The University Of Sheffield.	Automatic Control and Systems Engineering

Date: 18/02/2008

Supervisor: Prof. D.J. Allerton

Module Code: ACS330

Group 1

Autonomous Scalextric Car
Group Interim Report

Group:	Tsz To Cheng	050196489	coa05ttc@shef.ac.uk
	Mohammed Bouheraoua	050201842	cob05mb@shef.ac.uk
	R.W Chathura Jayasinghe	050137633	coa05rwj@shef.ac.uk
	David Hermida Martinez	070212417	zzc07dh@shef.ac.uk

Index:

1. Introduction:

This project is to convert a standard Scalextric car to autonomous operation. Track is constructed using the standard Scalextic track components and car has to travel 10 laps on its maximum speed using on-board processing and sensors. The Scalextric car used is a Porsche Boxster.

1.1. Aims:

The project was aiming to work as a group to solve a control system design on a specific task and to understand how real-life project works. This can also give an opportunity to the group to be creative in design problem. Moreover, this should develop and enhance the ability of writing technical information in good presentation.

1.2. Objectives:

The objective of the project was to identify any design problems and solve these problems within the group using related concepts and theory. While solving the problems, the group should be able to analysis and design experiments that help to solve the problem under deadlines. At the end of the project, the group should also produce documents illustrating the progress of the project in a good format.

2. Project Management:

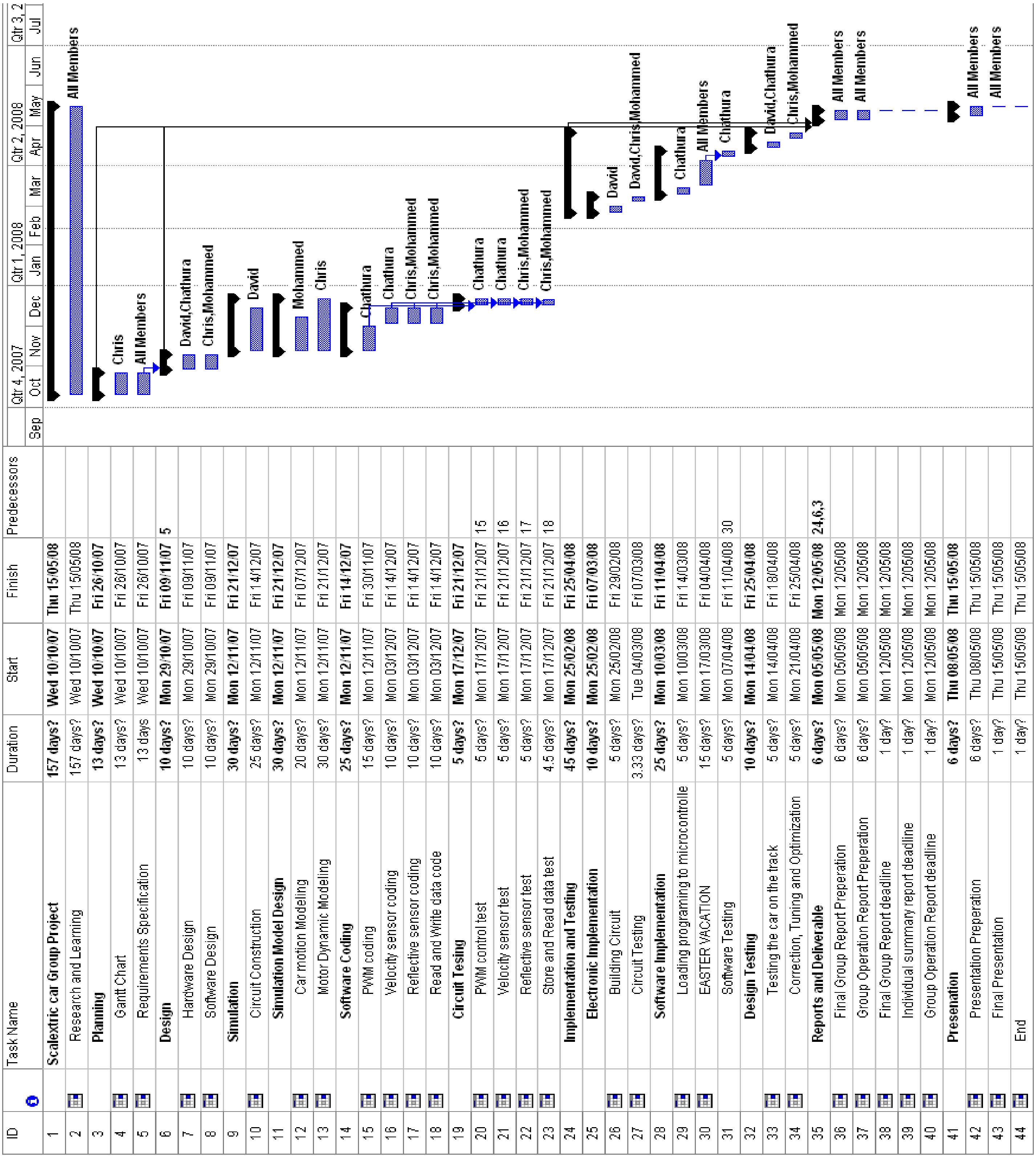
Initially, the group agreed to have a manager for the project which was responsible for organizing meetings, assigning work, checking for progress. When the group were planning the project, they have chosen a rotating manager system. The length of the managing period was spilt according to the project plan so every member had a fair chance of being the manager. One advantage of this approach was every member can learn and apply skills on management. Another advantage was there was a tendency of doing less technical work while managing the others. Therefore, this can eliminate the possibility of the manager not having enough technical work since the role was switching regularly. Meetings were held up regularly both with the supervisor and the group itself. In the meetings with the supervisor, the manager brief the supervisor about the work done, problems raised, and asked for direction.

The managers responsible for the tasks were as follows: Planning and Design – Mohammed Bouheraoua; Simulation – Tsz To Cheng; Implementation – R.W. Chathura Jayasinghe; Testing – David Hermida. When working through out the project, some problems raised. Since every members of the group came from different countries, language and culture difference were the problems. To overcome this, the only way for the group was speak slowly and have more patients. However, this gave the group a chance to work with different people. Another problem the group faced was knowledge. Because every member do different subject, the group was lack of knowledge in the electronic field. This makes the circuit construction more difficult. Moreover, sometimes the group found difficult to get the work done due to the other work load from other modules. This created delays on schedule and the group had to catch up the work and reschedule the plan.

Although the group approached the project according to the Gantt chart plan, when they progressed, they found some of the plan has to be changed. For example, in the simulation stage, the plan was to have an electronic simulation for the circuit before building, but they discovered that electronic simulation was not needed but instead building the circuit and test it along, so they changed the Gantt chart. More details were described in the following sections. They also discovered a good requirement specification document helps to all the tasks.

During the simulation stage, there were problems to access the lab. This was because the group had to have an assistor to access the lab, but the assistor was not always available.

2.1. Gantt chart:



ID	Task Name	Duration	Start	Finish	Predecessors
1	**Scalextric car Group Project**	**157 days?**	**Wed 10/10/07**	**Thu 15/05/08**	
2	Research and Learning	157 days?	Wed 10/10/07	Thu 15/05/08	
3	**Planning**	**13 days?**	**Wed 10/10/07**	**Fri 26/10/07**	
4	Gantt Chart	13 days?	Wed 10/10/07	Fri 26/10/07	
5	Requirements Specification	13 days	Wed 10/10/07	Fri 26/10/07	
6	**Design**	**10 days?**	**Mon 29/10/07**	**Fri 09/11/07**	5
7	Hardware Design	10 days?	Mon 29/10/07	Fri 09/11/07	
8	Software Design	10 days?	Mon 29/10/07	Fri 09/11/07	
9	**Simulation**	**30 days?**	**Mon 12/11/07**	**Fri 21/12/07**	
10	Circuit Construction	25 days?	Mon 12/11/07	Fri 14/12/07	
11	**Simulation Model Design**	**30 days?**	**Mon 12/11/07**	**Fri 21/12/07**	
12	Car motion Modeling	20 days?	Mon 12/11/07	Fri 07/12/07	
13	Motor Dynamic Modeling	30 days?	Mon 12/11/07	Fri 21/12/07	
14	**Software Coding**	**25 days?**	**Mon 12/11/07**	**Fri 14/12/07**	
15	PWM coding	15 days?	Mon 12/11/07	Fri 30/11/07	
16	Velocity sensor coding	10 days?	Mon 03/12/07	Fri 14/12/07	
17	Reflective sensor coding	10 days?	Mon 03/12/07	Fri 14/12/07	
18	Read and Write data code	10 days?	Mon 03/12/07	Fri 14/12/07	
19	**Circuit Tesing**	**5 days?**	**Mon 17/12/07**	**Fri 21/12/07**	
20	PWM control test	5 days?	Mon 17/12/07	Fri 21/12/07	15
21	Velocity sensor test	5 days?	Mon 17/12/07	Fri 21/12/07	16
22	Reflective sensor test	5 days?	Mon 17/12/07	Fri 21/12/07	17
23	Store and Read data test	4.5 days?	Mon 17/12/07	Fri 21/12/07	18
24	**Implementation and Testing**	**45 days?**	**Mon 25/02/08**	**Fri 25/04/08**	
25	**Electronic Implementation**	**10 days?**	**Mon 25/02/08**	**Fri 07/03/08**	
26	Building Circuit	5 days?	Mon 25/02/08	Fri 29/02/08	
27	Circuit Testing	3.33 days?	Tue 04/03/08	Fri 07/03/08	
28	**Software Implementation**	**25 days?**	**Mon 10/03/08**	**Fri 11/04/08**	
29	Loading programing to microcontrolle	5 days?	Mon 10/03/08	Fri 14/03/08	
30	EASTER VACATION	15 days?	Mon 17/03/08	Fri 04/04/08	
31	Software Testing	5 days?	Mon 07/04/08	Fri 11/04/08	30
32	**Design Testing**	**10 days?**	**Mon 14/04/08**	**Fri 25/04/08**	
33	Testing the car on the track	5 days?	Mon 14/04/08	Fri 18/04/08	
34	Correction, Tuning and Optimization	5 days?	Mon 21/04/08	Fri 25/04/08	
35	**Reports and Deliverable**	**6 days?**	**Mon 05/05/08**	**Mon 12/05/08**	**24,6,3**
36	Final Group Report Preperation	6 days?	Mon 05/05/08	Mon 12/05/08	
37	Group Operation Report Preperation	6 days?	Mon 05/05/08	Mon 12/05/08	
38	Final Group Report deadline	1 day?	Mon 12/05/08	Mon 12/05/08	
39	Individual summary report deadline	1 day?	Mon 12/05/08	Mon 12/05/08	
40	Group Operation Report deadline	1 day?	Mon 12/05/08	Mon 12/05/08	
41	**Presenation**	**6 days?**	**Thu 08/05/08**	**Thu 15/05/08**	
42	Presentation Preperation	6 days?	Thu 08/05/08	Thu 15/05/08	
43	Final Presentation	1 day?	Thu 15/05/08	Thu 15/05/08	
44	End	1 day?	Thu 15/05/08	Thu 15/05/08	

Figure 1 – Gantt chart

3. Project execution:

3.1. Requirements & specifications:

These were the requirements the group had done:

Overall requirements:

1 A Porsche car will be modified for autonomous operation by building a controller which will be placed on top of the car.
2 The only power supplied to the track is 12V (2A), there will be no additional power used; additional components will be used to minimize noise and polarity.
3 The navigation is done using a line sensor and which is connected to the controller as mentioned in 1 and only one white strip which is on the track to indicate the start no additional sensor are used neither modification to the track.
4 The C8146 motor will used to drive the car
5 The car must travel at maximum speed in the strait and an optimised speed in the curves so the car does not leave the track to achieve that the guidance system must be optimized to its best.
6 The car must complete at least 10 laps.

1. Autonomous operation:

1.1 PIC16F876 must be used in the car controller to perform the desired navigation, guidance and motor control details in [2].
1.2 The language used to program the microcontroller is C; function will be written separately for different operations then is linked to work together in the main.
1.3 An infrared port must be built in the controller to allow downloads of the software from a near by pc.
1.4 128k X 8 bit SRAM will be used to record the data on the track it will also be connected to 2 X 12 stage Binary counter to address it.

2. Navigation

2.1. A slotted disc with 15 slots and the SDP8600 optical sensor is attached to the motor shaft.
2.2. The SDP8600 optical sensor is powered by the 5V regulator.
2.3. The velocity is measured by the frequency of the slots on the disc detected by the SD-P8600 sensor when the motor moves.
2.4. The slotted disc frequency is used to calculate the velocity by the microcontroller.
2.5. The SDP8600 sensor outputs the readings at a frequency of 1000Hz to the microcontroller to achieve 1000 samples per second.
2.6. The OPB705 line detecting sensor is installed on the car and connected to the microcontroller.
2.7. The OPB705 line detecting sensor is used to detect the reflective line on the track when the car navigates across it.
2.8. The OPB705 sensor output is sent to the PIC for recording the number of times the car navigated around the track.
2.9. The recorded data is stored sequentially in the SRAM on the circuit board.
2.10. The sequentially storage is generate by two 12 stage binary counter connected between the SRAM and the microcontroller

2.11. After navigation is completed, the recorded data can be extracted from the SRAM memory to the PC via an infra-red transmitter connected between the PC and the serial port of the microcontroller.
2.12. The track information is gathered as types of pieces and configured on the PC using C programming code.
2.13. The coded track configuration can be uploaded to the car via an infra-red transmitter connected between the PC and the serial port of the microcontroller on the car.

3. Power Supply Requirements:

3.1 - The power supply (12 Volt, 2A) provides power to the track and from the track will provide power to the microcontroller, the velocity sensor and the car motor. The dimensions of the track components and the electrical consumption values are fixed.

3.1.1 - The car will get the power from the track lanes by the braids. Nothing can be modified on the track components. Except the start strip in the track, no more strips can be used for the sensing devices. The car should be able to move in the external or internal path. Nothing can be added either to prevent planning or help the car in its way.

3.1.2 - The 12V power supply from the track will be connected to the PIC Processor (through a 7805 voltage regulator on disposal), the SDP8600 Velocity Sensor (through the 7805 regulator), and the motor C8146-model which is connected through a LG203 DMOS bridge driver (directly from the 12V power supply).

3.1.3 - The 7805 voltage regulator (Figure 1) reduces the 12V power supply from the track to 5V voltage necessary for the control system and the motor, and it has to be connected as following:

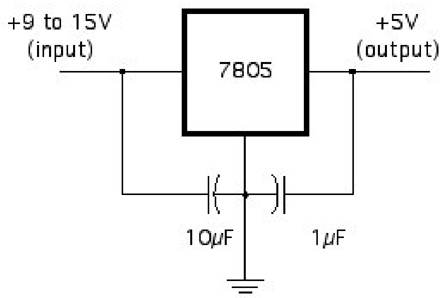


Figure 2 – Example of 7808 Voltage Regulator Configuration

3.1.4 – Capacitors are required in the input of 7805 because regulator is located an appreciable distance from power supply; another in the output is required to a better transient response (values specified by manufacturer).

3.2 - The PIC processor will switch the power to the motor, providing high\low voltage levels in its control outputs (by programming) which will be connected to the L6203 Driver (ENABLE, IN1 and IN2 inputs).

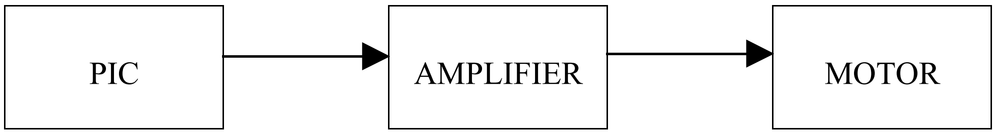
3.3 - No additional power can be used.

4. Guidance:

4.1 The circuit is formed of straight track and curved track. The straight racks come into 3 different sizes full length 350 mm half length 170 mm and the quarter length 87 mm.
The curves come in the form of 22.5^0 45^0 or 90^0.
The track will be measured depending on the number of tracks forming the circuit indicating where a curve start and where it finishes, these will be mapped and stored inside the PIC, then the speed is applied according to coming corner, a model will be developed later to show the relationship between the curves and the velocity.

4.2 The motor will be controlled by the DMOS full bridge driver L6201 which has two inputs, each one of them will receive a PWM signal which is supplied to the motor to give it speed and direction. Input 1 and input2 have different direction. This will be used to break the car.
The Pulse Width Modulator will be generated by the microcontroller which has a built in PWM (see page 57 [2] for details on implementation.

4.3 The PWM signal can be varied from 0% to 100% duty cycle.
The signal coming from the PIC has to be amplified before applied to the motor.



5 <u>Construction requirements:</u>

5.1 The Electronic components will be utilised using wire wrap method

5.2 Most of the components were to be mounted inside the car because it minimise the effect of centre of gravity and few components were to be mounted on top of the car as inside space is not enough to mount all the components.

5.2.1 Car's Wheel base is 75mm and car's overall length is 132mm, therefore the size of the circuit chooses to be smaller than that to fit inside the car.

5.3 The Infra red port will be created using following parts :

- SEP8706 Algaas Infrared Emitting Diod
- SDP8600 Optoschmitt Detector
- 330 R Resistor

6 <u>Software requirements:</u>

6.1 Software will be written in ANSI C.

6.1.1 Full operation of the car is to be controlled by the software.

6.1.2 Operations like complete navigation and sending and receiving data with the computer will be handled by the software.

6.1.3 Final running software's size should be less than 256 bytes.

6.2 Software is to use integer arithmetic.

6.2.1 Length is to be calculated in integer numbers.

6.3 Off-line simulation should be developed to test the different functions in the software.

3.1.1. Requirements & Specifications Outcome:

The group did not know the significant of producing a requirement & specification documents, the first produced version was very vague, not clear and not specific. Group members were not sure what to write and why is so important. The group then produced a second version based on the supervisor's feedback and side reading; overall it was better than the previous one in some sections.
The group has realise the importance of that documents when the project has moved forward and different challenges start to face the group and some times going back to our produced documents did not give us sufficient direction. The group realises the importance of making a well thought out requirement documents, it eases the coming steps and make the group aware of the problems may arise in future and make decision and think about solution to a specific problem before it occurs.

3.2. Software Design:

Overview:

The software will be designed to control an autonomous scalextric car, it will be written in C and then compiled to hex file then loaded into the chip which is the PIC16f87.
Inputs:

- track info
- wheel sensor
- start line sensor
- reading the data from the RAM

Outputs:

- PWM output
- Saving data to RAM
- Transferring data to the PIC

Functionality:

The car can start in any position on the track; the PWM will be set to drive the car to a constant steady speed as soon as the car crosses the white strip “start line” the internal timer will be reset and the car will go for full speed. PIC starts reading data coming from the wheel sensor and calculating the distance travelled from the start. This data will be compared with the data stored in a function called “TrackInfo” allowing us to determine the position of the car in that particular moment, and apply accordingly the appropriate control law.
The velocity of the car is workout by knowing the distance travelled and the time from the start of each lap then saved into the RAM.

Schematic Diagram:

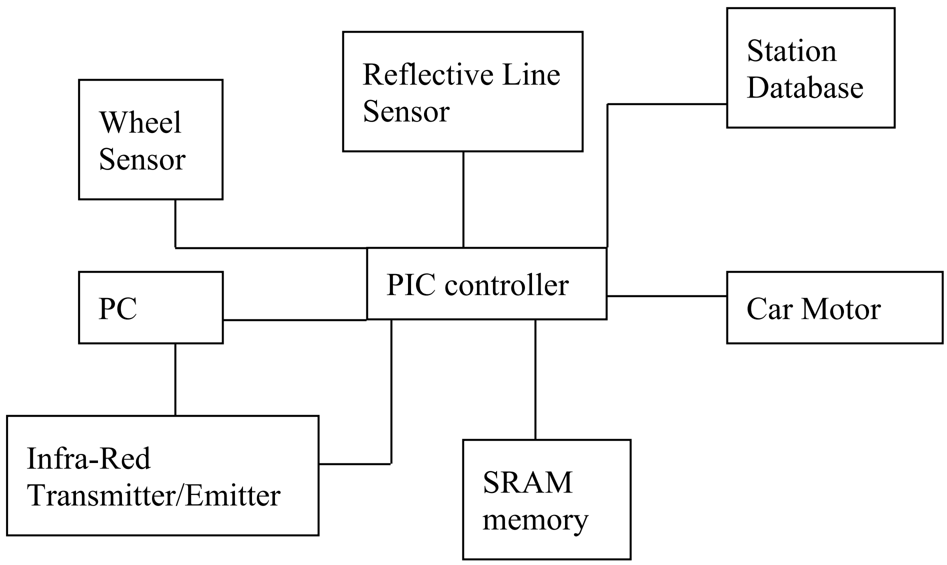


Figure 3 – Schematic Diagram

Sequence diagram:

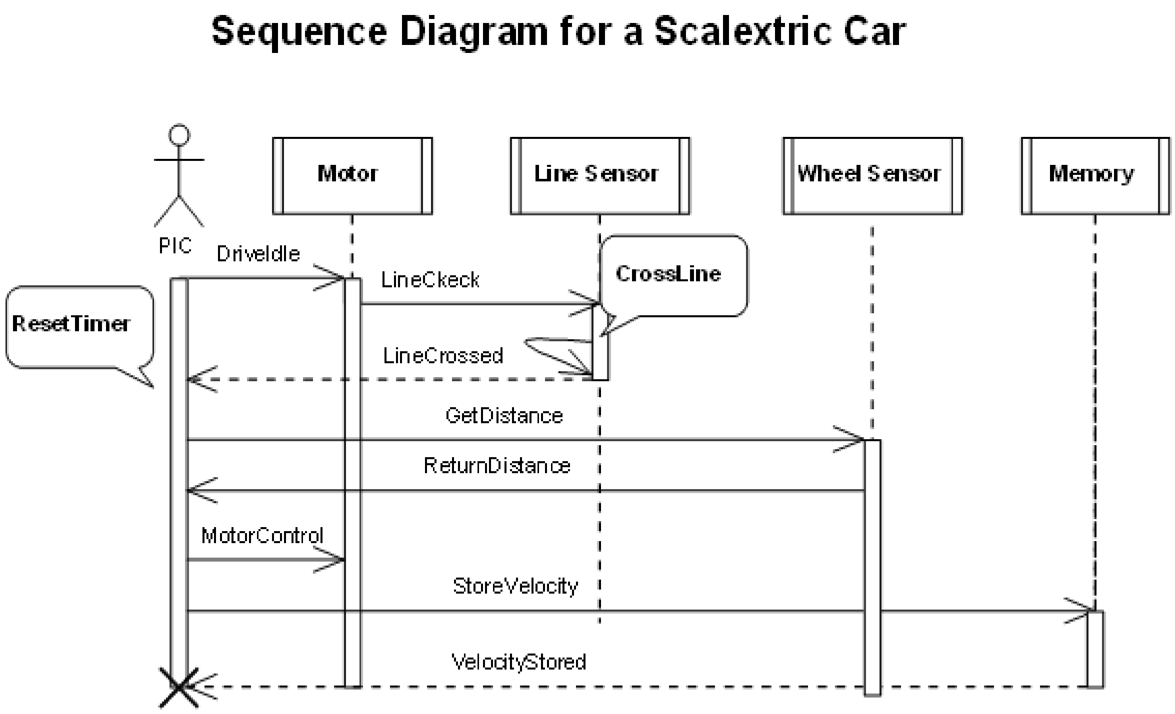


Figure 4 – Sequence Diagram

Use case Diagram:

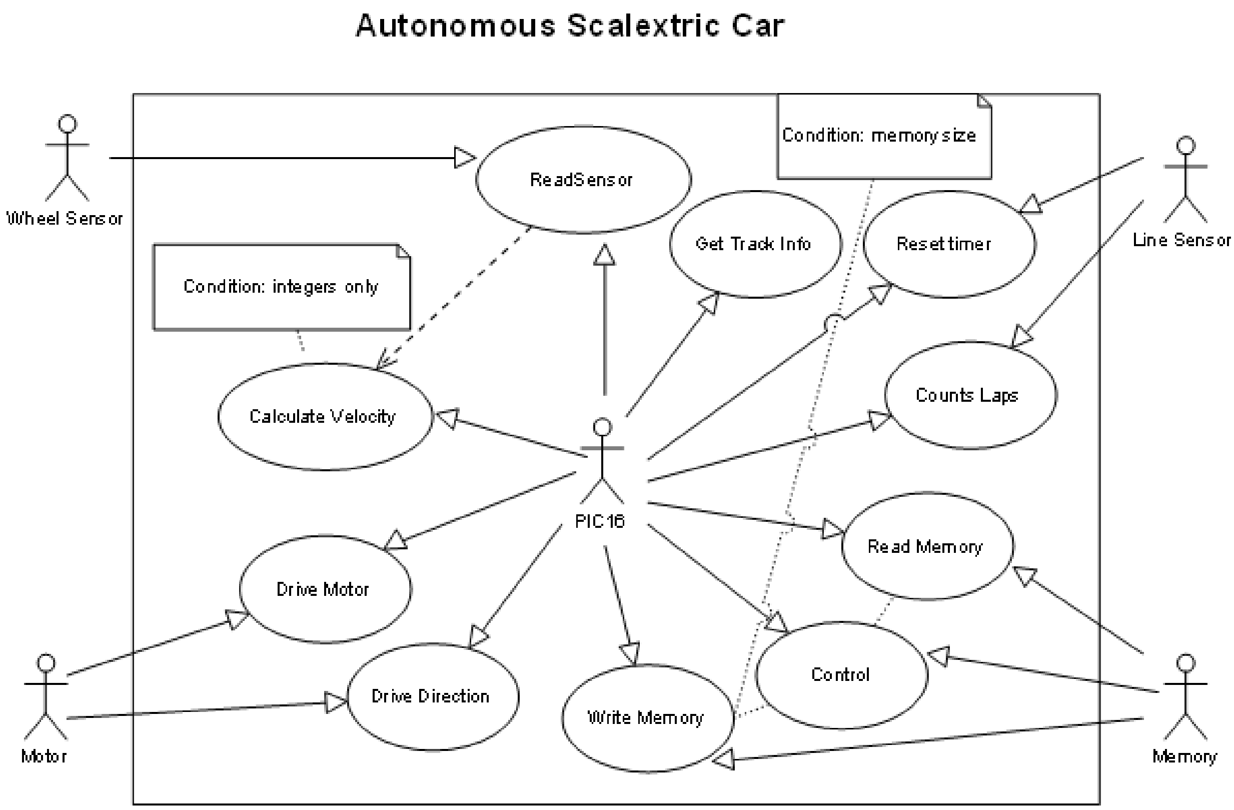


Figure 5 – Use case diagram

Context Diagram:

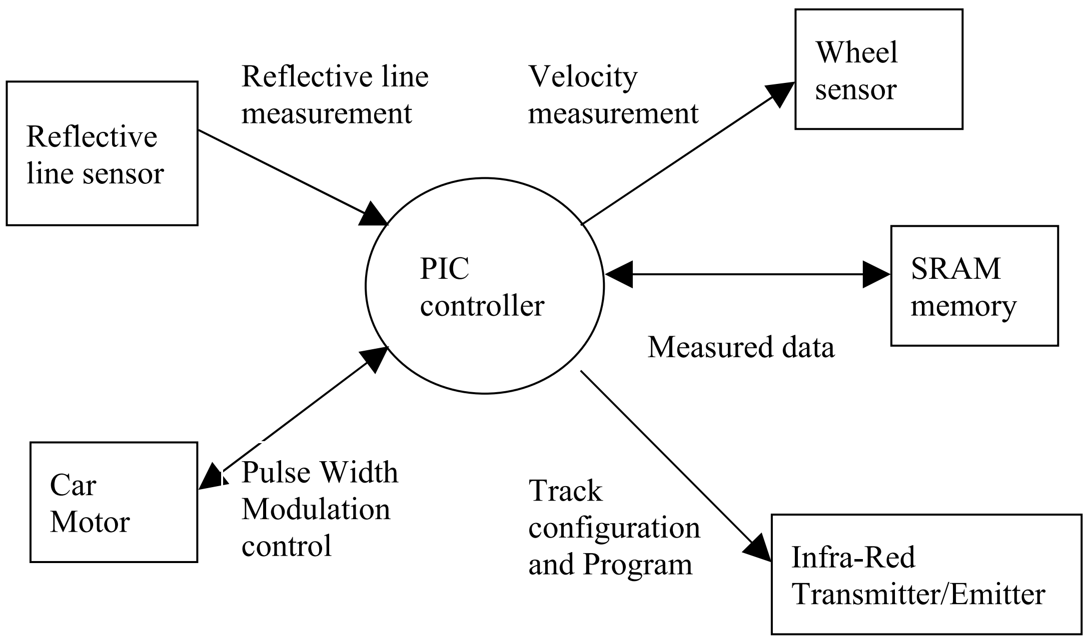


Figure 6 – Context diagram

Class Diagram:

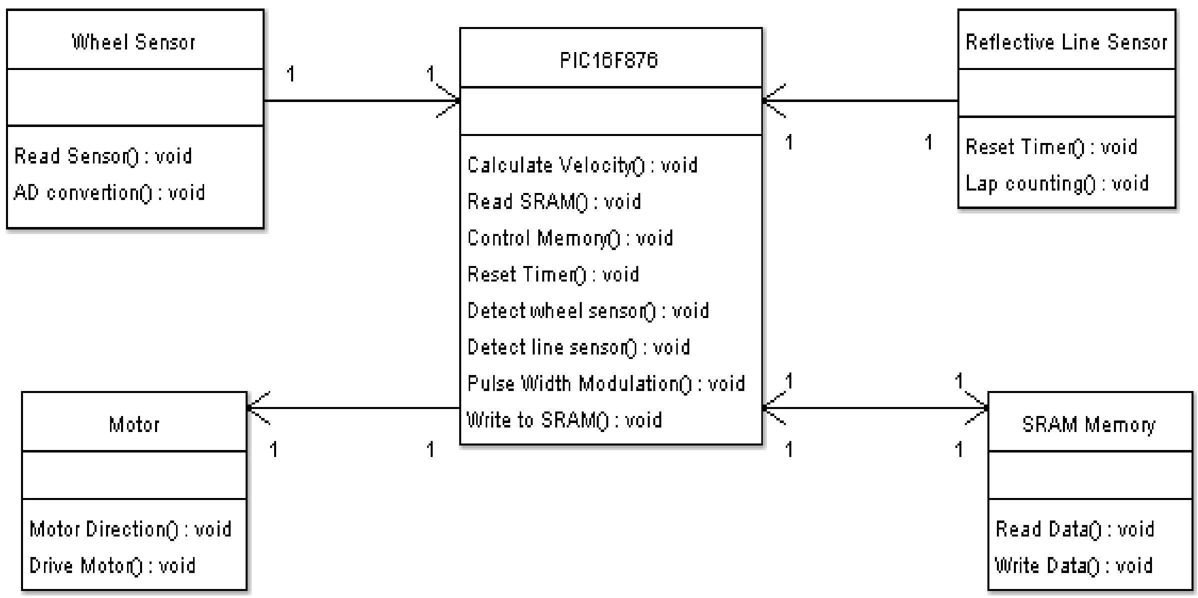


Figure 7 – Class diagram

3.2.1. Software Design Outcome:

In the software design, the main problem the group had was lack of knowledge and experience in designing software, so the group had to learn approaches on designing software. That was done by reading books and researches on the internet. The learning outcome was the ways of using the use case diagram, sequence diagram, context diagram, and class diagram and their interaction. Those designing methods were used to write the code.

3.3. Hardware Design:

Components Operation:

PIC16F876 Microcontroller:

This device yields the control on the 12Vdc-motor velocity through a Pulse Width Modulated signal (Pin CCP1) using a H-Bridge L6203 driver. The program will be loaded through and IrDA optical dock SDP8600 (from PORTC). Clock is configured with a 20Mhz crystal (OSC pins). Velocity is measured from a wheel sensor SDP8600 which is also connected to the microcontroller. Data is stored in a 128kx8bit SRAM BS62LV1027 which is connected to the microcontroller from PORTB (and PORTA/PORTC for the memory control pins). Also is connected to an opto-reflective OPB705 device through its RB0 (PORTB) pin. Its power is regulated by a 7805 12-to-5 voltage regulator. $\overline{MCLR}$/VPP pin is the voltage programming input and also can reset the device by a low level (active). See Appendix 1 for pin connection table.

Driver L6203 DMOS H-Bridge:

This device controls the motion of the motor by processing the PWM signal of microcontroller. The PIC processor will switch the power to the motor, providing high\low voltage levels in its control outputs (by programming) which will be connected to the L6203 Driver: with a high level in its ENABLE input, and a high (or low) level in both IN1 and IN2 inputs, the motor will be stopped. Then, with IN1/IN2 combinations you can switch on the motor turning right or left. With a low voltage level in the ENABLE input of the driver, the motor will be disconnected.

128Kx8bit CMOS SRAM BS62LV1027:

This SRAM memory stores the data from the PIC microcontroller. A0-A16 inputs are the address inputs to select one of the 132072 8-bit words (and it will be connected to the binary counters, see HCT4040). DQ0-DQ7 Input/Output ports are the bi-directional write/data ports of the memory (and it will be connected to the PIC, see PIC16F84). CE, WE and OE pins controls the write/read operations of the memory.

SDP8600 Devices for Wheel Sensor and IrDA Dock:

This component is connected between the 5v output of the 7805 voltage regulator and the ground with a LED emitter and a resistor, and its external pins are connected to the microcontroller (see PIC16G876 section).

7805 Voltage Regulator:

This device is connected from the 12V track power supply between its input and ground pins and yields a 5V voltage for all components required. This regulator needs two capacitors in its input/output and a protection diode in its input (see the circuit schematic design).

OPB705 Opto-Reflective Device:

This device is used like line sensor, and is connected between the 5V and voltage regulator supply and ground. The emitter of the internal transistor will be connected to RB0 pin of the microcontroller. This device needs two resistors in the diode anode and the transistor emitter.

HCT4040 12 Stage Binary Counters:

This counter addresses the memory locations and therefore will be connected as follows: the first counter U3 will be connected its Q1-Q11 pins to the A0-A11 address inputs of SRAM. U3-Q12 pin will be connected to the CLK clock pin of the second counter U4, which Q1-Q5 pins are connected to A12-A16 memory address inputs. The CLK clock input of the first counter U3 is connected to RC0 PORTC bi-directional input of the microcontroller and both of the reset-pins in the counters are connected to RC1 PORTC pin.

Basic Operation Block Diagram:

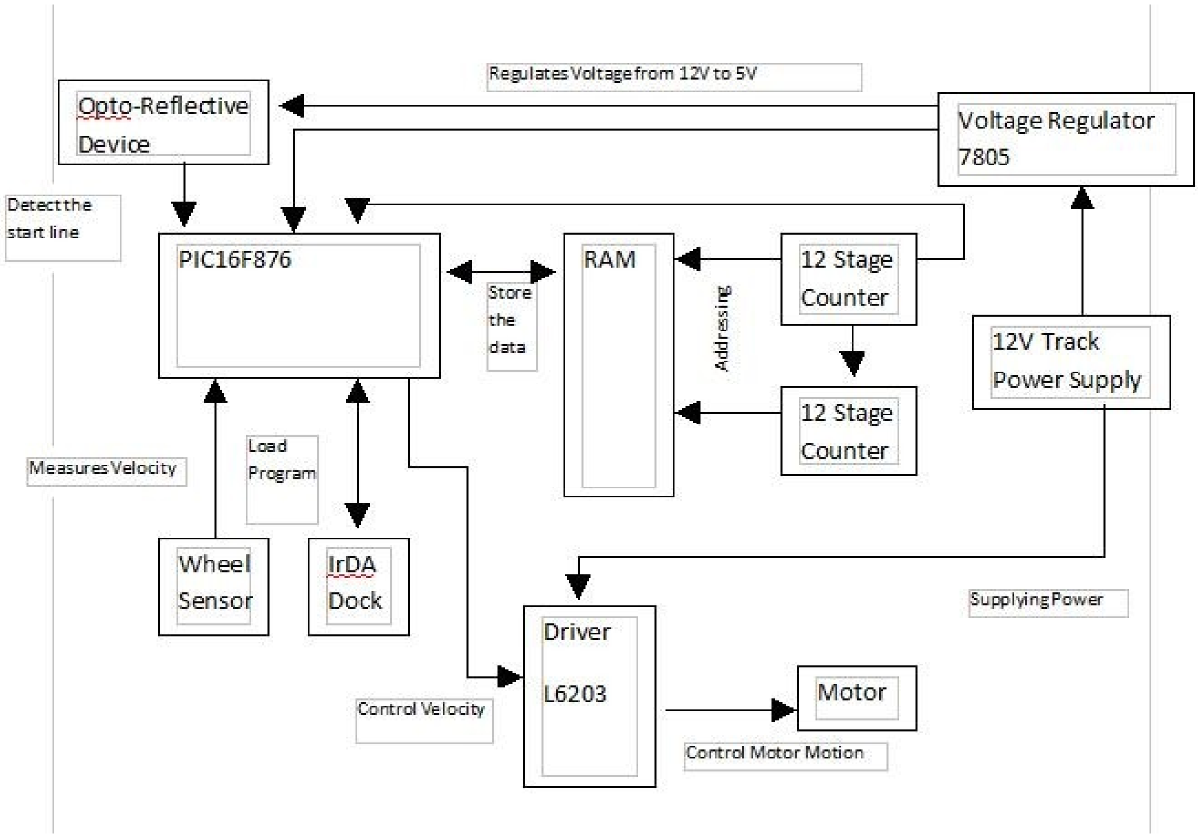


Figure 8 – Operation Block Diagram

Hardware Schematic:

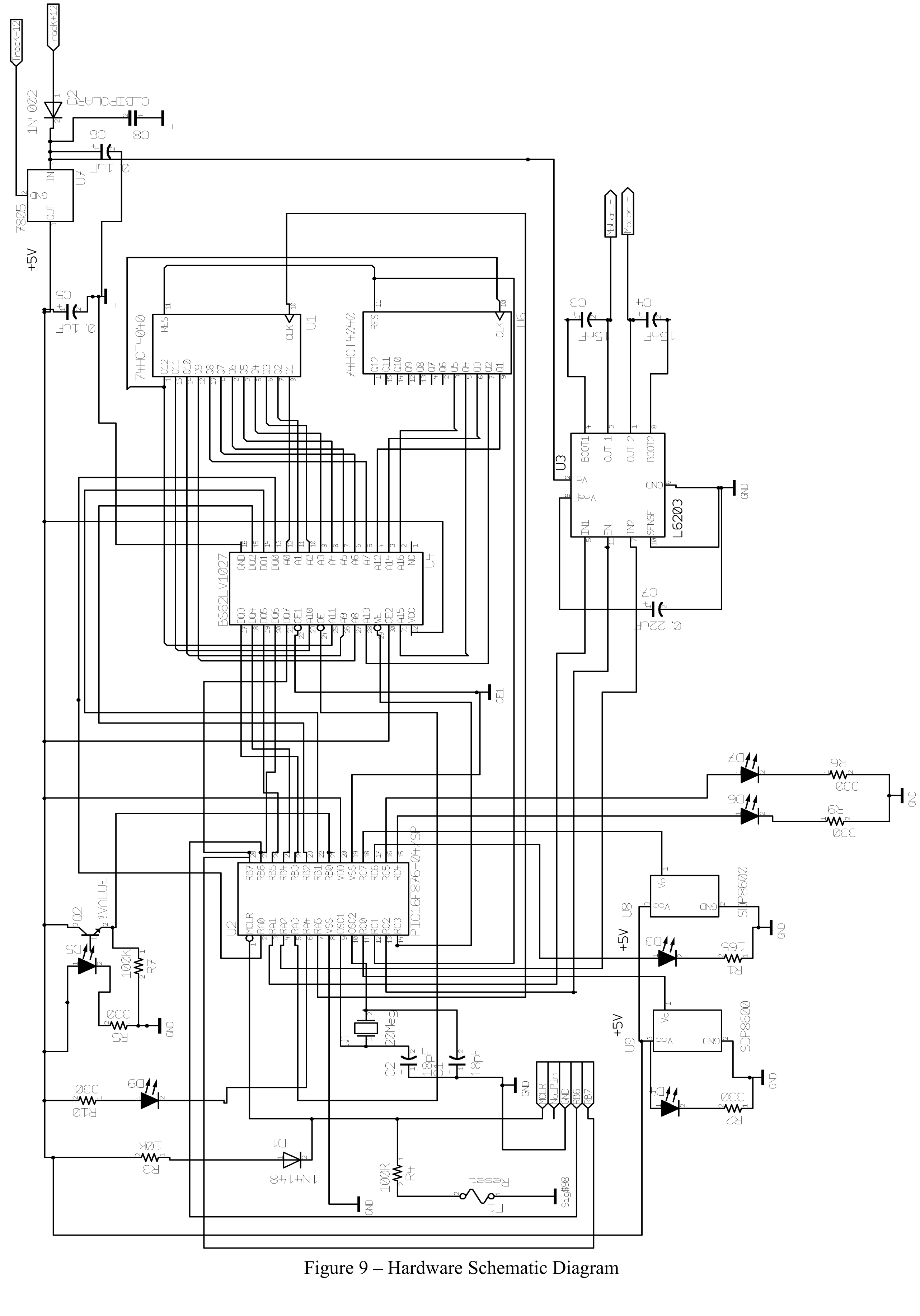


Figure 9 – Hardware Schematic Diagram

3.3.1. Circuit Construction:

According to the project planning, circuit construction was included into the simulation stage since it was not necessary to develop an electrical simulation of the circuit, because circuit can be tested with the help of a scope during the construction stage. This topic caused to spend the group some amount of time until the project supervisor feedback.

Circuit is being constructed using *Wire Wrapping* method. Wire Wrap is an alternative technique to PCB (printed circuit board) and is a good way for prototyping since it can be easily changed. It is a nice characteristic in a group project because allows other people can modify your work easily, if the wire-colours are respected. A manual wire wrapping tool and copper insulated wires of four colours were used in this project to build the circuit.

None of the group members have experience with this technique that is a little bit difficult to manage at the beginning because is a very delicate process to make a "perfect twist" with the wire around the pin, but after several attempts it can be managed more fast and efficient. It is also quite difficult to know with prior the length of the piece of wire that is required. Another problem is the use of the scope to see the waveforms in a efficient way by changing time/division and axis options. The sum of this things makes that the process of construct the circuit was a little bit exasperating.

Following with the Simulation Stage, circuit is being tested by introducing test codes in the chip and checking if the connections are well done using a scope. For example, connections between memory and counters was checked by introducing a test code that generates a squared wave on an output pin of the microcontroller; if connections with binary counters are well done, each pin of the counters must divide the frequency of the squared waveform. This waveform can be obtained using a while loop, as follows:

```
while(1)                          // Loop forever
  {
    Output_Low(Pin_A5);
    delay_ms(500);
    Output_High(Pin_A5);
    delay_ms(500);
  }
```

Since the test codes are provided, there are no problem between the circuit construction and the software coding parts of the project. But can arise problems if one member of the group try to test something in the circuit system and the code of colours of the wires is wrong.

3.3.2. Hardware Design Outcome:

In hardware design, the group had to learn about all the electronic parts which were used to create the circuit as there wasn't any expertise in the group in electronic background. Therefore, the group learnt operation, use and the connection of the new electronic components by reading data sheets and getting information from the internet. Also, one of the most challenging parts of this project was to create the schematic of the electronic components because none of the members in the group has done any hardware design before. First, problem which the group faced regarding this matter was deciding which software to use to layout the circuit and learning it. After reading and using a few number applications like OrCad, PCB 123 and IC design, the Group decide to use Target 3001 as it included most of the components which were used in the circuit.
Even though, there were few minor errors in hardware design like pin connection, the group manage to overcome those errors after demonstrator's help.

3.4. Modelling & Simulation:

The Car Motion Modelling:
General equation for the car motion is as follow:

$$M\frac{dv}{dt} = TF - \overbrace{\lambda_f m g}^{\text{Rolling resistance}} - \overbrace{\frac{1}{2} C_d A_f \rho_a v^2}^{\text{Aerodynamic drag}} \quad (1)$$

C_d : is the drag coefficient, which is about 0.35.
A_f : is the frontal area which is the surface facing the direction of movement which is around 0.05m x 0.05m = 0.0025 m^2.
v : is the speed of the car.
ρ_a : is the density of air, and under standard atmospheric conditions ρ_a =1.225kg/m^3.

The rolling resistance is given by: $\lambda_f m g$
λ : Rolling resistance coefficient, which is around 0.01?
g: = 9.81 m/s^2.
TF : traction force generated by the DC motor.

The aerodynamic drag: is really small and it can be ignored due to the size of the car (small) and the speed it travelling (not fast enough to generate a lot of drag)

Rolling resistance: the track resistance is related to the mass of the car and the gravity g.

$v \leq \sqrt{r.g.\frac{\cos(\beta_w)}{\cos(\beta_c)}}$ (1) This equation will be used to control the speed of the car. For more details refer to the individual reports.
This is the maximum speed the car can travel on the curve, and **r** represent the radius of the curve.

Therefore the maximum speed can the car travel into a corner is related to the two angles made by the centre of gravity and the rotational point, and most important the radius of the curve, the mass has no effect on the speed.

Motor Modelling:

In this section, the group assumed the motor in the car was a DC motor. Then they used equations (2) to build the model.

$$\frac{\theta(s)}{V(s)} = \frac{K}{s\cdot(Js+b)\cdot(Ls+R)} = \frac{K}{s\cdot(s+\frac{b}{J})\cdot(s+\frac{R}{L})\cdot JL} \quad (2)$$

Where: K = motor constant,
J = moment of inertia of motor
L = inductance of motor
R = resistance of motor
B = friction between the motor and load

Simulation in Simulink:

The torque is transformed to Traction force on the circumference of the car wheel, and is obtained by the following equation:

Traction force = $\tau .(1/rw)*G_r$

Where G_r is the gear ratio = 3

rw is the radius of the car wheel.

The car kinematics and the motor are put together to form the final model figure 9

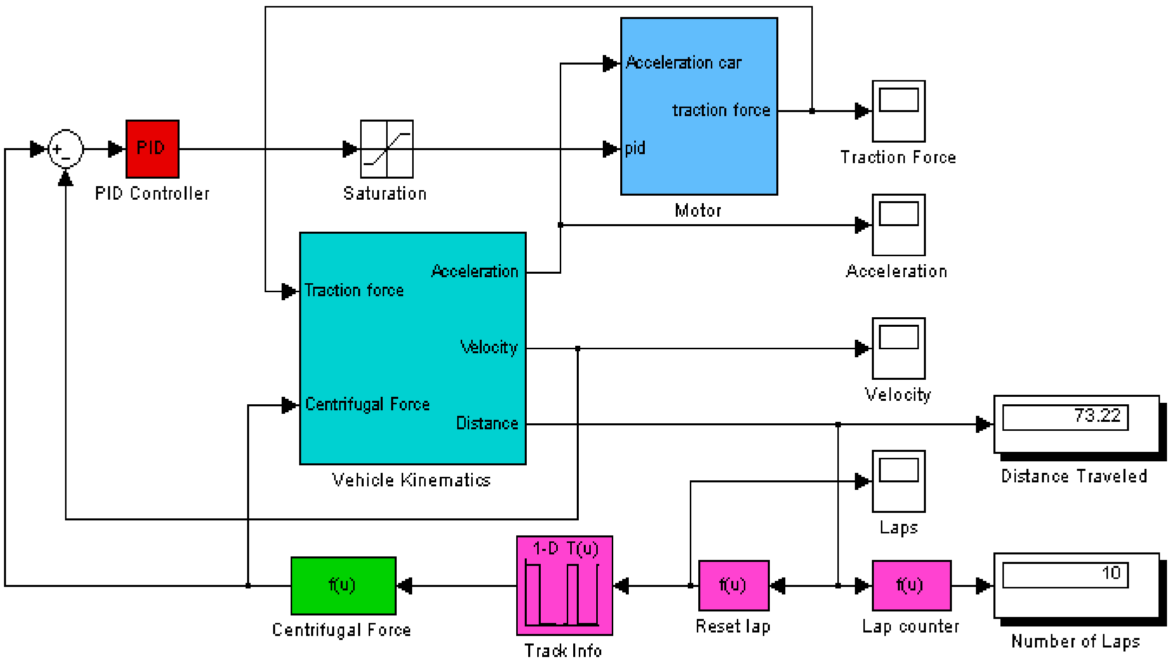


Figure 10 – Kinematic model of scalextric car

Where **The Centrifugal force** see equation (4) is fed by the appropriate **r** depending on the position of the car on the track this is done in the **Track Info**.
Track Info is a digital map of the track; it mapped the track by sectors. For each sector there is a radius **r** applied to it, for the strait line the radius is infinity so the speed applied is the maximum speed.
It is represented in matlab as two vectors one hold the distance for the sector the other one holds the radius for that sector, so the track info act as a lookup table reads the distance an then apply the relevant radius.
Reset Lap uses the **MOD** function in matlab to provide only the distance from zero to track length.
Lap Counter uses the **ceil** function to count the laps from the total distance.

For this particular track the total distance was 7.4*m*, so it will be partitioned into sectors as follow:

Sector 1: from the start to the beginning of the first corner is 1.4*m* the radius applied here is infinity (allow the car to carry maximum speed).
Sector 2: a corner has a radius of 0.37*m* for the length of 1.5 *m*.
Sector 3: a corner has a radius of 0.214 for the length of 0.86*m*.
Sector 4: a corner has a radius of 0.37*m* for the length of 0.46 *m*.

Sector 5: the strait line for the length of 1.05m maximum speed applied.
Sector 6: a corner has a radius of 0.37m for the length of 1.96 m.
Sector 7: the strait line for the length of 0.17m maximum speed applied
Then the new lap starts from sector1 and so on.

Control & optimization:

The speed of the car is set depending on its position on the track, using equation (1)

$$v \leq \sqrt{r.g.\frac{\cos(\beta_w)}{\cos(\beta_c)}}$$

β_w, β_c and g are constants, **r** is variable radius depending on which sector of the track the car is located.
The desired velocity **v** is **proportional** to **r**, for a very large **r** (driving in the strait) the demand for **v** is the maximum speed the motor can produce.

The **PID** is used to control the car by comparing the desired **v** with the actual speed.
The performance of the car is measured by the fastest lap. And the method used to tune the PID is the **Ziegler–Nichols** and tune by feel.

Best setting
P=12, I=0.3, D=2.1

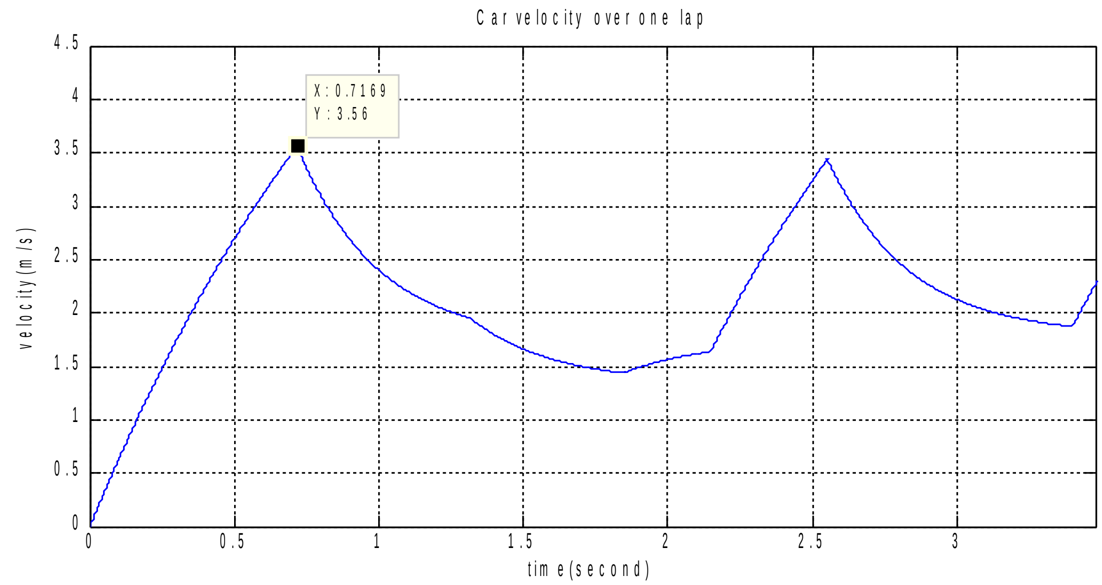


Figure 11

Lap time was 3.48 second.
Maximum speed 3.56 m/s in the strait
Minimum speed 1.43 m/s in the sharper turn
The response time to reach max speed was 0.71 second

3.4.1. Modelling & Simulation Outcome:

The group intention was to go from the hardware design to a computer aided design tool CAD, so the circuit can be built and tested and certified against the requirements.
However the CAD's available for the group did not have all the components needed in our design also it takes long time to find another CAD and familiarise with it.
The best solution was to build the circuit by part and test each part as it is built, so any errors can be spot earlier.

This option seemed more realistic and cheaper for us since components are cheap if any had gone wrong. The matlab simulation was used to optimize the control of the car. The complete model was composed of the motor model and the car kinematics model, the system is controlled using a PID controller tuned with Ziegler-Nichols tuning method.
The group had to adjust to the plan to add an extra week for the simulation time due to difficulties getting the parameters for the DC motor driving the car.
The simulation will used to validate the design, and as an off-line optimization tool rather than changing parameters in the microcontroller. The simulation results were close to the expectations.

3.5. Software Coding:

All operations of the car were managed by the software coding. PCW C compiler was used to write the code and learning PCW functions were another challenging part of this project. Different functions in the code were used to handle different operations.

<u>Pulse Width Modulation</u>

Navigation of the car was controlled by the Pulse Width Modulation (PWM) in the microcontroller. CCP1 pin has to set up using the coding to enable the PWM in the microcontroller. Code below shows how to set the CCP1 to PWM operations.

setup_ccp1(CCP_PWM);

Also, duty cycle of the PWM has to be set according to the speed which car needed to be travelled. A PWM output has a period and a time that the output stays high (duty cycle) as shown in *figure 4*. Code below shows how to set the PWM duty cycle.

set_pwm1_duty (value);

In the above code high period is given by the *value**(1/clock)*t2div. Where clock is oscillator frequency and t2div is the timer 2 prescaler.

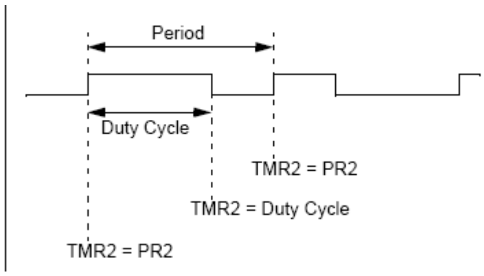


Figure 4

Correctness of the PWM code was tested using the oscilloscope.

<u>Velocity Sensor</u>

It is important to keep track of the car's velocity. Therefore, velocity was calculated using the slotted disc optical sensor (Wheel Sensor). Timer one in the microcontroller was used to count the number of pulses occurred due to wheel rotation in a one millisecond. One millisecond was generated by an interrupt. Code below shows the C function to read the value in the timer 1 (number of pulses generated by slotted disk). Moreover, refer to appendix 3 for software coding for interrupt.

get_timer1();

Furthermore, group realised that doing normal calculation caused a considerable floating point problem. Therefore useful way of avoiding this matter is being solved in the group.

Motor forward and motor reverse

Motor forward method drives the car forward and motor reverse method reverse the car. To drive the car forward Pin A1 of the microcontroller has to set High and Pin A2 has to set in to low. Likewise, to reverse the car Pin A2 and A1 of the microcontroller has to set High and Low respectively. For software coding for these functions were shown in software coding in appendix (Appendix 3).

3.5.1 Software Coding Outcome:

In software coding main problem which the group faced was to learn the PCW C compiler and its functions as its help was not detailed enough. Also, calculate the velocity using integer arithmetic and creating the interrupt were another two major problems.
However, after spending some time with understanding example codes written in the PCW compiler group manage to understand the basic operations of the compiler for some extent. After, following the example coding and getting help from the demonstrator, interrupt problem was solved. The group realised calculating velocity normally can give a wrong velocity and solution for that was being solved.

3.6. Coding and Testing

In this section, the group tested the microcontroller pin connection with the circuit board. This was done by writing a small function that produced a square wave on one pin and checks whether other pins will produce the wave. Fortunately, the microcontroller and the pin connected well. The group then test the SRAM memory on the circuit board. This was also done by writing two functions, read and write to the memory. The code for testing the pin, memory and the result of testing were shown in appendix 2.

4. Conclusion:

To sum up, the group had done work on 'software and hardware design and implementation', 'circuit construction and testing' and 'simulation and modelling the car dynamics'. By doing these works, the group had gained knowledge on group management; this includes time management and being a disciplined group member. Furthermore, members in the group had learnt skills on using applications to solve the technical problems like the MatLab, PCW C complier and Target 3001 (circuit design). The ability on writing technical information had also been enhanced and developed.

5. Future Plans to continue the Project

Up to the beginning of this semester, the group were one week behind on one task which is building the hardware. However, the group had made good progress on all the other tasks which are within the schedule. To continue the project, the group will have the regular meetings twice a week as planed. The tasks left are the following:

- The circuit board is ready. The only part left from the hardware is to install the rest of the components inside the car and connect them to the circuit board. This would take one week to finish.
- Implementation will start on the 25th February as planed. In this task the group will integrate parts of the software into the finished hardware.
- Software will be completed by 11th April.
- Testing and Optimization will start on the 14th April. This will last for two weeks.
- All the technical work will be finish by 25th April as planed.

6. Reference

- PIC16F876 microprocessor data sheet
- OPB705 reflective sensor data sheet
- SDP8600 opt-schmitt detector data sheet

7. Appendix

1.Pin connection table:

PIC PIN NAME	PIC PIN NO	OTHER END COMPONENT	OTHER END PIN NO	OTHER END PIN NAME	DISCRIPTION
OSC1/CLKIN	9	Crystal			Oscillator crystal input
OSC2/CLKOUT	10	Crystal			Oscillator crystal output
MCLR/VPP	1				Master Clear (Reset) input or programming voltage input. This pin is an active low RESET to the device.
RA0/AN0 RA1/AN1	2 3	SRAM L6203	15 5	DQ2 IN1	To store and retrieve data from the SRAM To the motor controller.
RA2/AN2/VREF-	4	L6203	7	IN2	To the motor controller.
RA3/AN3/VREF+ RA4/T0CKI RA5/SS/AN4	5 6 7	SRAM SRAM SPD8600(wheel sensor)	24 29 1	OE WE Vo	The output enable input is active LOW. If the output enable is active while the chip is selected and the write enable is inactive, data will be present on the DQ pins and they will be enabled. The DQ pins will be in the high impedance state when OE is inactive. The write enable input is active LOW and controls read and write operations. With the chip selected, when WE is HIGH and OE is LOW, output data will be present on the DQ pins; when WE is LOW, the data present on the DQ pins will be written into the selected memory location. Generate 1,0 sequence according to the speed of the vehicle

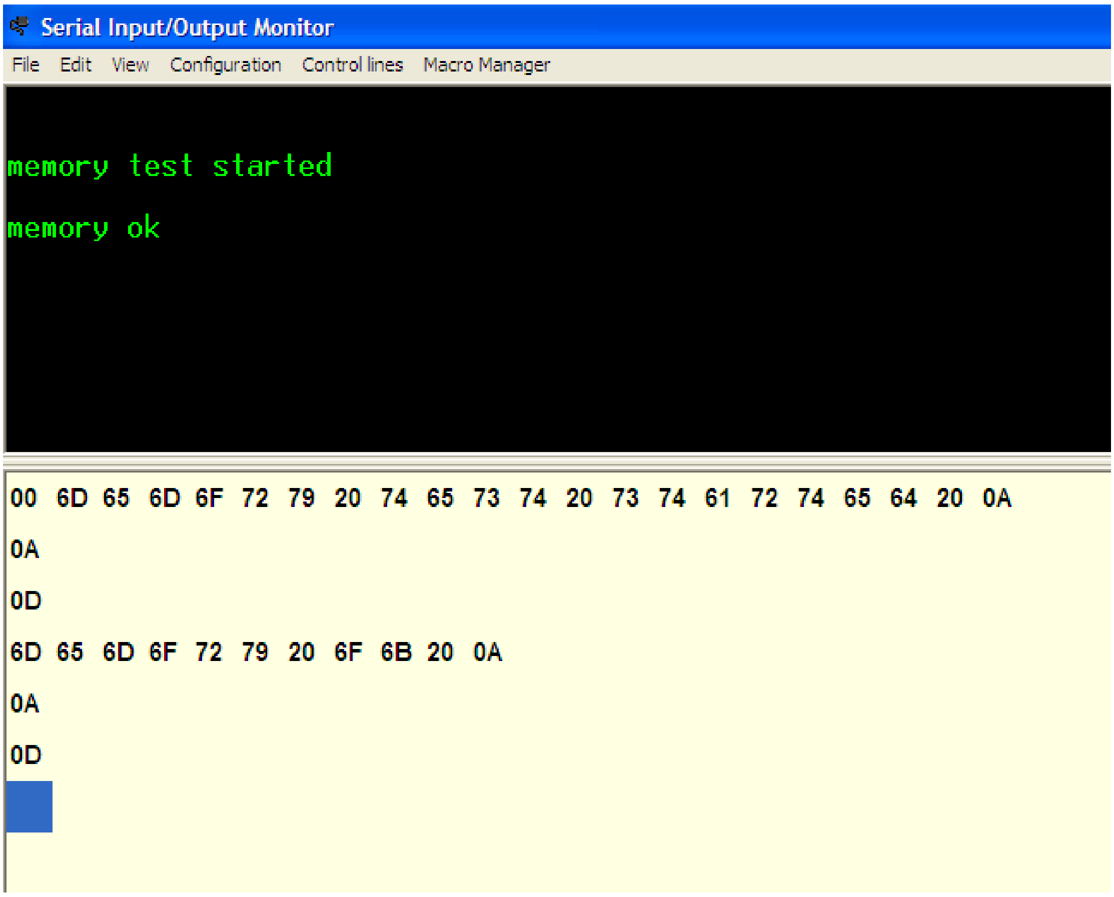
RB0/INT	21	OPB705			Lapcounter..
RB1	22	SRAM	14	DQ1	Data store load
RB2	23	SRAM	15	DQ2	Data store load
RB3/PGM	24	SRAM	17	DQ3	Data store load
RB4	25	SRAM	18	DQ4	Data store load
RB5	26	SRAM	19	DQ5	Data store load
RB6/PGC	27	SRAM	20	DQ6	Data store load
RB7/PGD	28	SRAM	21	DQ7	Data store load
RC0/T1OSO/T1CKI	11	HCT4040	10	CLK	Counter clock
RC1/T1OSI/CCP2	12	HCT4040(both)	11	RES	Counter reset
RC2/CCP1	13	L6203	11	EN	PWM out put
RC3/SCK/SCL	14				Test LED
RC4/SDI/SDA	15				Test LED
RC5/SDO	16				Test LED
RC6/TX/CK	17	SEP8706(Infra red)	1		Optical Docking Port
RC7/RX/DT	18	SDP8600(Infra red)	1		Optical Docking Port
VSS	8,9				Ground
VDD	20				+5V (Positive supply)

2. Code for testing memory:

```
#include <16F876.h>
#fuses HS,NOWDT,NOPROTECT,NOLVP,PUT,NOBROWNOUT
#use delay(clock=20000000)
#org 0x1F00,0x1FFF {}              // Reserve for Bootloader
#use fast_io(a)
#use rs232(baud=19200, xmit=pin_c6, rcv=pin_c7)
#define counter_clk Pin_A5
#define counter_reset Pin_C1
#define Read_Enable Pin_A3
#define Write_Enable Pin_C3

//declare function prototypes.
unsigned int ReadRam();
void WriteRam(int Data_write);

/* Global Variables */
unsigned int32 Ram_address_counter;

/* Main */

void main(void)
{
/* Config Ports */
unsigned int i;
Set_Tris_a(0b00000000);
Set_Tris_b(0b00000001);
Set_Tris_c(0b10000001);
```

```
/* Mainloop of Code */
output_High(counter_reset);
output_low(counter_reset);                    //force rest pin low
for (i=0; i<10; i++)
{
  WriteRam(i+0x41);
}
output_High(counter_reset);
output_low(counter_reset);
for (i=0; i<10; i++)
{
 putc(ReadRam());
}
while(1)                              // Loop forever
  {
}
}
/***********************************************************************/
                    /*                          Functions                         */
/***********************************************************************/
Output_low(Counter_reset);
Output_low(Counter_clk);
Output_High(Read_Enable);     //initialise pins to initial state high
Output_High(Write_Enable);

/*the read ram function*/

unsigned int ReadRam()
{
  unsigned int Data_Read;
  Set_Tris_b(0b11111111);    //set port B to input
  Set_Tris_a(0b00000001);    //set port A0 to input

  Output_Low(Read_Enable);   //enable the read operation
  Data_Read = ((Input_b() & 0b11111110 ) | (Input_a() & 0b00000001));

  Output_High(Read_Enable);  //disable the read operation
  Output_High(Counter_clk);  //advance Ram  address counter
  Output_Low(Counter_clk);
  return Data_Read;
}

/* Write to RAM function*/

void WriteRam(int Data_write)
{

  Set_Tris_b(0b00000001);                  //set port B to output
  Set_Tris_a(0b00000000);                  //set port A to output

  Output_b(Data_write);

  if (Data_write & 01)                 //checks if the LSB is equal to "0"
```

```
        {
          Output_High(Pin_A0);                  //sets port A0 to "0"
        }
      else
        {
          Output_Low(Pin_A0);                  //sets port A0 to "1"
        }
      Output_Low(Write_Enable);                  //enable the write operation
      Output_High(Write_Enable);

      Output_High(Counter_clk);                  //advance Ram  address counter
      Output_Low(Counter_clk);
      Ram_address_counter++;                  //increment the write counter
    }
```



2. Software coding for PWM interrupt and velocity

```
#include <16F876.h>
#fuses HS,NOWDT,NOPROTECT,NOLVP,PUT,NOBROWNOUT
#use delay(clock=20000000)
#org 0x1F00,0x1FFF {}              // Reserve for Bootloader
#use fast_io(a)
#use rs232 (baud=19200, xmit= pin_c6, rcv= pin_c7)
#define INTS_PER_SECOND 76      // (20000000/(4*256*256))
#define RED_LED PIN_C4  //red led
#define YELLOW_LED PIN_C5  //Yellow led
```

```
//Global variables
int holes;
BYTE seconds;      // A running seconds counter
BYTE int_count;    // Number of interrupts left before a second has elapsed
unsigned long test_timer=1000;
short test_flag=0;
//***********************************
//Interrupt function
//***********************************
#int_timer0
void sample_ISR()
{

set_timer0(217);  // = 256-39, 39(8bit value)= 20*10^6(crystal)/4/1000Hz/128(pre
scaler),
holes=get_timer1();

hole=holes;
set_timer1(0);

//************************************************
// Testing code for interrupt
/*
test_timer--;
  if(!test_timer)
  {
    test_timer=1000;
    test_flag^=1;
      if(test_flag)Output_High(RED_LED);
      else Output_Low(RED_LED);
  }
*/
//       Output_High(YELLOW_LED);
//      delay_us(100);
//    Output_Low(YELLOW_LED);
//****************************************
}


//#define motor_rev PIN_A1


/* prototypes */
void pwm(int corner_value);
int velocity(unsigned char weelsensor);
void motor_forward();
void motor_reverse();
```

```
/* Main */

void main(void)
{
/* Config Timer 0 and allow Interrupts */

/* Config Ports */

  enable_interrupts(INT_TIMER0); // set the timer 0 as an interrupt
  setup_timer_1(T1_EXTERNAL |T1_DIV_BY_1);  //set the timer 1 to velocity sensor
  setup_timer_0 (RTCC_INTERNAL | RTCC_DIV_128);
  enable_interrupts(GLOBAL);
  Set_Tris_a(0b00000000);
  putc('*');
 //  output_a(0xff);

/* Mainloop of Code */

  while(1)                          // Loop forever
  {
    Output_Low(Pin_A5);
    delay_ms(500);
    Output_High(Pin_A5);
    delay_ms(500);
    pwm(0);
  }
}

//*************************************************
//Motor Forward
//*************************************************

void motor_forward()
{

    output_high(PIN_A1); //check the data sheet again..
                   //because not sure which one is reverse and which one is
forward*******
    delay_ms(1);
    output_low (PIN_A2); // check the data sheet again..
                  //because not sure which one is reverse and which one is
forward********
}
//*************************************************
//Motor Reverse
//*************************************************


void motor_reverse()
{
    output_high(PIN_A2); // check the data sheet again..
                   //because not sure which one is reverse and which one is
forward*******
    delay_ms(1);
```

```
    output_low (PIN_A1); // check the data sheet again..
                      //because not sure which one is reverse and which one is
forward*******
}

//*************************************************
//PWM
//*************************************************

void pwm(int corner_value)
{
  int corner_value1;
  corner_value1=corner_value;


 // setup_ccp1(CCP_PWM);      // Configure CCP1 as a PWM
  //setup_timer_2 ( T2_DIV_BY_4,255, 1);   //have to set up this also to turn on the
pwm.

 // int value;
  switch (corner_value1)
  {

    case 0:

    motor_forward();
    set_pwm1_duty (100);   //where high period is given by value*(1/clock)*t2div
    break;

    case 1:

    motor_forward();
    set_pwm1_duty (50) ;  //where high period is given by value*(1/clock)*t2div
    break;
  }
}
//*************************************************
//Velocity
//*************************************************
int velocity(unsigned char weelsensor)
{
  int distance,velocity;

  //delay_ms(2);
  //enable_interrupts(INT_TIMER0);
  //holes=get_timer1();

   distance= 3*holes;//3=69/45*2 69= 2*pi*r
                 //as the value getting from the timer1 value is deviding by 2
                 //here we have to * by 2.(reduce the flooting point error),
                 //r=radius ***********loosing 0.5333 ????? check...
   velocity= distance; // distance/1 (1= 1ms)
  return velocity; /*velocity;*/
}
```