CCP1CON

            bcf                 move,0                  ;clear flag to show mouse has entered interrupt, then decrement counter and resume
;end of ISR Routine
;context Saving restore

            movf        pclath_temp,w           ; retrieve copy of PCLATH register
            movwf       PCLATH                  ; restore pre-isr PCLATH register contents
            movfstatus_temp,w   ; retrieve copy of STATUS register
            movwf       STATUS          ; restore pre-isr STATUS register contents
            swapf   w_temp,f
            swapf   w_temp,w        ; restore pre-isr W register contents

            retfie
  end
```

References

[1] Last accessed on 10/05/2012
http://uk.farnell.com/astrosyn/129/stepper-motor-1-8deg-12v/dp/9598642

[2] Distance sensor graph.

http://letsmakerobots.com/files/userpics/u11554/pastedpic_06252011_223851.jpg

[3] http://www.matrixmultimedia.com/product.php?Prod=EB006&PHPSESSID=

[4]Lecture 3 notes p34 - Motors and Actuators by M.Giles

[5] Lecture 2 Notes p40- Sensors and Chassis by M.Giles