**Princess Sumaya University for Technology**

King Abdullah II Faculty of Engineering

Computer Engineering Department



Hand Controlled RC Vehicle

Microprocessors and Embedded Systems Project

**Supervisor**

Doctor Belal Sababha

|  |  |  |  |
| --- | --- | --- | --- |
| Name | ID | Major | Email |
| Omar Hussein | 20190221 | Networks and Information Security Engineering | oma20190221@std.psut.edu.jo |
| Victor Dawood | 20190616 | Electrical Power and Energy Engineering | vic20190616@std.psut.edu.jo |
| Badi Abu al Ghanam | 20190211 | Networks and Information Security Engineering | bad20190211@std.psut.edu.jo |

Contents

[1 Introduction 2](#_Toc125406125)

[2 Background 2](#_Toc125406126)

[2.1 DC Motor 3](#_Toc125406127)

[2.2 PIC16F877A 3](#_Toc125406128)

[2.3 H-Bridge 4](#_Toc125406129)

[2.4 Accelerometer Sensor 7](#_Toc125406130)

[2.5 Arduino UNO 9](#_Toc125406131)

[2.6 RF Sensor 10](#_Toc125406132)

[2.7 Batteries and voltage regulators. 10](#_Toc125406133)

[3 Design 12](#_Toc125406134)

[3.1 Hardware 12](#_Toc125406135)

[3.2 Software 15](#_Toc125406136)

[3.3 Flow Chart 27](#_Toc125406137)

[3.4 Results: 28](#_Toc125406138)

[4 Problems and Recommendations 28](#_Toc125406139)

[5 Conclusion 29](#_Toc125406140)

[6 References 29](#_Toc125406141)

**Table of Figures**

[Figure 1: Dc Motor 4](#_Toc125406142)

[Figure 2: PIC 16F877A Ports 5](#_Toc125406143)

[Figure 3: H-Bridge 6](#_Toc125406144)

[Figure 4: “L298” Dual H-Bridge Driver and its internal design 7](#_Toc125406145)

[Figure 5: The Axes for the accelerometer sensor 8](#_Toc125406146)

[Figure 6: MPU6050 accelerometer sensor 9](#_Toc125406147)

[Figure 7: Arduino UNO Ports 10](#_Toc125406148)

[Figure 8: RF Sensor, the right one is the transmitter and the left one is the receiver 11](#_Toc125406149)

[Figure 9: 12V Lithium Battery (3 Batteries 3.7V connected in series) 12](#_Toc125406150)

[Figure 10: 9V Battery 12](#_Toc125406151)

[Figure 11: Voltage Regulator (L780SCV) to convert from 9V – 5V. 12](#_Toc125406152)

[Figure 12: Proteus Hardware Design 14](#_Toc125406153)

[Figure 13: The Connection between the H-Bridge, the pic and the DC motors 15](#_Toc125406154)

[Figure 14: The connection between the accelerometer sensor and Arduino UNO 15](#_Toc125406155)

[Figure 15: The Overall Design, the movement of the sensor affects the car movement and direction 16](#_Toc125406156)

[Figure 16: Flow Chart 28](#_Toc125406157)

***Abstract***

In this project, the design process for a car whose movement is controlled by an accelerometer sensor placed away from it will be discussed. An Arduino UNO will be used in order to capture the values from the accelerometer and with the help of a RF sensor, data will be transmitted from Arduino UNO to a 16F877A pic microcontroller in order to receive data using the RF sensor whose transmitter will be on the Arduino UNO board and whose receiver will be on the pic so the car’s speed and direction can be controlled. A detailed explanation of each component and sensor used in this project while showing the interfacing between all of them will be done, also, the way data is transmitted between the Arduino and the pic using a RF sensor, and the car’s control process will be shown alongside their generated codes. The circuit design for the whole project using Proteus will be added with a supporting block diagram/flow chart to summary the whole process.

# Introduction

RC (Radio Controlled) Vehicles are used nowadays in game stores and applications in their wide and different ways of design and creativity. Sensors, signal processing and communication techniques, and the availability of coding nowadays play a huge role in developing compact systems that vastly contribute to our daily lives. Radio controlled vehicles and machinery are one of the most notable and useful systems that save us a lot of time and effort, the creativity of design and ideas can be a great factor in marketing these devices in game stores which boosts someone’s business financially.

In this project, a way of establishing a Hand controlled RC vehicle will be designed in order to control the movement and speed of it by someone’s hand moving relying on the hand’s direction and tilt without the need for wiring connection. After interfacing an accelerometer with an Arduino UNO placed on someone’s hand, the movement will become a main factor in controlling the car’s speed and direction which will be the main concentration and goal to be achieved.

The project will begin by connecting the inputs of a H-bridge with a 16F877A pic while the outputs of the H-bridge will be connected to the terminals of two DC motors that will move the car. Later on, the accelerometer sensor which will capture the tilt values will be interfaced with an Arduino UNO in order to store the tilt values so the RF sensor can transmit information to be received on the PIC from the Arduino UNO board. All these components will be explained in deep details in this report.

# Background

In order to understand the design and process for this project, it is a must to understand the background behind each component used:

## DC Motor

The demand of energy conversion systems and devices increased with the revolution of development and DC Machines have become widely used in many different industries which perform the conversion between electrical energy to mechanical energy and vice versa, and DC motors are considered one of the main machines which are used widely in different applications.

When DC voltage is applied from an external DC source or a battery, the current resulted moves in a wire and if that wire is adjusted inside an external magnetic field using permanent magnetics, a force will be applied.

There are two important parts of the DC Motor, the stator which is static and the rotor which is the part that moves, there are also two types of the windings, the armature windings where the voltage and polarities are being applied in the stator windings and also there are the field windings where the current flows inside the rotor windings. The coils are connected to a “Commutator” and the connections outside from the stator area are connected with “Brushes”. When DC current is applied, a magnetic field is formed inside the coils so the coil’s magnetic field interacts with the permanent magnets causing a force to be applied with a torque and rotation, and the motor is connected to a shaft as it rotates because the polarities are reversed as a result of the commutator keeping the current flow in the field rotor windings in the same direction, and that’s how the torque and the rotation movement are produced. [1]

In this project, two DC Motors will be used alongside the car’s wheels so they can rotate and move.

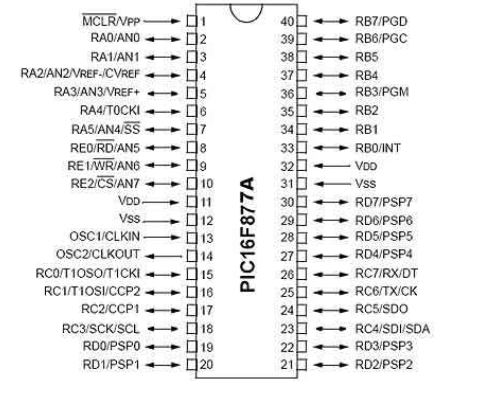


**Figure 1: Dc Motor**

## PIC16F877A

Microcontrollers play a huge role in embedded computer systems and digital electronic circuits nowadays, their flexibility and functionality allow them to be beneficial in performing certain applications, and PIC16F877A is considered one of the most used microcontrollers in current applications. This pic can be write-erase as many times as possible because of the usage of Flash Memory Technology, as shown in the figure below, it has 40 pins and 33 pins of them are for output and input. Another thing about this pic that makes it special is that it is so small and its cost is considered low, also, an EEPROM is featured inside this pic which allows it to store some of the information permanently. Moreover, it has a smaller 35 instructions set, it would operate up to 20Mhz frequency and the operating voltage would be between 4.2 Volts to 5.5 Volts but it should not cross that range in order to protect it from being damaged. On the other hand, it lacks an internal oscillator so an external one has to be brought instead. [2]

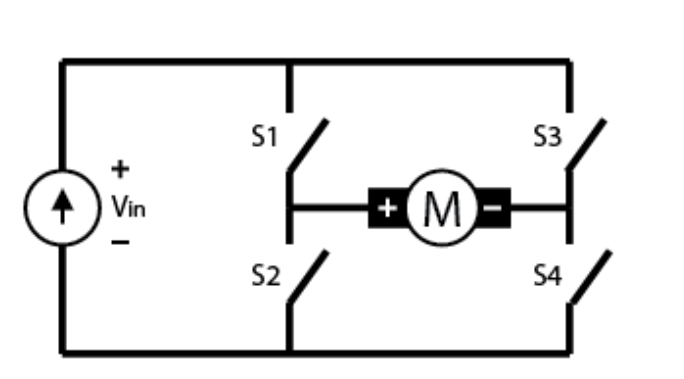
In this project, PIC16F877A will be used as a main component in order to be coded so, it can control the motor’s speed and direction after receiving data.

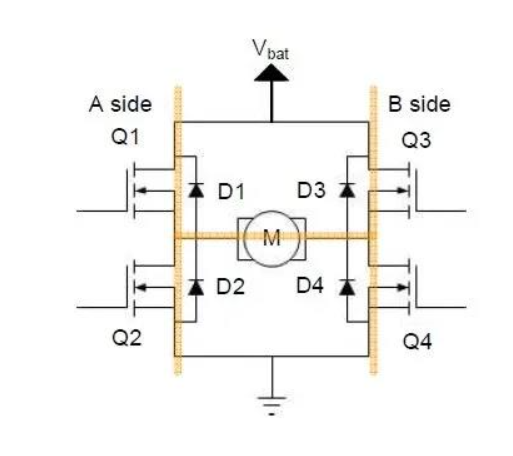


**Figure 2: PIC 16F877A Ports**

## H-Bridge

The H-Bridge is basically an electronic circuit which switches the polarities of a voltage applied on a load (which will be the DC motor in this project), it can be used in many other applications such as: robotics and power electronics converters and devices but it will be used in order to control the speed and direction of the two DC motor used in this project. As shown in the below figure, the H-Bridge has 4 switches, if the two switches (S1 and S4) close, the polarities around the DC motor would be (+ -) and it would move forward but if the other switches (S2 S3) close instead, the polarities would be reversed and they would become (- +) so the motor would move backward, closing other switches together (S1 S2 or S3 S4) should be forbidden since it would cause a short circuit and no current would flow through the motor and this might damage the bridge. The process to control the speed can be done by generating a PWM (Pulse-Width-Modulation) code which will be explained in details in the design section. The H-Bridge can actually contain certain electronic components such as: diodes or transistors (Mosfets) in order to function properly. [3]





**Figure 3: H-Bridge**

In this project, two DC motors will be used, so there is a need for a dual H-Bridge. The H-Bridge used in this project is called “L298” Dual H-Bridge Driver, it is important to know that in this bridge, there are two enable pins that should be connected to the pic used, also, it is not allowed to give two inputs for the same DC motor power (1) in order to avoid short circuit since current would flow through the closed path instead and this could damage the bridge. [4]

The L298 Dual H-Bridge is considered a high voltage and high current motor drive chip, 12V is required in order to let it function, it is basically a driver with 2 H-Bridges used to drive the two DC motors. [5]

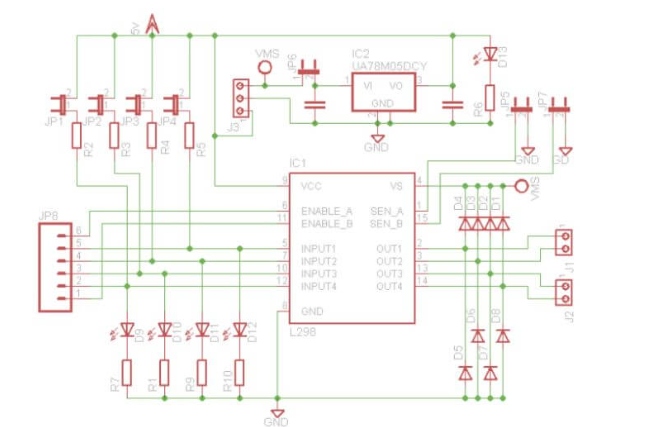
The table below shows the outcome direction for the DC motors based on which input is connected to VDD (5V) or to the ground.

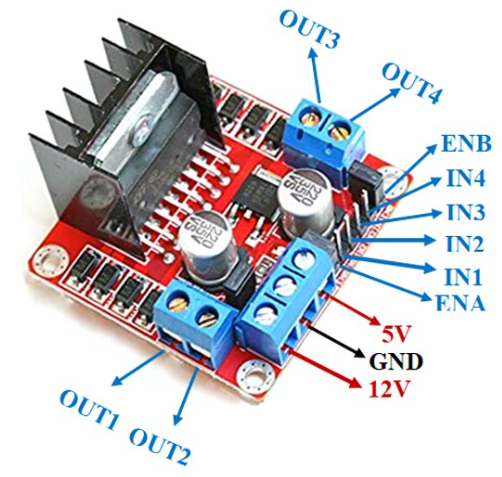
The first two inputs (input 1 and 2) are used for the first DC motor while the last two inputs (input 3 and 4) are used for the second DC motor.

**Table 1: H-Bridge Operation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input (4) | Input (3) | Input (2) | Input (1) | Outcome |
| 0 | 0 | 0 | 0 | Static |
| 0 | 1 | 0 | 1 | Forward |
| 1 | 0 | 1 | 0 | Back ground |
| 0 | 1 | 1 | 0 | Right |
| 1 | 0 | 0 | 1 | Left |
| 1 | 1 | 1 | 1 | It might cause a short circuit and destroy the bridge. |

The H-Bridge has pins for the outputs which can be connected to the terminals of the DC motor which will be explained in details in the design section.





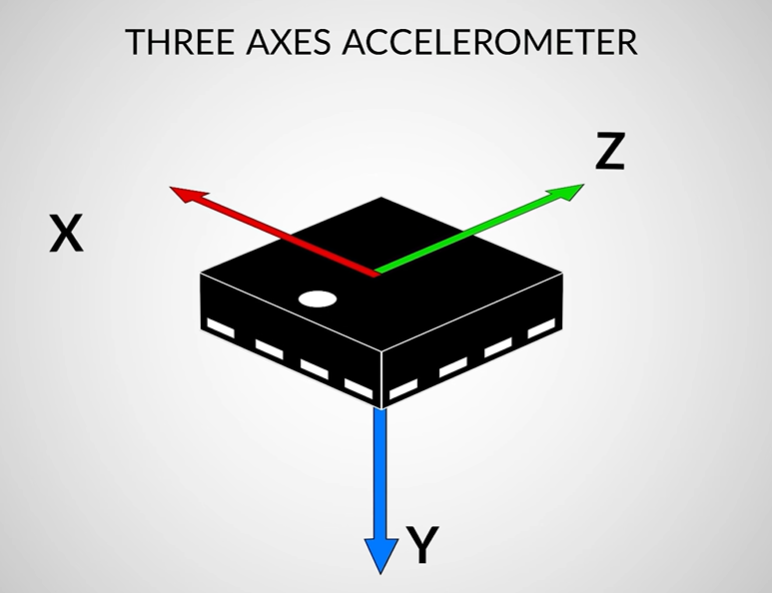
**Figure 4: “L298” Dual H-Bridge Driver and its internal design**

## Accelerometer Sensor

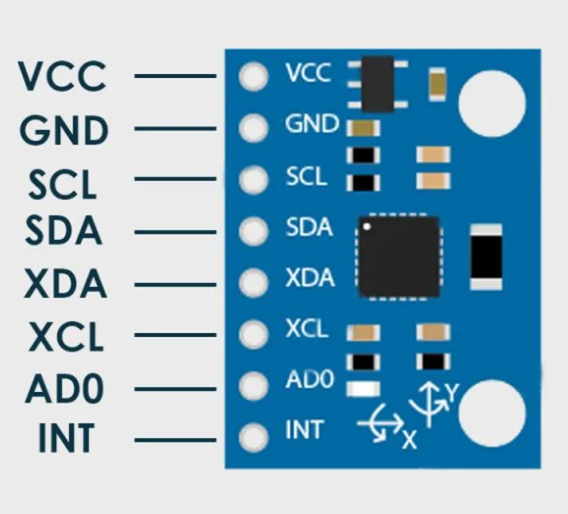
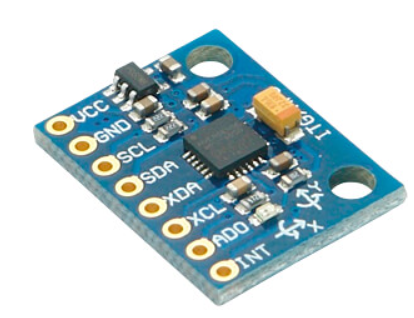
An accelerometer is a sensor that specializes in measuring linear and angular acceleration of devices as it has the ability to alter obtained physical acceleration from motion or gravity as it converts it to a voltage output. It is used in many different areas in certain applications such as detecting if a system is falling, monitoring several signals and controlling the movement of a device by transmitting the measured value to a speed value and also controlling the direction for a device as it will be applied in this project. An accelerometer can be used as a tilt sensor just like in this project; this can happen by detecting the orientation which is the tilt of the device by measuring the acceleration due to earth’s gravity which is a constant downward force acting on all objects. So basically, the sensor uses the gravity vector and its projection on the axes of it in order to measure the tilt angle. So basically, the accelerometer measures the static acceleration force when it is tilted and takes measurements and x-y-z planes. It also can measure the dynamic acceleration that helps detecting the change and the vibration that would occur.

The operation for the accelerometer sensor is being described as the movement of the micro-electrotechnical systems that is inside it which moves when the device is titled, the analog data is captured from inside it and proceeds into a digital data.

The accelerometer sensor used in this project is called “MPU6050”, it is basically a six-axis motion tracking device that measures a three-axis accelerometer and three axis gyroscope data as it provides motion data. [6]



**Figure 5: The Axes for the accelerometer sensor**

**Figure 6: MPU6050 accelerometer sensor**

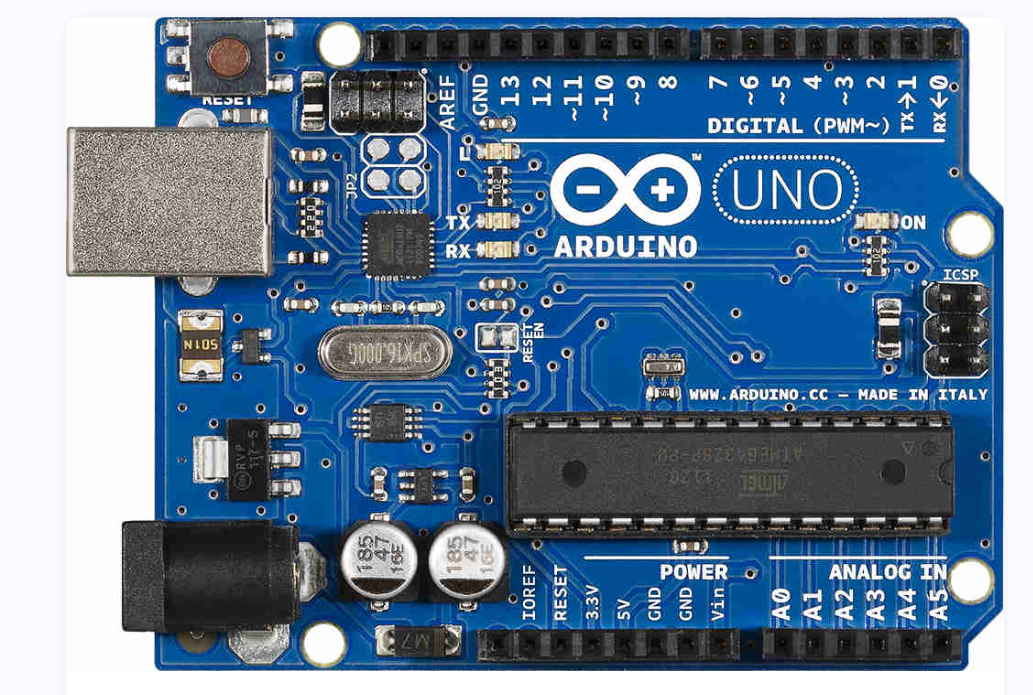
Brief description on each pin for the MPU6050 accelerometer sensor:

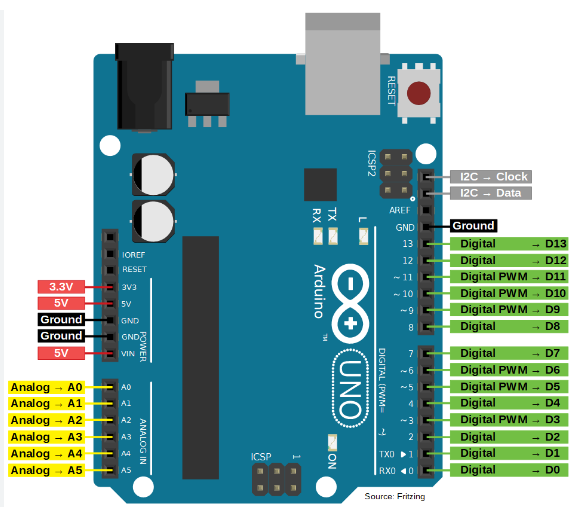
* VCC: Power (5V).
* GND: Ground.
* SDA: This pin is used for obtaining data serially from the sensor.
* SCL: This pin is used for serial clock input.
* XDA: SDA for configuring and reading from an external sensor.
* XCL: SCL for configuring and reading from external sensor.

## Arduino UNO

In this project, Arduino UNO which is a microcontroller board based will be used in order to interface with the accelerometer sensor; because of its huge qualities and characteristics that allow the interfacing to be more flexible. The code generated to interface Arduino UNO with the accelerometer sensor is much easier than the code generated for interfacing the same sensor with PIC16F877A.   
Arduino UNO contains 14 digital inputs and outputs, 6 are capable of PWM outputs, 6 analog inputs and a USB connection. Furthermore, its cost is considered to be very cheap compared to other microcontrollers. [7]

The pins A from the Arduino UNO board which are shown in the figure below are connected to the SDA and SCL pins from the accelerometer sensor because; the A pins are considered to be analog and the sensor will capture analog values for the tilt. [8]



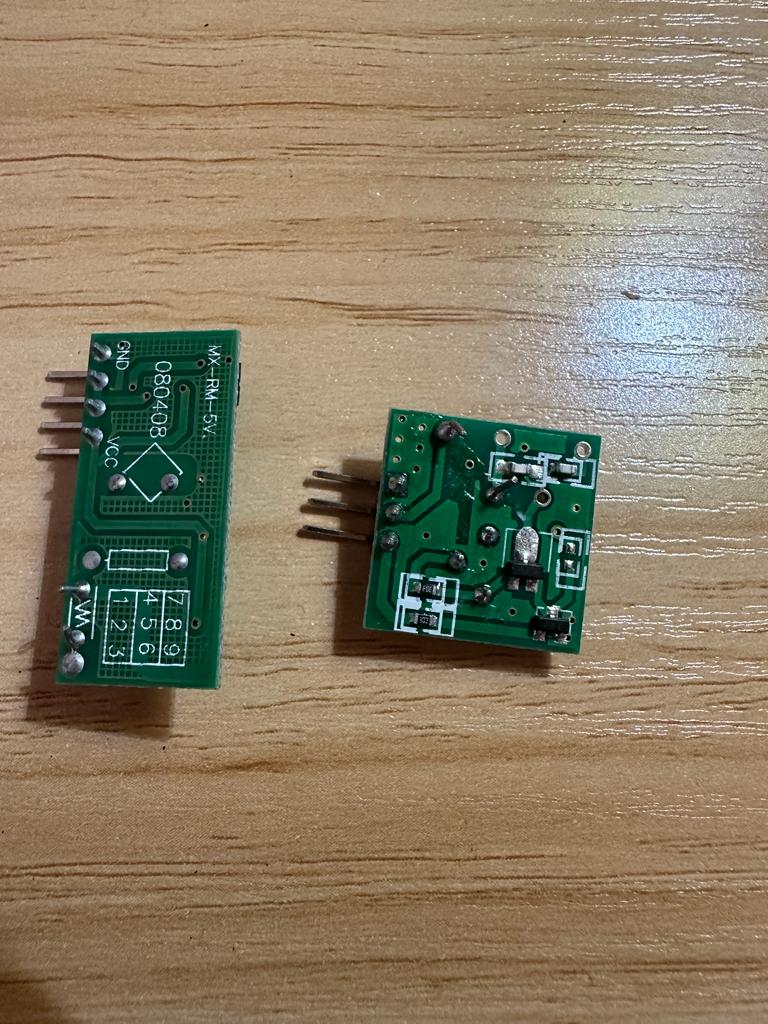


**Figure 7: Arduino UNO Ports**

## RF Sensor

Radio frequency sensors are a huge factor in modern day telecommunications and wireless signal transmitting (serial communication), they are crucially used in faster, long-distance ways of communications between two things. Their operation is represented by measuring signals by four diverse parameters which are the real and imaginary parts of electrical permittivity and magnetic susceptibility, if the real part of the permittivity or susceptibility becomes different, then the operating frequency of the sensor would definitely change but changing the imaginary part would affect the amplitude of the signal that is being transmitted. Furthermore, RF sensors operate at specific frequencies at which they have the biggest response, as their types vary in ranges of frequencies. Moreover, the size of these sensor plays a huge role in the improving the performance of the transmission. [9]

In this project, a RF sensor will be used in order to transmit data between the Arduino UNO board and PIC16F877A after capturing the tilt values from the accelerometer sensor, the transmitter would be on the Arduino UNO board and the receiver would be connected to one of the pic’s pins.



**Figure 8: RF Sensor, the right one is the transmitter and the left one is the receiver**

## Batteries and voltage regulators.

12V Lithium battery were used in this project in order to be connected with the H-Bridge, there was not a one-12V battery, so, three 3.7V batteries were added together in series so it can sum up to almost 12V in order to be used with the H-Bridge. 1 to 2 Voltage difference won’t affect but it should not cross 12V so the H-Bridge won’t be damaged.

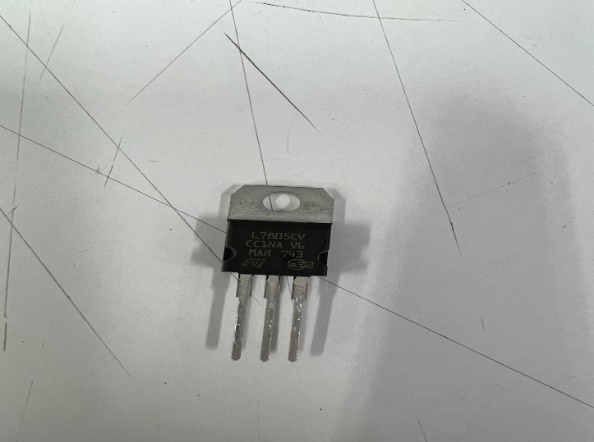
Furthermore, the 16F877A needs 5 Volts but there was not a 5V battery available so, 9 Volts battery was used with a voltage regulator in order to convert it to 5 Volts, this was also done for Arduino UNO so, two 9V batteries and two voltage regulators were used in this project.



**Figure 9: 12V Lithium Battery (3 Batteries 3.7V connected in series)**



**Figure 10: 9V Battery**



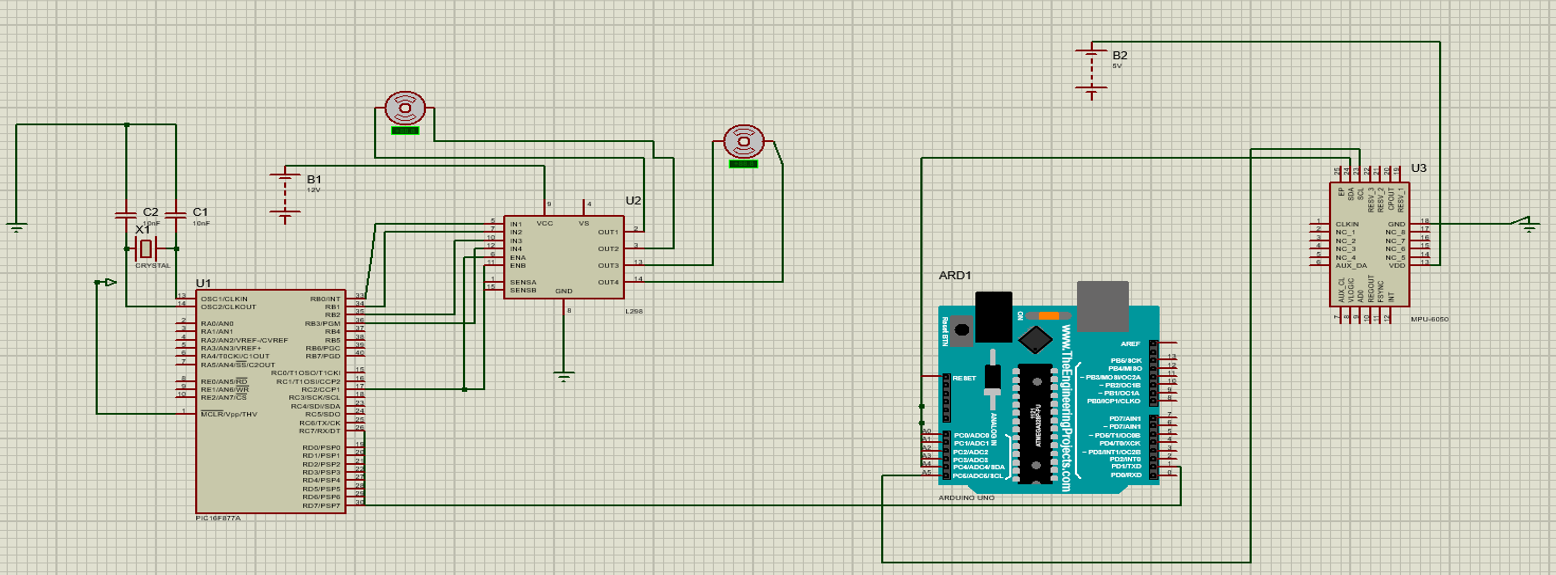
**Figure 11: Voltage Regulator (L780SCV) to convert from 9V – 5V.**

# Design

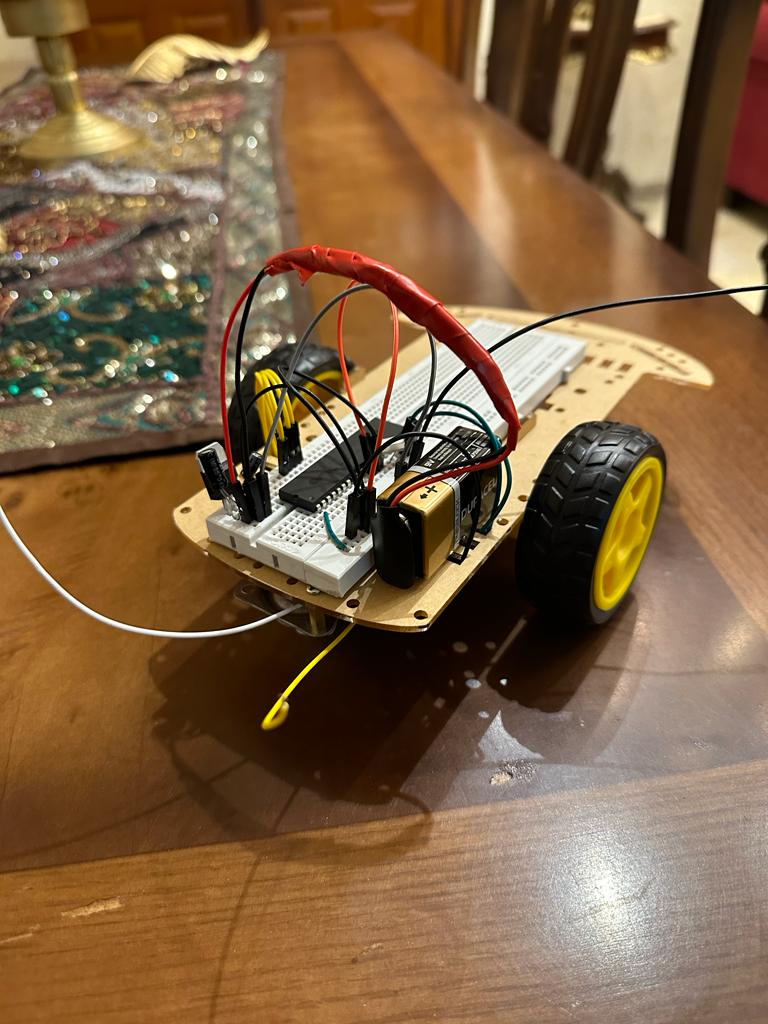
The design process for this project is:

## Hardware

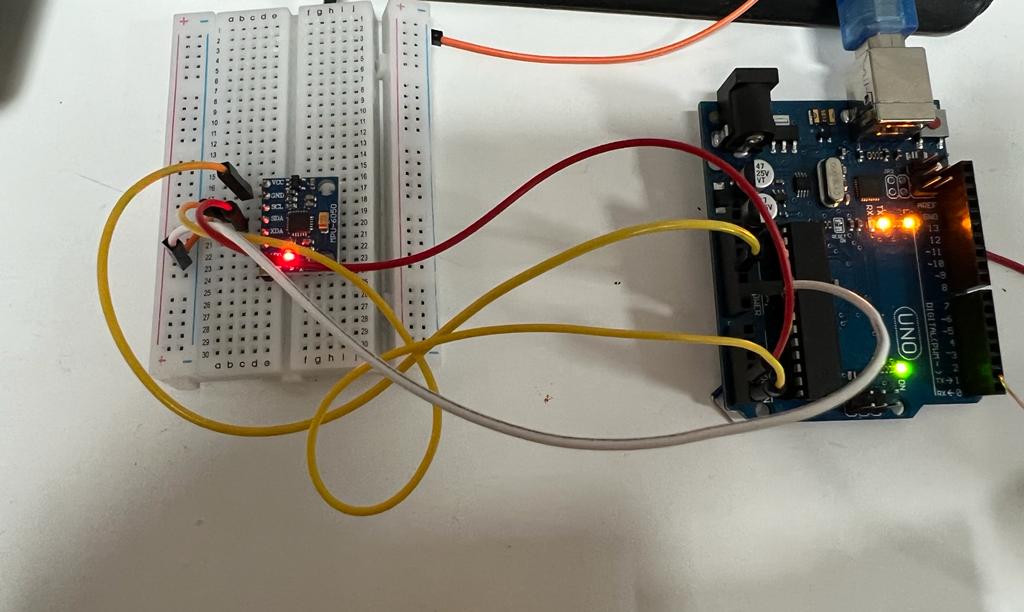
* After writing the code on the PIC16F877A microcontroller, using a breadboard, connect the input terminals of the H-Bridge to the Port B pins of the pic (RB0, RB1, RB2, RB3), also, connect the enable pins of the H-Bridge to RC2 pin of the pic.
* Connect the output terminals of the H-Bridge to the terminals of the two DC motors.
* Connect the 9V battery to the input terminal of the voltage regulator and connect the ground pin to a common ground node, then connect the output terminal of it to a common node (VDD) so, this node can be considered 5V in order to connect the (VDD) pin of the pic to it. Then, connect the ground pin to a common ground on the breadboard.
* Connect a wire between RC6 pin from the pic (as it is considered to be the receiver) and Transmitter pin of the Arduino, it can be replaced with the RF sensor after making sure it is working.
* Connect the 9V battery to the input terminal of the voltage regulator and connect the ground pin to a common ground node, then connect the output terminal of it to a common node (VDD) so, this node can be considered 5V in order to connect the (VDD) pin of the Arduino Uno board to it. Then, connect the ground pin to a common ground on the breadboard.
* Connect the pins [SCL and SDA] between the accelerometer sensor and the Arduino UNO board, so they can communicate with each other. SCL from the sensor to A5 on the Arduino, and SDA to A4 on the Arduino.
* Make sure to connect the voltage and ground pins of the accelerometer.



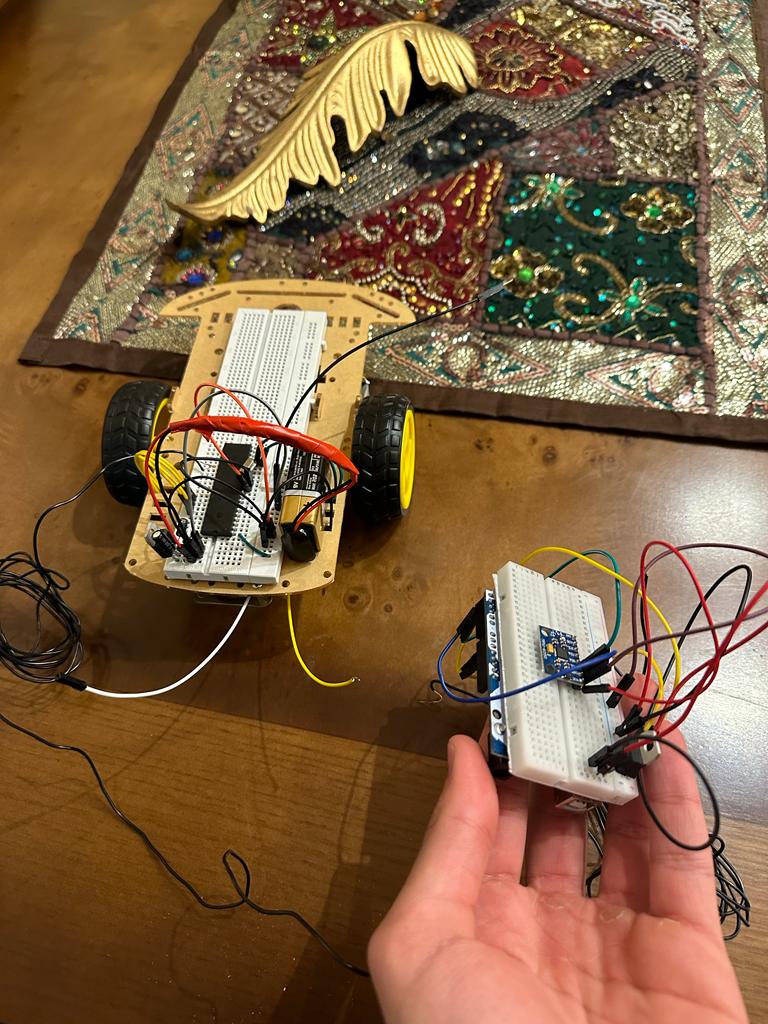
**Figure 12: Proteus Hardware Design**



**Figure 13: The Connection between the H-Bridge, the pic and the DC motors**



**Figure 14: The connection between the accelerometer sensor and Arduino UNO**



**Figure 15: The Overall Design, the movement of the sensor affects the car movement and direction**

YouTube Link for Project Demonstration: https://youtu.be/YEkX9SElL4M

## Software

In the software part of the code, we will have four concerns to address.

* MPU6050 Accelerometer Configuration
* Communication between the Arduino and the Accelerometer
* Communication between the Arduino and The PIC
* PWM Configuration (Speed control using the PWM Module in the PIC)

**Arduino Code**

The Arduino code has two functions. The Setup function, and the Loop function. In this Project, we will be using a dedicated library for the MPU6050 accelerometer sensor called Adafruit\_MPU6050 library. This library will address the two concerns of configuring the MPU6050 sensor and communicating with it. The communication between the Arduino and the accelerometer is done using the I2C Synchronous Serial Communication Protocol, which uses the two pins, SCL (Serial Clock) and SDA (Serial Data) for communication, and we can see those pins clearly on the MPU6050 sensor.

#include <Adafruit\_MPU6050.h>

#include <Adafruit\_Sensor.h>

#include <Wire.h>

Adafruit\_MPU6050 mpu; //Initializing an MPU Object

Before entering the functions, we include the needed libraries. The first two are for the MPU6050 sensor and the “Wire.h” library is for serial communication.

**Setup Function**

**void setup(void) {**

**Serial.begin(9600); //UART Serial Communication with PIC at 9600 Baud Rate**

**// Finding the MPU**

**if (!mpu.begin()) {**

**Serial.println("Failed to find MPU6050 chip");**

**while (1) {**

**delay(10);**

**}**

**}**

**//MPU Found!**

**// set accelerometer range to +-8G**

**mpu.setAccelerometerRange(MPU6050\_RANGE\_8\_G);**

**// set gyro range to +- 500 deg/s**

**mpu.setGyroRange(MPU6050\_RANGE\_500\_DEG);**

**// set filter bandwidth to 21 Hz**

**mpu.setFilterBandwidth(MPU6050\_BAND\_21\_HZ);**

**delay(100); //Delay for things to stabalize**

**}**

In the Arduino Setup code, we first begin by initializing the Serial Communication and setting up the right Baud Rate that we will be communicating with the PIC with. Then, we start by trying to find the MPU6050 Accelerometer sensor. If found, we then move on to initializing the sensitivity of the accelerometer, gyroscope, and temperature sensors in the MPU6050, as well as setting the digital low pass filter to filter out noise. Note that we are not interested in the accelerometer and the temperature values; however, we have to initialize them and keep reading data from them because the built functions in MPU6050 AdaFruit library does not have a dedicated function for each one of them, instead, it has functions that reads the whole data from the sensor.

Before entering the infinite loop, we define some bytes to be used for communication. These bytes, from their name, store the direction of movement that will later be sent to the PIC.

**Loop Function**

**\*Note that the gyroscope readings are stored in the accelerometer readings, that’s why we are dealing with a.acceleration.x and a.acceleration.y.**

**void loop() {**

**/\* Get new sensor events with the readings \*/**

**sensors\_event\_t a, g, temp;**

**mpu.getEvent(&a, &g, &temp);**

**// a.acceleration.x is the tilt around the X-axis (Left - Right)**

**// a.acceleration.y is the tilt around the Y-axis (Forward - Backward)**

**byte sendbyte; // Sensor Readings (bytes) that are going to be sent to the PIC**

**byte backward = 0xFF;**

**byte forward = 0xFE;**

**byte right = 0XFD;**

**byte left = 0XFC;**

**byte stop = 0XFB;**

**if(a.acceleration.x >= 5) // Go-to Right Threshold**

**{**

**Serial.write(right);**

**delay(500);**

**while(a.acceleration.x >= 5) // While we are tilting right and above the threshold**

**{**

**sendbyte=(a.acceleration.x - 2) \* 250 / 10; // Scale the sensor readings from 0 - 10 to 0 - 250 (the -2 is for lowering the initial speed)**

**Serial.print((a.acceleration.x)); // Just to display the readings in the Arduino IDE Serial Monitor to get an idea of what we are sending.**

**Serial.write(sendbyte); // Send the scaled reading to be used in the PWM Module on the PI**

**delay(2);**

**mpu.getEvent(&a, &g, &temp); // Keep reading sensor data while in this loop**

**}**

**}**

**if(a.acceleration.x <= -5) // Go-to Left Threshold**

**{**

**Serial.write(left);**

**delay(500);**

**while(a.acceleration.x <= -5) // While we are tilting left and above the threshold**

**{**

**sendbyte=(a.acceleration.x - 2) \* 250 / 10; // Scale the sensor readings from 0 - 10 to 0 - 250 (the -2 is for lowering the initial speed)**

**sendbyte=sendbyte\*-1; // Convert readings from negative to positive (The sign will only affect direction)**

**Serial.print((a.acceleration.x));**

**Serial.write(sendbyte); // Send the scaled reading to be used in the PWM Module on the PI**

**delay(2);**

**mpu.getEvent(&a, &g, &temp); // Keep reading sensor data while in this loop**

**}**

**}**

**if(a.acceleration.y >= 2) // Go-to Forward Threshold**

**{**

**Serial.write(forward); // Signal to the PIC that we are going forward**

**delay(500);**

**while(a.acceleration.y >= 2) // While we are tilting forward and above the threshold**

**{**

**sendbyte=(a.acceleration.y - 2) \* 250 / 10; // Scale the sensor readings from 0 - 10 to 0 - 250**

**Serial.print((a.acceleration.y));**

**Serial.write(sendbyte); // Send the scaled reading to be used in the PWM Module on the PIC**

**delay(2);**

**mpu.getEvent(&a, &g, &temp); // Keep reading sensor data while in this loop**

**}**

**}**

**if(a.acceleration.y < -2) // Go-to Backward Threshold**

**{**

**Serial.write(backward);**

**delay(500);**

**while(a.acceleration.y < -2) // While we are tilting backward and above the threshold**

**{**

**sendbyte=(a.acceleration.y + 2) \* 250 / 10; // Scale the sensor readings from 0 - 10 to 0 - 250**

**sendbyte=sendbyte \* -1; // Convert readings from negative to positive (The sign will only affect direction)**

**Serial.print((a.acceleration.y));**

**Serial.write(sendbyte); // Send the scaled reading to be used in the PWM Module on the PI**

**delay(2);**

**mpu.getEvent(&a, &g, &temp); // Keep reading sensor data while in this loop**

**}**

**}**

**Serial.write(stop); // Signal to the PIC that we are in Rest/Stop state. Speed is zero. We are not tilting to any direction.**

**delay(2);**

**}**

In the infinite loop, we have four ‘if’ statements. Each ‘if’ statement checks whether the tilt sensor is tilted in any of the four directions (forward ‘+y ‘, backward ‘-y’, right ‘+x’, left ‘-x’) and ensures that it’s above a certain threshold. This threshold is to eliminate the interference of one axis on the other, and to have a ‘safe spot’ to safely change directions by going back to the safe spot and changing the direction afterwards. Then, if the tilt sensor is tilted in any of the directions and above the threshold, we enter one of the four ‘if’ statements. Upon entering, we immediately send a dedicated byte to inform the PIC that we are tilted to a certain direction, and that the PIC must change the directions of the car accordingly. After sending the direction byte, we enter a while loop that traps us in it until we go back to the ‘safe spot’, meaning that we will stay in this while loop until we go back beyond the threshold of the current direction. Note that the gyroscope tilt readings are of a value between 0 and 10, thus, we need to scale them to a value that suits the PWM Duty Cycle; because we are going to use these readings to control the speed of the Car. The readings will be scaled to a value between 0 and 250 (Will be discussed later on in the PWM section). Something to keep in mind is that when we have negative readings, in the cases where we are tilted backwards ‘-y’ or to the left ‘-x’, we need to convert these readings to positive and scale them too; because the sign will not have any effect on the speed, it just affects the direction, and the direction is handled first thing when we enter one of the four ‘if’ statements. After scaling the readings properly and saving them in a byte called ‘sendbyte’, we then send these bytes serially using Serial.write(sendbyte). The bytes are now sent successfully to the pic, which will then interpret them and control the speed and direction of the car. On the last line of the while loop, we read the data from the sensor again; and that’s because we are stuck in an internal while loop, meaning that we will not iterate in the main infinite loop that already reads the data from sensor until we go back to the ‘safe spot’. If we didn’t read the data while we are in the internal while loop, we would get stuck in it with the reading that we had when we first entered this internal while loop. One thing to add, is that if we weren’t in any of the four direction ‘if’ statements, meaning that we were in the external infinite loop, which means that we are also in the ‘safe spot’, we will send a byte called stop, and this byte will be interpreted by the PIC so that it stops the car from moving while we are not tilted in any direction.

**Serial Communication between Arduino and the PIC**

Serial Communication is done using shift registers. When we want to transmit, we save the byte we want to send in a special function register, this register then copies the data down to a shift register where the data is sent serially to another shift register on the receiving end. After the whole byte is sent and received in the receiver shift register, the receivers shift register copies the data to another special function register, where the data is then dealt with and handled from there. The same principle is applied for the two communication protocols used in this project, the I2C and the UART.

The communication between the Arduino and the PIC was done using the UART Asynchronous Serial Communication Protocol.

* **Arduino to PIC**

In the Arduino, serial communication is handled fairly easily. We just initialize the serial communication with the right Baud Rate using Serial.Begin(Baud\_Rate). Then, we use the function Serial.write(data) to send data, and that’s it! With the help of the <Wire.h> library of course.

* **Receiving PIC**

In the PIC, there is a dedicated module and registers for serial communication. We will just use these to receive data, as we will not be transmitting anything from the PIC. To start, we initialize the serial communication module and registers by setting up the right values for them.

*void UART\_Init()*

*{*

*TRISC = TRISC | 0x80; // Rx/Rc7 Input*

*TRISC = TRISC & 0xBF; // Tx/RC6 Output*

*TXSTA=0x20; // Enable 8-bit Transmitter in Asynchronous Mode*

*RCSTA=0x90; // Enable Serial Port and 8-bit continuous receive*

*SPBRG = 12; // Low Speed 9600 Baud Rate with Fosc = 8Mghz*

*PIE1=PIE1|0X20; // Enable Receive Interrupt*

*INTCON=INTCON|0xC0; // Enable GIE and PIE*

*}*

In the UART\_Init() function, we set up the receiving and transmitting ports for the PIC. We enabled serial ports, which are used for receiving and transmitting data. Rc6 for transmitting and it was configured as output. Rc7 for receiving and it was configured as input. We set the communication to be 8bit data communication and the speed to be low speed with 9600 Baud Rate, by saving the value 12 in the SPBRG register according to the following equation.

In our project, we will be receiving data using an interrupt and not a dedicated UART receive function, thus we enabled the receive interrupt as well as the global and peripheral interrupts, to allow the receive flag to actually interrupt the program when data is received.

*unsigned char receivedbyte=0X00;*

*#define Fosc 8000000 // Oscillator Freq 8Mghz*

*#define TMR2PRESCALE 16*

*#define forward 0XFF*

*#define backward 0XFE*

*#define right 0XFD*

*#define left 0XFC*

*#define stop 0xFB*

*void interrupt()*

*{*

*if(PIR1&20) // Receive flag raised*

*{*

*receivedbyte=RCREG; // Read received data (byte)*

*PIR1=PIR1&0XDF; // Clear receive flag*

*}*

*}*

In the interrupt service routine, we check if the flag is raised, which it must be, otherwise we wouldn’t have entered the interrupt service routine. The reason behind this check is just for further updates, if for example we wanted to add more interrupt sources. After entering the service routine of the receive interrupt, we save the byte we received from the Arduino in a variable we declared before, called receivedbyte. Then, we clear the receive flag to not cause the program to get stuck, and to let the program to continue normally until another byte is received.

**Handling Received Data from Arduino**

After the data is received and saved in receivedbyte, it is now time to interpret this data and translate it to speed and direction. The interpretation is done in the main function. In the infinite loop, we have five ‘if’ statements and a PWM\_Duty function (Will be discussed in the PWM section). These five ‘if’ statements are entered only when a direction byte is received or a stop byte is received. Depending on the direction byte, we change the PORTB pins to set the right direction of movement, if the byte received is a stop bit, we clear PORTB pins and set PWM\_duty to zero - refer to the H-Bridge Table 1 above. PORTB pins are not changed until another direction or stop byte is received. Furthermore, the PWM\_duty function is called in all cases, since it is outside the ‘if’ statements.

**PWM Operation**

In order to use the PWM module we need two functions. a function to initialize the right values for the specific registers and another function to alter the duty cycle. [10]

* **PWM\_Initalize**

*void PWM\_Init()*

*{*

*/\*Fosc = 8Mghz, Finp = 2Mghz, Tinp = 0.5us, Tinc = 0.5us*

*With 1:16 Prescale, Tinc = 8us (TMR2 increments every 8us)\*/*

*CCP1CON = 0X0C; //Configure the CCP1 module for PWM operation*

*T2CON = 0x06; //TMR2 on with 1:16 Prescale*

*PR2 = 250; // 8us \* 250 = 2ms = PWM\_Period*

*TRISC = TRISC & 0xFB; // CCP1/RC2 Pin Output*

*}*

In our project, we used PWM with a period of 2ms. to do that we should set a proper value in PR2. The value we stored was 250 according to the following:

* We know that PWM is associated with the TMR2 module. So we will do all our calculations with respect to TMR2.
* TMR2 without pre-scale increments every 0.5us (With an 8Mghz Oscillator Frequency) keeping in mind that the clock source for TMR2 is the internal clock (Fosc / 4).
* Without pre-scale, TMR2 overflows every 128us, which is not enough for our 2ms period.
* That said, we set a pre-scale of 1:16 to TMR2. In this case, TMR2 increments every 8us and overflows every 2.048ms, which is enough in our case.
* For TMR2 to increment every 2ms, it needs exactly 250 count.
* In result, we need to store 250 in the PR2 register.

In addition to storing the period in the PR2 register, we also configured the CCP1 module to operate in PWM mode by setting the CCP1CON register to 0x0C. Also, we enabled TMR2 with 1:16 pre-scale by setting the T2CON register to 0x06. And ofcourse, since the PWM signal is generated using the CCP1 module, we set Rc2/CCP1 to be an output pin so we can connect the enables of the H-bridge to it, to alter its speed.

* **PWM duty function**

*void PWM\_Duty(unsigned char duty)*

*{*

*if(duty<=250) // Make sure the duty cycle is within the PWM\_Period*

*{*

*//We will be using only 8 bits for the duty cycle*

*//duty resolution = 8-bits*

*CCPR1L = duty;// Store the 8 bits in the CCPR1L Reg*

*}*

*}*

In this function, we store the number of TMR2 counts that we want to be On/Set out of the whole 250 counts that represent the period. That said, the duty value should be a number between 0 and 250. And since we have 5 more extra numbers to use out of the 255, we will be using those extra numbers for sending and receiving directions.

**Main Function**

*void main()*

*{*

*TRISB=0X00; // PORTB is going to connect to the input pins from the H-Bridge to control the direction of the motors*

*/\*Not important\*/ PORTC=0X00; // EXTRA: turning off output pins of portc just in case*

*PORTB=0X00;*

*UART\_Init(); //Initialize Serial Communication between PIC and Arduino*

*PWM\_Init(); //Initialize PWM Module to be used for Motors.*

*while(1)*

*{*

*/\*If receivedbyte was stop. stop the motor and clear the duty cycle.\*/*

*if(receivedbyte == stop)*

*{*

*PORTB=0x00; // Motors Stop.*

*PWM\_Duty(0); // Duty cycle = 0*

*}*

*/\*Determining the direction, then the speed\*/*

*if(receivedbyte == right)*

*{*

*PORTB= 0X06; // Motors in Differet Directions. Turn right.*

*}*

*if(receivedbyte == left)*

*{*

*PORTB=0X09; // Motors in Differet Directions. Turn left.*

*}*

*if(receivedbyte == forward)*

*{*

*PORTB=0X05; // Motors Both Forward.*

*}*

*if(receivedbyte == backward)*

*{*

*PORTB= 0X0A; // Motors Both Backward.*

*}*

*PWM\_Duty(receivedbyte); /\* After determining the direction,*

*the next received bytes will control the speed.\*/*

*/\*If receivedbyte was a direction byte, it will be of a value*

*greater than 250. That said, the PWM\_Duty function will not*

*accept it and will not change the CCPR1L Reg. Meaning that*

*direction bytes do not affect the duty cycle. Check the if*

*statement in the PWM\_Duty function.\*/*

*}*

*}*

Back to our infinite loop, we said that the PWM\_Duty function will be called in all cases. Normally that would cause a problem where sending direction and stop bytes will be interpreted as a speed change as well, and we don’t want that. For that reason, we set a checking condition in the PWM\_Duty function. that checks if the duty supplied is less than or equal 250. And we also defined that all the direction and stop bytes are above 250, thus, we are safe. Meaning that direction and stop bytes will not change the CCPR1L register resulting in so speed change.

Finally, if the bytes received were not direction or stop bytes, which means they are tilt angle readings, we then pass those values to the PWM\_Duty function in the infinite loop, which in turn controls the speed proportional to the tilt angle.

## Flow Chart



**Figure 16: Flow Chart**

## Results:

* When the sensor is being moved down, the car will go forward, the more the sensor goes down the faster the car becomes.
* When the sensor is being moved backwards, the car will go backwards, the more the sensor goes back the faster the car becomes.
* When the sensor is being moved to the right, the car will go to the right, the more the sensor goes right the faster the car becomes.
* When the sensor is being moved to the left, the car will go to the left, the more the sensor goes to the left the faster the car becomes.
* If the sensor is put in the middle with no movement, the car stops.
* The accelerometer sensor tilt values are being transmitted to duty cycle values so, the signal would go a certain percentage on and a certain percentage off and that is what affects the speed.

# Problems and Recommendations

* **Problem 1 (The unavailability of 12V and 5V Batteries):**

In order for the "L298" Dual H-Bridge Driver used in this project to function, a 12V Battery has to be connected with it but there was not a 12V Battery available, so, the best recommendation is to use three 3.7V Batteries to be connected in series in order to sum almost 12V. 0.9V Difference won't cause an affect as the H-Bridge would still run however it should not cross 12V or it would damage the bridge. Furthermore, the 16F877A needs 5 Volts but there was not a 5V battery available so, 9 Volts battery would be recommended to be used with a voltage regulator in order to convert it to 5 Volts, this would also be done for Arduino UNO as the board has to be connected with a 5V supply as well so, in this project, two 9V batteries and two voltage regulators were used.

* **Problem 2 (RF Sensor functionality and quality):**

The Radio Frequency sensor which included a transmitter and receiver in this project lacked good quality and connection, a wire was used to transmit data from the Arduino UNO board to the pic and with the code used in the design section, everything seemed to be correct but replacing the wire with the sensor and the signal was not being transmitted in the correct way, as a result of the weak functionality of this sensor. So, a long wire between the transmitter pin in the Arduino and the receiver pin in the PIC16F877A could be used instead, or the Bluetooth module would have helped as well in order to control the car’s speed and direction remotely. The best recommendation to be given is to use a better RF sensor but it might cost money as good RF sensor is supposed to be expensive, so, the long wire is still a better option or the Bluetooth Module.

* **Problem 3 (The weak movement of the car)**

After connecting everything together for the first time, it was noticed that the car barely moves despite everything being correct, the accelerometer sensor is giving signal but the car’s movement seemed like it was responding slowly; the reason behind this was the absence of two capacitors for the voltage regulator. Between the voltage input pin of the voltage regulator and the ground, a 0.22Micro farad capacitor has to be connected while between the voltage output pin and the ground, a 0.1Micro farad capacitor has to be connected. After doing this, the car responded as it should; because the capacitors reduce the ripple on the output by improving the transient response, also, they reduce the source impedance and resistance on the input. So, the best recommendation is to never forget to bring capacitors in order to improve the output response required.

* **Problem 4 (Complexity in communicating with the MPU6050 Sensor)**

Due to the complexity of the MPU6050 sensor that uses the I2C communication protocol, it was hard to establish communication between the PIC and the MPU6050 sensor without using special libraries. However, even when using an I2C library in the code, it didn't work. So to overcome this problem, we used an Arduino to communicate with the sensor and send the data from the sensor to the PIC.

# Conclusion

In conclusion, game stores are always competing with each other in order to deliver the best products, the more creative and different the outcome is, the more attractive it becomes for customers to buy it. RC (Radio Controlled) Cars are widely used in current game stores and different applications, the creativity of design and ideas can be a great factor in marketing these devices in game stores which boosts someone’s business financially. The process of design relies on many factors, starting with understanding each electrical and physical component and sensor’s background wisely and deeply in order to interface them together properly. In this project, the control of a car’s movement and direction was successfully achieved by the tilt values of a person’s hand, this new creative idea was fully done by using an accelerometer sensor, Arduino UNO and PIC16F877A and other components and sensors. Finally, using a RF sensor helped a lot in transmitting the information between Arduino UNO (after capturing the tilt data from the accelerometer sensor) and the pic. A PWM Code was generated in order to control the motor’s speed and direction, a code that connected the information between the accelerometer sensor and Arduino UNO was also generated, and finally, a serial communication code was generated in order to transmit information between the Arduino UNO board and the pic.

# References

[1] Sen “Principles of Electric Machines and Power Electronics.” John Wiley and Sons.

[2] M. Lab, “PIC16F877A microcontroller introduction and features,” Microcontrollers Lab, 10-Aug-2015. [Online]. Available: https://microcontrollerslab.com/pic16f877a-introduction-features/. [Accessed: 20-Jan-2023].

[3] K. Franz, “What is an H-bridge?,” Digilent Blog, 09-Dec-2014. [Online]. Available: https://digilent.com/blog/what-is-an-h-bridge/. [Accessed: 20-Jan-2023].

[4] Utmel.com. [Online]. Available: https://www.utmel.com/blog/categories/motors/h-bridge-working-circuits-and-applications. [Accessed: 20-Jan-2023].

[5] “L298, a Dual H-Bridge Motor Driver module,” Latest Open Tech From Seeed, 08-Oct-2019. [Online]. Available: https://www.seeedstudio.com/blog/2019/10/08/l298-all-about-l298-motor-driver/. [Accessed: 20-Jan-2023].

[6] N. Renuga Devi, T. Suganya, S. Vignesh, R. J. Rathish, T. A. Nguyen, and S. Rajendran, “Animal health monitoring using nanosensor networks,” in Nanosensors for Smart Agriculture, Elsevier, 2022, pp. 573–608.

[7] N. Zaman, “Arduino UNO: A short review,” EduGonist, 11-Oct-2021. [Online]. Available: https://www.edugonist.com/arduino-uno/. [Accessed: 20-Jan-2023].

[8] “Arduino Uno Rev3,” Arduino Official Store. [Online]. Available: https://store.arduino.cc/products/arduino-uno-rev3. [Accessed: 20-Jan-2023].

[9] B. Butterfly, “Radio Frequency Sensors for long distance detection,” FindLight Blog, 20-Oct-2019. [Online]. Available: https://www.findlight.net/blog/radio-frequency-sensors-for-long-distance-detection/. [Accessed: 20-Jan-2023].

[10] “Generating PWM using PIC microcontroller with MPLAB and XC8,” Circuitdigest.com. [Online]. Available: https://circuitdigest.com/microcontroller-projects/pic-microcontroller-pic16f877a-pwm-tutorial. [Accessed: 20-Jan-2023].

[11] Robu.in. [Online]. Available: https://robu.in/mpu6050-with-arduino/. [Accessed: 20-Jan-2023].

[12] T. Wilmshurst, Designing embedded systems with PIC microcontrollers: Principles and applications, 2nd ed. London, England: Newnes, 2009.