PROJECTS WITH ARDUINO & RASPBERRY PI

MOTOR CONTROL

Dogan Ibrahim

elektor

LEARN › DESIGN › SHARE

Motor Control

Projects with Arduino
&
Raspberry Pi Zero W

Dogan Ibrahim

an Elektor Publication

LEARN DESIGN SHARE

● This is an Elektor Publication. Elektor is the media brand of
Elektor International Media B.V.
78 York Street, London W1H 1DP, UK
Phone: (+44) (0)20 7692 8344

● Declaration

The author and publisher have used their best efforts in ensuring the correctness of the information contained in this book. They do not assume, or hereby disclaim, any liability to any party for any loss or damage caused by errors or omissions in this book, whether such errors or omissions result from negligence, accident or any other caus

● British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

● **ISBN 978-1-907920-66-0**

Prepress Production: D-Vision, Julian van den Berg
First published in the United Kingdom 2017
Printed in the Netherlands by Wilco

PREFACE

Arduino is probably the most widely used platform for developing microcontroller based projects. It is used by students all over the world, by electronic hobbyists, and by anyone else interested in developing a microcontroller based project. One of the strong points of Arduino is that it is an open source computer hardware and is therefore supported by large number of developers and groups. Arduino hardware is based on the popular Atmel AVR and ARM Cortex processors. The boards are equipped with sets of analog and digital input-output pins that may be interfaced to various external devices for monitoring and control applications. Arduino programs are widely developed using te Integrated Development Environment (IDE) which uses a dialect of features from the C and C++ programming languages.

Raspberry Pi Zero W is the latest low-cost member of the Raspberry Pi family of single board computers, which extends the basic family features by the addition of wireless LAN and Bluetooth connectivity. This is a tiny board costing about $15 and including a 1 GHz single-core BCM2835 CPU, RAM, and wireless chip supporting 2.4 GHz Wi-Fi 802.11n and Bluetooth 4.0. Raspberry Pi Zero W uses the Linux distribution Rasbian operating system, designed specifically for the Pi. The board includes large number of input-output ports and supports various peripheral protocols, such as I2C, SPI, UART and so on. The processor can be programmed using the highly popular Python programming language.

This book is about DC electric motors and their use in Arduino and Raspberry Pi Zero W based projects. Many tested and working projects are given in the book for real-time control of standard DC motors, stepper motors, and servo motors, and mobile robots.

One of the interesting and strong points of this book is that it gives complete projects for remote control of a mobile robot from mobile phones, using the Arduino Uno as well as the Raspberry Pi Zero W as the controller processors. These projects are developed using Wi-Fi as well as the Bluetooth connectivity.

I hope you like reading the book and find it useful for your next motor control or robotic project.

Prof Dr Dogan Ibrahim
London, 2017

CHAPTER 1 • ELECTRIC MOTORS

1.1 Overview

In this Chapter we will be looking at the types and basic principles of the electric motors, namely the DC motors which are commonly used in microcontroller based systems such as robotics, factory automation, domestic and industrial control and so on.

1.2 Types Of Electric Motors

An electric motor is an electrical machine that converts electrical energy into mechanical energy in the form of linear or rotational movements (or torque). Electric motors operate with the principles of magnetic fields where a force is generated between the motor's magnetic field and the current through its windings and this force causes linear or rotational movement.

Electric motors are classified into two categories as follows:

- AC motors
- DC motors

Within these categories, there are subdivisions where Figure 1.1 shows some important motor types in these subdivisions.

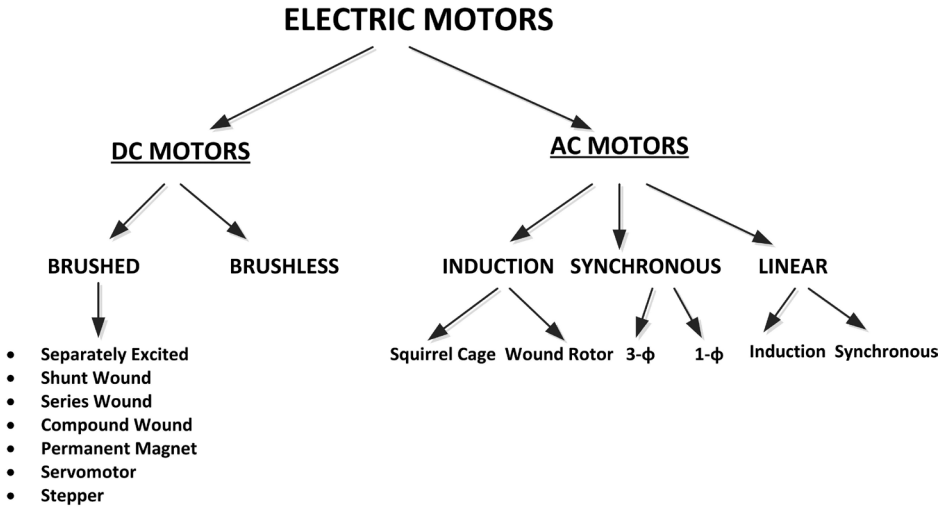

Figure 1.1 Electric motor types

Alternating Current (AC) motors are not commonly used in robotics because most robots are powered with batteries which are DC voltage. AC motors are generally used in industrial automation applications where very large torques may be required, such as in conveyor belts, cranes and so on.

In this book we are only interested in the following types of DC only electric motors:

- Brushed DC motors
- DC Servo motors
- DC stepper motors

The advantages of the brushed DC motors are:

- They are low-cost
- They operate with low DC voltages
- They are small in size
- It is easy to control their speeds
- They are widely available in all sizes
- They have good torque-speed characteristics

The disadvantages of the brushed DC motors are:

- The brushes wear out in brushed DC motors and may need replacement
- Brushes generate electrical sparks which could fire or explosions in certain environments
- Brushes generate RF noise that may interfere with nearby electronic equipment such as TVs, radios etc.
- Professional quality DC motors are expensive

1.3 Brushed DC Motors

Brushed DC motors normally have two terminals and when voltage is applied across these terminals the motor rotates proportionally. Brushed DC motors (BDC) are widely used in most electronic toys and in other low-cost motor applications (Figure1.2). They are very low-cost, easy to drive, and are available in many sizes and shapes, with and without gear mechanisms. Unlike other motor types (e.g. brushless DC, and AC), brushed DC motors do not require a controller to switch to switch the current in the motor windings. As shown in Figure1.3, all BDC motors are made up of:

- A stator
- A rotor
- Brushes and a commutator

Figure 1.2 Small brushed DC motor

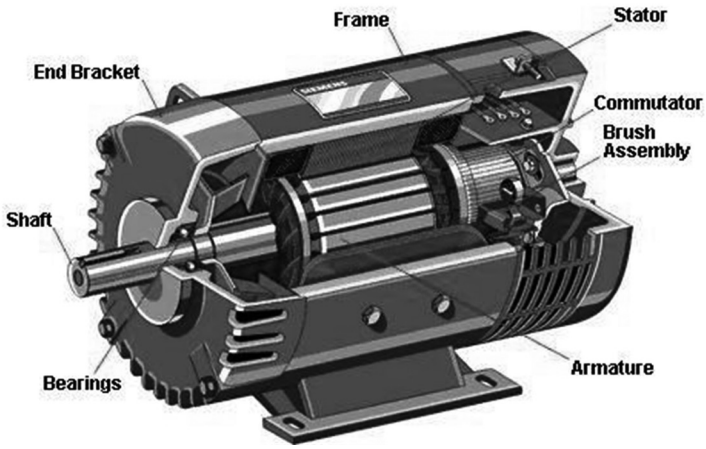

Figure 1.3 A typical BDC motor

The stator generates a stationary magnetic field that surrounds the rotor. This field is generated by either permanent magnets placed around the rotor, or by electromagnetic windings surrounding the rotor.

The rotor (see Figure 1.4) is also called the armature and it is made up of windings which produce magnetic field when energized by applying a DC voltage. The magnetic poles of the rotor are attracted to opposite magnetic poles of the stator, thus causing the rotor to turn.

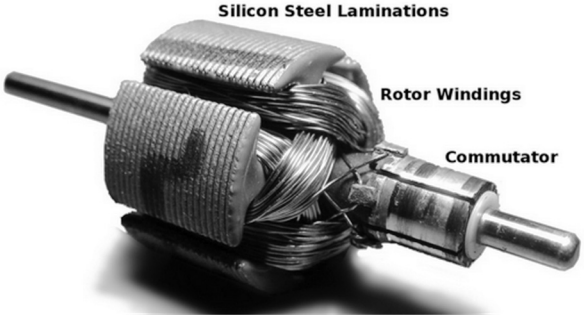

Figure 1.4 The rotor

The commutator (Figure 1.5) is placed on the axel of a BDC motor and as the motor turns, carbon brushes provide supply voltage to the rotor windings through the commutator. The brushes come in contact with different segments of the commutator, thus providing the required supply voltage to different windings of the rotor. Note that the commutator is part of the rotor and as the rotor turns so does the commutator.

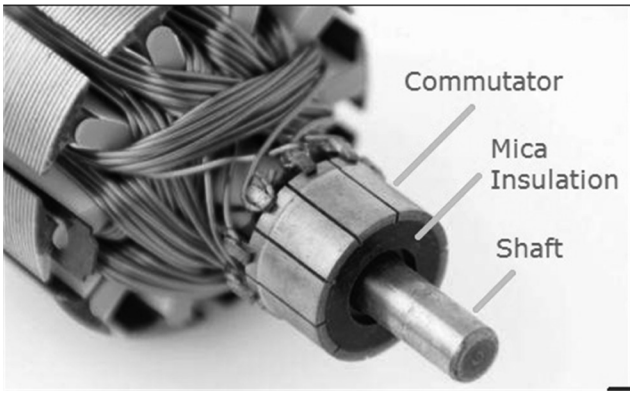

Figure 1.5 The commutator

1.3.1 Permanent Magnet BDC Motors

In a permanent magnet BDC motor the stator field is generated by permanent magnets (see Figure 1.6). i.e. the stator is a pair of permanent magnets. These motors have the advantages that they can be very small, and there are no field circuit copper losses. They have the disadvantages that the torque produced is low since the permanent magnets produce weak flux densities compared to externally supported magnetic fields. There is also the risk of demagnetization which can stop the motor functioning. Figure1.7 shows the circuit diagram of a permanent magnet BDC motor.

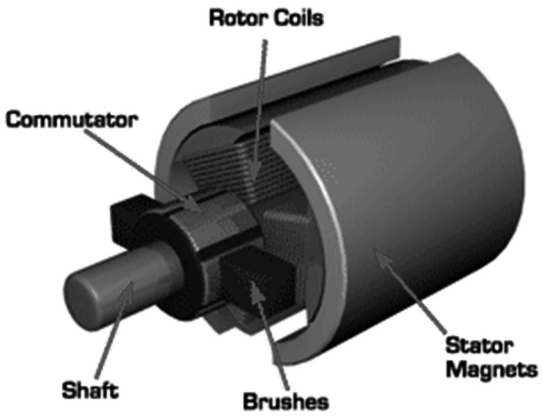

Figure 1.6 Permanent magnet BDC motor

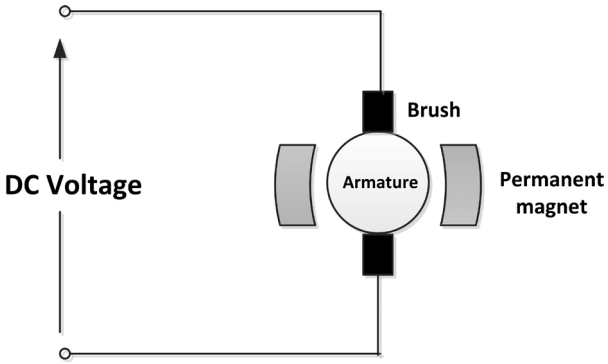

Figure 1.7 Circuit diagram of a permanent magnet BDC motor

1.3.2 Series Wound BDC Motors

In a series wound BDC motor the field coil is in series with the armature and thus, the field current is equal to the armature current. The advantage of these motors is that they produce high torque. These motors have the disadvantage that it is difficult to control their speed, and also the maximum speed is limited to around 5000 rpm. Figure 1.8 shows the circuit diagram of a series wound BDC motor.

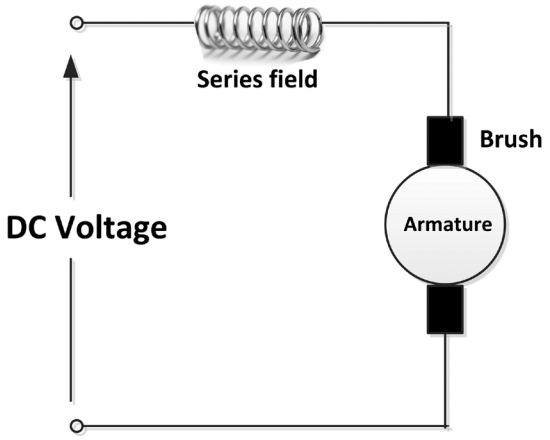

Figure 1.8 Circuit diagram of series wound BDC motor

1.3.3 Shunt Wound BDC Motors

Shunt wound BC motors have the field coil in parallel with the armature and thus the current in the field coil and the armature are independent of each other. The advantage of these motors is that they have excellent speed control properties. The speed can easily be changed by inserting a resistor in series with the armature or with the field coil. Figure 1.9 shows the circuit diagram of a shunt wound BDC motor.

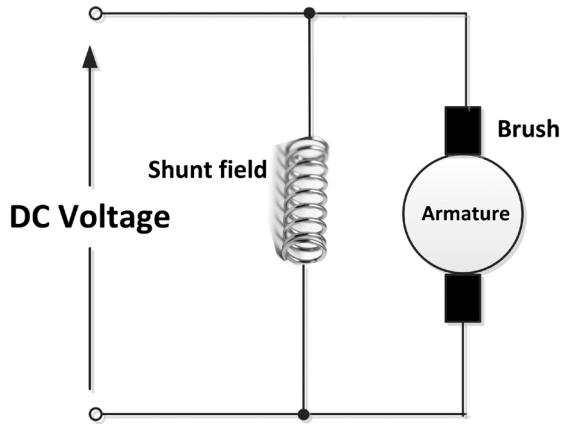

Figure 1.9 Circuit diagram of a shunt wound BDC motor

1.3.4 Compound Wound BDC Motors

These motor are a combination of shunt wound and series wound motors. These motors have the advantages that they have higher torque than series wound motors and their speed control is stable. Figure 1.10 shows the circuit diagram of a compound wound BDC motor.

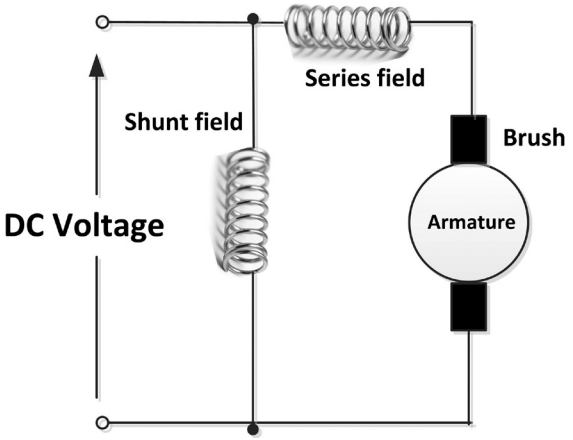

Figure 1.10 Circuit diagram of a compound wound BDC motor

1.3.5 Separately Excited BDC Motors

In a separately excited BDC motor the field coils are supplied from an independent source and thus the field current is not affected by changes in the armature current. These motors have high torques and stable speed control properties. Figure 1.11 shows the circuit diagram of a separately excited BDC motor.

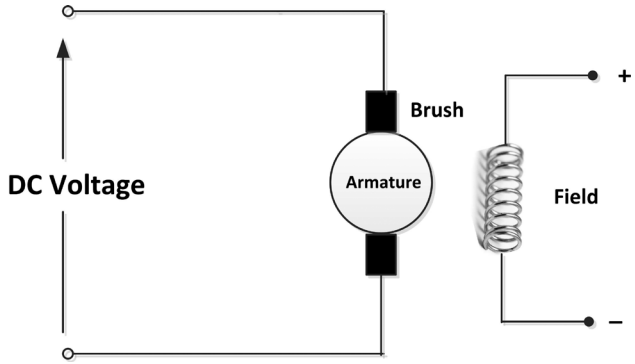

Figure 1.11 Circuit diagram of a separately excited BDC motor

1.3.6 Servo Motors

A servo motor allows for precise control of angular position of its rotor. It usually consists of a DC motor coupled with a sensor for feedback so that the rotary position of the motor can be controlled accurately. Servo motors run in closed loop mode to provide high performance and accuracy. These motors are commonly used in radio controlled cars, boats, helicopters, aeroplanes etc.

Servo motors are normally operated with PWM type pulses (positive going square waveform pulses) with a period of 50Hz (20ms period). In most types, a 1.5ms pulse puts the motor in a stationary position, a 1ms pulse turns the motor 90 degrees to one direction, and a 2ms pulse turns the motor 90 degrees in the opposite direction. Some types of servo motors are modified such that they rotate continuously. Figure 1.12 shows a small servo motor.

Figure 1.12 Small servo motor

1.3.7 Stepper Motors

Stepper motors (see Figure 1.13) rotate in discrete steps when electrical pulses are applied across its terminals in the correct sequence. The sequence of the applied pulses determine the direction of rotation, while the speed of the motor shaft rotation is directly related to the frequency of the input pulses, and the length of rotation is related to the number of applied pulses. Stepper motors are ideal for open loop position control applications such as in printers, plotters, PCB drilling machines and any other applications requiring precise control of the rotor position. Stepper motors have the disadvantage that the torque produced is not very large.

Figure 1.13 A small stepper motor

Basically, there are two types of stepper motors: unipolar and bipolar. The main difference between these two types is the applied voltage levels. Unipolar stepper motors (see Figure 1.14 for its circuit diagram) operate with positive voltages only (e.g. 0 and +12V). Bipolar motors (see Figure 1.15 for its circuit diagram) on the other hand operate with positive and negative voltages (e.g. -5V and +5V). As shown in the figures, unipolar motors require an additional wire in the middle of each coil. Bipolar motors have the advantages that they produce larger torques since the current flows in the entire coil and generates stronger electromagnetic field, and also they require one less wire for operation.

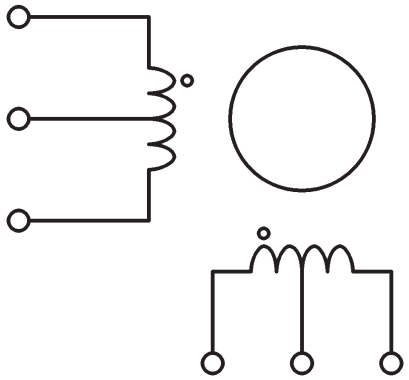

Figure 1.14 Circuit diagram of the unipolar stepper motor

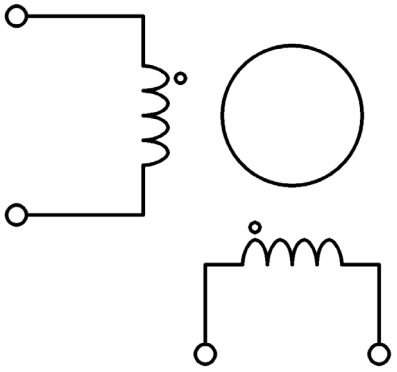

Figure 1.15 Circuit diagram of the bipolar stepper motor

1.4 Brushless DC Motors

In a brushless DC motor (BLDC) the rotor is a permanent magnet and the stator has windings. This is basically like a bushed DC motor turned inside out. There are no brushes or a commutator in a brushless DC motor and because of this these motors last longer and require little or no maintenance. In addition, brushless motors do not generate electrical sparks and are therefore cleaner, less noisy, and more reliable. The disadvantage of these motors is that they are more expensive. The sequence of voltages applied to the stator windings replace the function of the commutator and turn the motor. The control of a brushless DC motor is very different from the normal brushed DC motor as it requires sensors to detect the rotor angular position before applying the voltage pulses. Speed control of brushless DC motors are achieved using electronic Speed Control (ESC) modules.

1.5 Motor Selection

Choosing an electric motor for an application is an important task when designing a mobile robot or any other electric car. Some of the important motor specifications to consider during a choice are:

- Torque
- Speed
- Accuracy
- Operating voltage
- Cost
- Physical size and weight

1.5.1 Torque

Torque is the turning ability of an electric motor. The higher the torque the more power the motor has while turning an object. Torque is similar to force, but it is the rotational (or twisting) force rather than linear force. Torque is calculated by multiplying the force applied to an object with the perpendicular distance from the pivot point to the point where the force is applied. A simple example is to imagine pushing a door to open. The applied force causes the door to rotate about its hinges (the pivot point). The closer you are to the hinges the harder it is to open the door because you apply a smaller torque. In SI units torque is measured in Newton-metres, or Nm for short.
The required torque to drive the wheels of a robot (or an electric car) depends upon the following factors:

- Rolling resistance
- Slope resistance
- Accelerating force

Rolling Resistance

The rolling resistance occurs as a result of the rolling motion of the wheel and it depends upon the type of surface the wheel is on. The rolling resistance can be calculated as:

$$R = W \times C_r$$

and

$$W = m \times g$$

Where, **W** is the gross vehicle weight, **m** is the mass of the vehicle, **g** is the acceleration due to gravity ($9.81 m/s^2$), and C_r is the rolling resistance coefficient, given in Table 1.1 for some common surfaces.

Surface	**Rolling Resistance**
Concrete	0.010
Asphalt	0.012
Snow (2 inches)	0.025
Mud (firm)	0.037
Mud (soft)	0.150
Grass (firm)	0.055
Sand (firm)	0.06

Table 1.1 Rolling resistances of some surfaces

Slope Resistance
The slope resistance is important when the wheel is moving on an inclined surface. The slope resistance is given by:

$$S = W \times \sin \varphi$$

Where **φ** is the inclination angle of the surface. For flat surfaces φ = 0 and therefore there is no slope resistance.

Accelerating Force
Accelerating force helps the vehicle to come to a pre-defined speed from rest in a specified period of time. It is given by:

$$A = m \times a \text{ or } A = W \times a / g$$

Where, **a** is the required acceleration, **m** is the mass of the vehicle, and **W** is the gross vehicle weight as before.

The total required motor torque can then be calculated as follows:

$$T = R + S + A$$

1.5.2 Speed
Speed requirement is easier to estimate as it depends on how fast we want the robot to run. DC motors usually run at speeds of thousands of RPMs with very low torque. In practical applications we need to use gear mechanisms between the motor shaft and the actual wheel. The resultant motor speed with a gear mechanism is given by:

$$\text{Resultant speed} = \text{Actual speed} / \text{Gear ratio}$$

1.5.3 Accuracy
Accuracy depends upon whether the motor is used for speed control or for position control. In speed control applications we want the motor speed to stay the same even if a load is applied to the motor shaft. In position control applications we want the motor shaft to rotate the required amount and then stop. Accurate motor control is normally obtained using closed loop control where feedback is used to sense the current speed of the position of the motor. With the help of this feedback the error between the desired value and the actual measured value is eliminated or minimized.

1.5.4 Operating voltage
The operating voltage is also an important factor when choosing a motor. Most electric motors used in toys and robotics are brushed DC motors operating with batteries. In general, higher voltages result is higher motor speeds but this may require more batteries. Using more batteries will make the vehicle heavier and also the cost of the batteries will increase. Common voltages used in DC motors are 3, 5, 12 and 24 Volts. It is important to realise that the motor windings may be damaged if a higher voltage than the absolute maximum

is applied to a motor. Similarly, the motor will not rotate if the applied voltage is lower than the minimum operating voltage.

1.5.5 Cost

The cost of the motor is important in robotic applications. Usually lower cost motors may not be of high quality and the brushes may need to be changed frequently. Such motors can only be used in hobby applications. A choice should be made between the desired quality and the cost.

1.5.6 Physical size and weight

Electric motors are available in many different shapes, sizes, and weights. If the motor is to be used on a robot vehicle then it is important to choose a light weight motor with as small size as possible. A lighter vehicle can be moved with less torque and this lowers the overall cost of the project.

1.6 Transfer Function of a Brushed DC Motor

The transfer function of a system is the ratio of its output to its input, represented using the Laplace transform. It is important to know the transfer function of a system before feed-back can be applied to it for closed loop control. In this section we shall derive the transfer function of a permanent magnet brushed DC motor (or a separately excited DC motor) which is one of the most commonly used motors in robotic applications.

Figure 1.16 shows the circuit diagram of the motor where R (in Ohms) and L (in Henries) are the resistance and inductance of the motor windings, V (in Volts) is the applied DC voltage, J (in $Kg.m^2/s^{2)}$ is the inertial of the load attached to the motor, and V_e (in Volts) is the back EMF (electromagnetic force) of the motor.

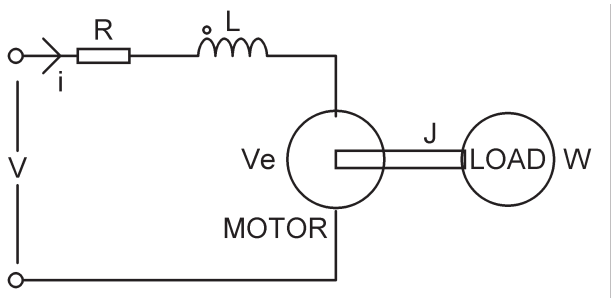

Figure 1.16 Permanent magnet DC motor

The torque generated by the motor is proportional to the current through the motor windings and it can be written as:

$$T = K_T\, i$$

Where K_T is the motor torque constant and i (in Amperes) is the current (1.1)

The back EMF is proportional to the motor angular speed W (rad/s) and is given by:

$$V_e = K_E W \quad (1.2)$$

Notice that W = dΘ/dt where Θ is the angle rotated by the motor (in radians). K_E is the motor back emf constant.

Using Kirchoff's law, we can write the following equation about the motor circuit:

$$V - V_e = Ri + L\, di/dt \quad (1.3)$$

Or,

$$V = Ri + Ldi/dt + K_E W \quad (1.4)$$

Also,

$$T = J\, dW/dt \quad (1.5)$$

From (1.1) and (1.5) we can write:

$$i = J/K_T\, dW/dt \quad (1.6)$$

using equations (1.6) and (1.4):

$$V = \frac{RJ}{K_T}\frac{dW}{dt} + \frac{LJ}{K_T}\frac{d^2W}{dt^2} + K_E W \quad (1.7)$$

In small motors the inductance L is very small and can be neglected. Equation (1.7) then becomes:

$$V = \frac{RJ}{K_T}\frac{dW}{dt} + K_E W \quad (1.8)$$

Equation (1.8) is a first order equation. Using the Laplace transforms where s is the Laplace operator, we can re-write it as:

$$\frac{W(s)}{V(s)} = \frac{1}{K_E + sRJ/K_T} \quad (1.9)$$

or,

$$\frac{W(s)}{V(s)} = \frac{K_T}{K_T K_E + sRJ} \quad (1.10)$$

We can therefore represent the transfer function of the motor as in Figure 1.17.

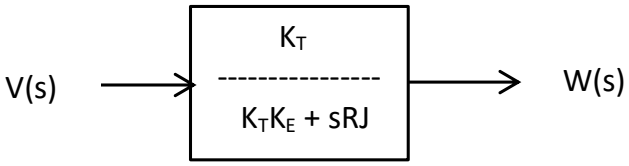

Figure 1.17 DC motor transfer function

When a step voltage V_O is applied across the motor terminals the motor speed increases exponentially until it settles down at a final value. The step response of the motor (i.e. the motor speed when a sudden step voltage is applied) can be derived by multiplying the transfer function by 1/s and then taking the inverse Laplace transform. The result is given below:

$$W(t) = \frac{V_o}{K_E}\left(1 - e^{-\frac{K_T K_E t}{RJ}}\right) \tag{1.11}$$

Initially the speed is 0, and it increases exponentially, having a final value of $W(t) = V_O/K_E$. Figure 1.18 shows the typical step response of the motor.

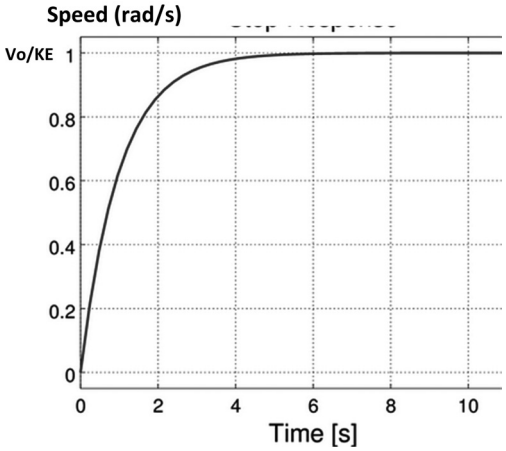

Figure 1.18 Step response of a DC motor

Notice that the term $T = RJ/K_TK_E$ is known as the motor time constant of the motor and this is the time it takes for the motor to reach 63% of its final value. Using the time constant, we can re-write equation (1.11) as:

$$W(t) = \frac{V_O}{K_E}\left(1 - e^{-t/T}\right) \tag{1.12}$$

1.7 Torque – Speed Characteristics

Torque-speed characteristics is important for a given motor as it shows the change of the available motor torque with the speed. In general, greater torque is obtained at lower speeds. Figure 1.19 shows the torque-speed characteristic curve of some brushed DC motors.

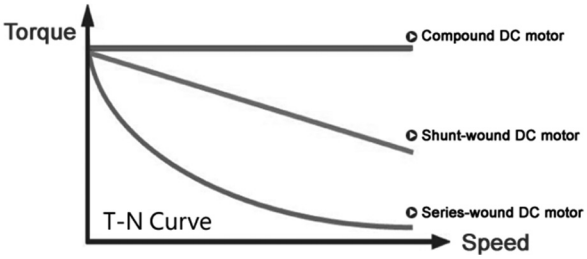

Figure 1.19 Torque-speed curve of DC motors

1.8 Summary

In this Chapter we had a brief look at the types of electric motors. DC motors, especially the brushed types are commonly used in most microcontroller based robotic and mechanical movement applications. We have explored the basic principles and advantages/disadvantages of these motors. Additionally, the servo motors and stepping motors have been described briefly. The important topic of the choice of a motor for a project, and the DC motor transfer function and step response, have also been explained briefly.

In the next Chapter we shall be looking at how to use the brushed DC motors in microcontroller based projects.

CHAPTER 2 • SIMPLE DC MOTOR PROJECTS

2.1 Overview

In this Chapter, simple DC motor projects are explained. In all the projects in this Chapter a small brushed permanent magnet DC motor will be used with an Arduino UNO development board and also with a Raspberry Pi Zero W development board. Arduino programs will be developed using the Arduino IDE and the Raspberry Pi programs will be developed using Python programming language.

The projects will be described wherever possible by giving the following for each project:

- Project Title and brief description
- Block diagram
- Circuit diagrams (both Arduino Uno and Raspberry Pi)
- Project construction
- Program listings (both Arduino Uno and Raspberry Pi)
- Suggestions (where possible)

In this Chapter the term microcontroller will be used to refer to the Arduino UNO development board and also to the Raspberry Pi Zero W development board.

2.2 PROJECT 1 – Motor ON/OFF Control

This is a very simple project where a small brushed DC motor that can operate with 3 – 6V is connected to the microcontroller through a transistor driver. The motor is turned ON for 10 seconds and then OFF for 5 seconds. This process is repeated forever until stopped manually. The aim of this project is to show how a DC brushed motor can be connected and operated from a microcontroller.

2.2.1 Block Diagram

Figure 2.1 shows the block diagram of the project. One of the output ports of the microcontroller is connected to the motor through a transistor driver.

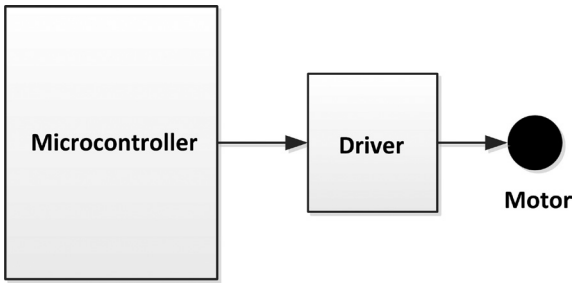

Figure 2.1 Block diagram of the project

2.2.2 Circuit Diagram – Arduino Uno

Figure 2.2 shows the circuit diagram for the Arduino Uno. The base of a BC108 type NPN transistor is connected to pin 3 (*see Appendix C for Arduino Uno pin diagram*) of the Ardui-

no Uno through a 3.3K resistor, where the transistor is used as a switch. The motor is connected across the collector of the transistor and the positive power supply. In this project the motor operates with 3V – 6V. If you are using a larger motor requiring voltage higher than 5V (or higher current that cannot be supplied by Arduino Uno) make sure that you use an external power supply to drive the motor. Notice that a diode is connected across the motor. This diode prevents back emf voltage from damaging the transistor. Back EMF is generated when the motor is spinning and when the motor stops its windings still carry charge which result in a reverse current to flow and this current can damage the transistor if the diode is omitted. The diode must be chosen so that it can dissipate this current.

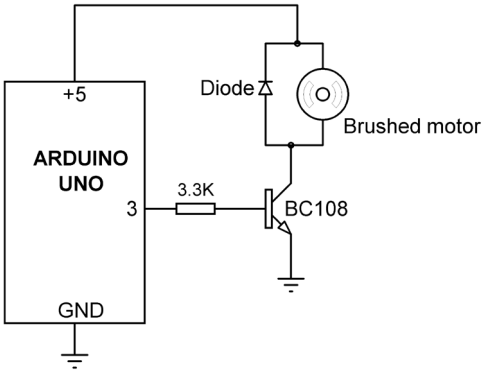

Figure 2.2 Arduino Uno circuit diagram

Figure 2.3 shows the project constructed on a breadboard.

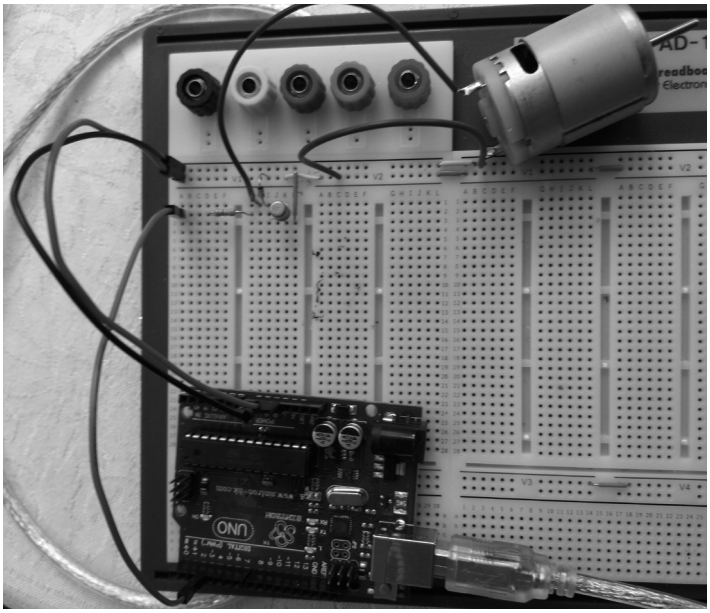

Figure 2.3 Arduino Uno project on a breadboard

2.2.3 Circuit Diagram – Raspberry Pi

Figure 2.4 shows the circuit diagram of the project where pin 2 (*see Appendix D for Raspberry Pi Zero W pin diagram*) is connected to the base of the switching transistor. An external power supply (shown as +V in Figure 2.4) is recommended since the +5V supply voltage of the RPi Zero W does not have enough current to drive most motors. The project constructed on a breadboard is shown in Figure 2.5.

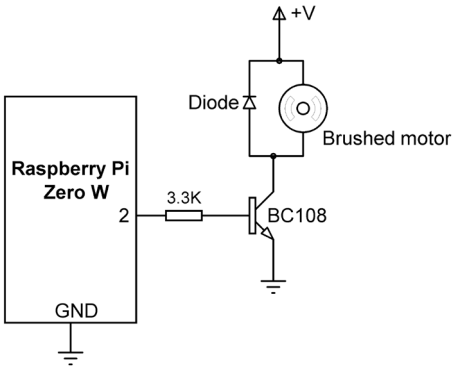

Figure 2.4 Raspberry Pi circuit diagram

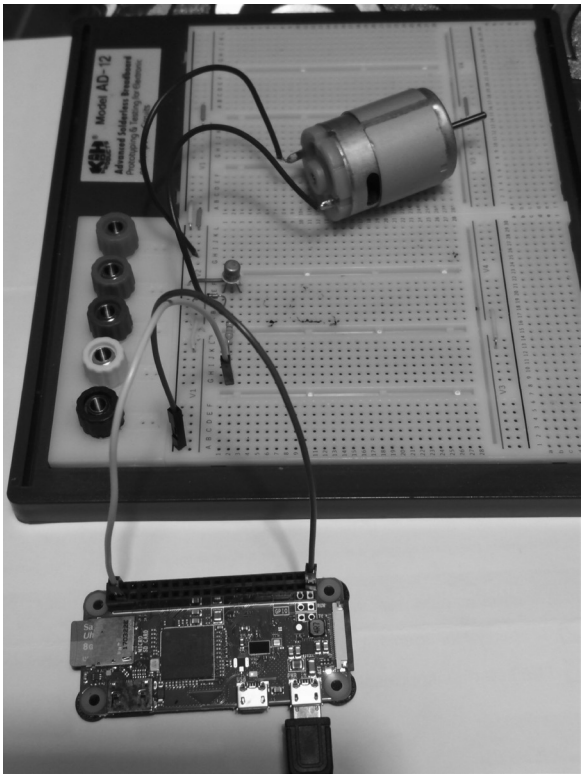

Figure 2.5 Raspberry Pi project on a breadboard

2.2.4 Program Listing – Arduino Uno

Figure 2.6 shows the Arduino program listing (program: Motor1). Compile the program as usual and upload to your Arduino Uno.

At the beginning of the program port 3 is defined as Motor, and ON and OFF are defined as HIGH and LOW respectively. Inside the setup routine port 3 is configured as an output. The main program runs in an endless loop where the motor is turned ON for 10 seconds, and OFF for 5 seconds.

```
/*****************************************************************
*              SIMPLE ON/OFF MOTOR CONTROL
*              ===========================
*
* This is a very simple motor ON/OFF control. Pin 3 of the Arduino
* is connected to a small brushed DC motor through a transistor
* switch. The motor is turned ON for 10 seconds, then OFF for
* 5 seconds, and then ON again for 10 seconds. This process is
* repeated forever until stopped manually.
*
*
* File  : Motor1
* Date  : August 2017
* Author: Dogan Ibrahim
*****************************************************************/
#define Motor 3                                  // Motor pin
#define ON HIGH
#define OFF LOW

//
// Configure pin 3 where motro is connected as output
//
void setup()
{
  pinMode(Motor, OUTPUT);

}

//
// Turn ON the motor for 10 seconds, then OFF for 5 seconds. This
// process is repated forever
//
void loop()
{
   digitalWrite(Motor, ON);                      // Motor ON
   delay(10000);                                 // Wait 10 sec
   digitalWrite(Motor, OFF);                     // Motor OFF
```

```
   delay(5000);                                  // Wait 5 sec
}
```

Figure 2.6 Arduino program listing

2.2.5 Program Listing – Raspberry Pi

Figure 2.7 shows the Raspberry Pi Python program listing (program: motor1.py). At the beginning of the program the GPIO library and the time library are imported to the program. Name **Motor** is assigned to port 2 and this pin is configured as output. The program then sets up an endless loop using a while statement. Inside this loop the motor is turned ON for 10 seconds, and OFF for 5 seconds.

Create the program using the **nano** editor and run it in the command mode (see Appendix A) by giving the following command:

sudo python motor1.py

```
#-------------------------------------------------------------------
#                         SIMPLE ON/PFF MOTOR CONTROL
#                         ===========================
#
# This is a very simplr motor ON/OFF control. Port 2 of the Rpi ZW
# is connected to a small brushed DC motor through a transistor
# switch. The motor is turned ON for 10 seconds, then OFF for 5
# seconds, and then ON again for 10 seconds. This process is repeated
# forever until stopped manually.
#
# The H bridge is connected to pins 2,3,4,17 of the RPi
#
# File  : motor1.py
# Date  : August 2017
# Author: Dogan Ibrahim
#-------------------------------------------------------------------
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
Motor = 2

GPIO.setup(Motor, GPIO.OUT)

#
# Motor spins for 10 seconds, then stops for 5 seconds. This process
# is repeated until stopped manually
#
while True:
  GPIO.output(Motor, 1)
```

```
time.sleep(10)
GPIO.output(Motor, 0)
time.sleep(5)
```

Figure 2.7 Python program listing

Notice that you can copy your program file from the RPi ZW to a PC (e.g. for printing etc) by using the **WinSCP** file transfer utility. This utility runs on the PC and can be downloaded from the Internet free of charge.

2.2.6 Using a MOSFET Transistor

In Figures 2.2 and 2.4 a bipolar transistor is used as a switch to control the motor. There are many applications however where a larger motor may be used and/or the motor requires large current to operate and a standard transistor may not be able to supply the required current. In such applications it is recommended to use a power MOSFET transistor, such as the IRL540 ,IRLI520N, VNF66AFD or many others, as a switch. It is recommended to use a MOSFET transistor where the Gate voltage is logic level compatible. In this project the IRL540 is used whose Gate voltage is logic level compatible. The continuous Drain current of this power MOSFET transistor can be as high as 20 Amperes (of course an external power supply must be used for large currents since the USB driven Arduino power supply cannot provide more than 0.8 Amperes of current). Figure 2.8 shows the use of the MOSFET transistor as a switch to control the DC motor. Notice that a 10K resistor is used to make sure that the transistor is OFF when its Gate pin is floating. The pin configuration of the IRL540 MOSFET transistor is shown in Figure 2.9. Notice that there is no change in the program. Figure 2.10 shows the circuit built on a breadboard. The Raspberry Pi construction is similar with the bipolar transistor replaced with the power MOSFET.

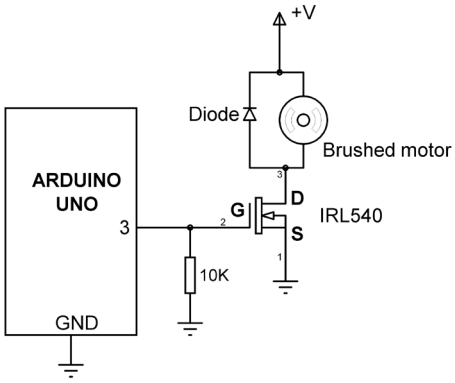

Figure 2.8 Using a power MOSFET transistor

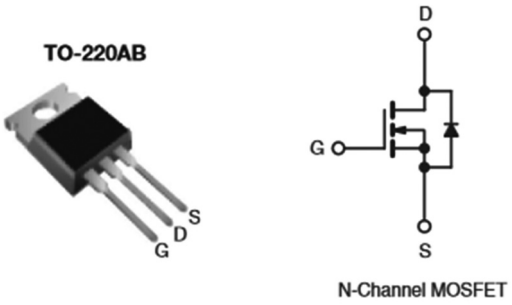

Figure 2.9 Pin configuration of IRL540

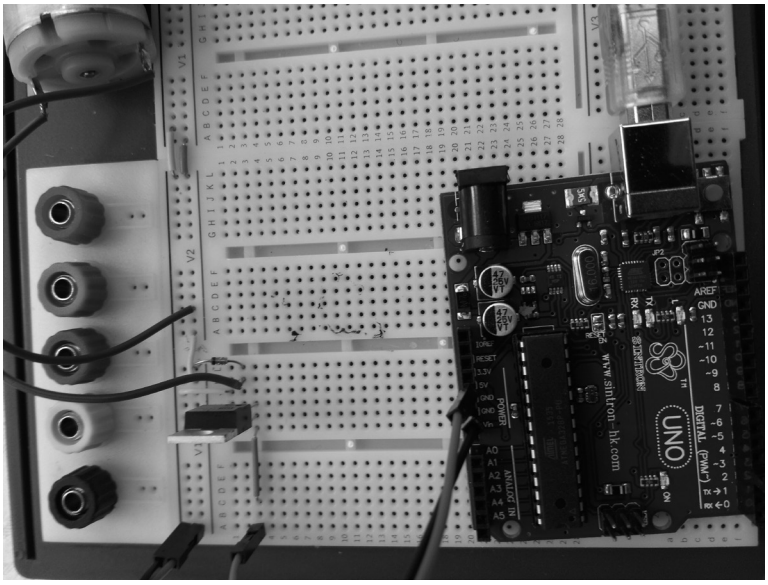

Figure 2.10 Circuit built on a breadboard

2.2.7 Using a Relay

Motors can also be controlled using relays as the switching devices. Figure 2.11 shows the circuit diagram where the transistor switch is replaced with a relay. The project constructed on the breadboard is shown in Figure 2.12. Notice that the 3-pin relay with built-in diode from Elektor is used in this project for simplicity. Again, it is recommended to use an external power supply for the motor. Note that there is no change in the program.

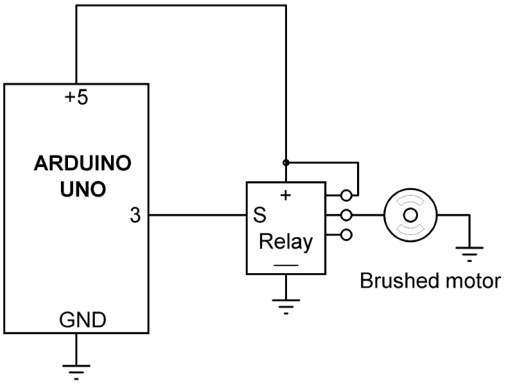

Figure 2.11 Using a relay as a switch

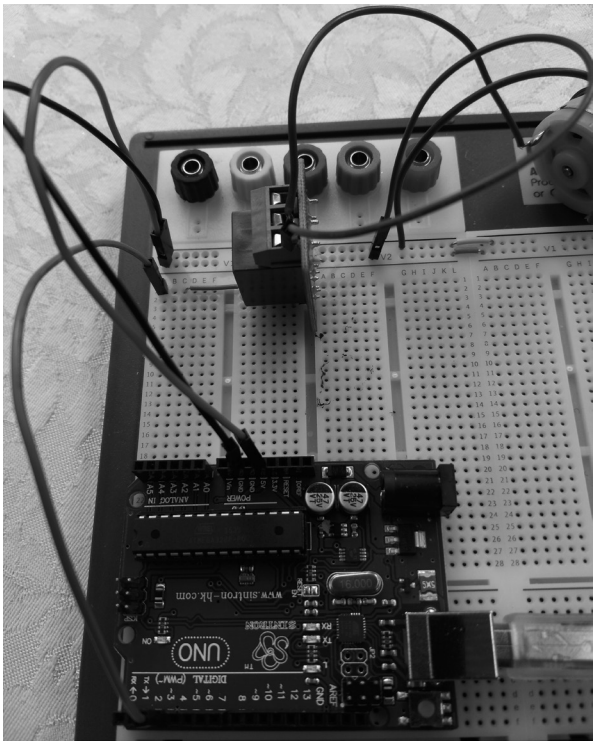

Figure 2.12 Circuit built on a breadboard

2.3 PROJECT 2 – Two Speed Motor Speed Control

This is again a very simple project where a small brushed DC motor is connected to the microcontroller through a power MOSFET transistor switch as in section 2.2.6. In addition, a push-button switch is connected to the microcontroller. Normally the motor spins at full speed. Pressing the button will reduce the speed of the motor to 25% of its maximum value. The aim of this project is to show how PWM can be used to control the speed of a DC brushed motor.

2.3.1 Block Diagram

Figure 2.13 shows the block diagram of the project. A motor driver (MOSFET transistor) and a push-button switch are connected to the microcontroller.

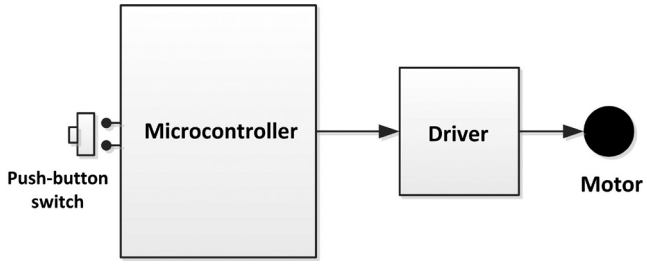

Figure 2.13 Block diagram of the project

The DC motor in this project is controlled using PWM waves. PWM control is commonly used to control motors efficiently. PWM signal is basically a high frequency positive only square wave (typically 1 kHz or greater). The duty cycle of this waveform is varied in order to vary the average voltage applied to the load. The duty cycle of PWM wave is normally expressed as a percentage:

$$\text{Duty Cycle} = \text{ON time} / (\text{ON time} + \text{OFF time}) \times 100\ \% \quad (2.1)$$

or, since Period, T = ON time + OFF time

$$\text{Duty Cycle} = (T_{ON} / T) \times 100\ \% \quad (2.2)$$

where T_{ON} is the ON time.

Figure 2.14 shows PWM waveforms with different duty cycles.

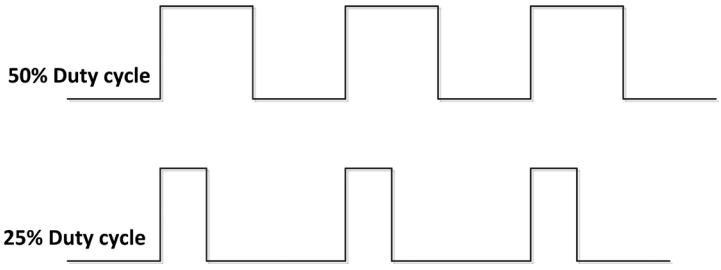

Figure 2.14 PWM waveforms

The average value of the voltage applied to the motor can be calculated by considering a general PWM waveform shown in Figure 2.15. The average value A of waveform f(t) with period T and peak value y_{max} and minimum value y_{min} is calculated as:

$$A = \frac{1}{T}\int_0^T f(t)\,dt \tag{2.3}$$

or,

$$A = \frac{1}{T}\left(\int_0^{T_{ON}} y_{\max}\,dt + \int_{T_{ON}}^{T} y_{\min dt}\right) \tag{2.4}$$

In a PWM waveform y_{min} = 0 and the above equation becomes

$$A = \frac{1}{T}\left(T_{ON}\, y_{\max}\right) \tag{2.5}$$

or,

$$A = D\, y_{\max} \tag{2.6}$$

where D is the duty cycle. As it can be seen from equation (2.6), the average value of the voltage applied to the motor is directly proportional to the duty cycle of the PWM waveform. If the peak value of the applied voltage is y_{max}, then as the duty cycle varies from 0% to 100%, the average voltage at the motor varies from 0V to y_{max}.

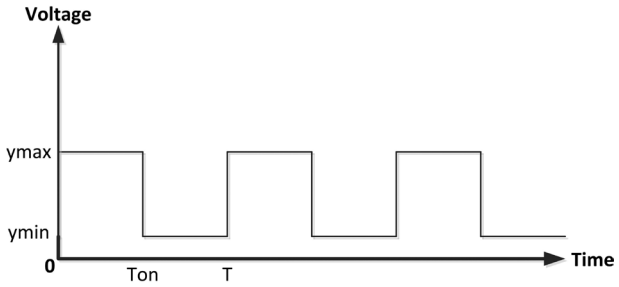

Figure 2.15 PWM waveform with y_{max} and y_{min}

2.3.2 Circuit Diagram – Arduino Uno

The Arduino Uno circuit diagram of the project is shown in Figure 2.16. The MOSFET transistor is connected to GPIO port 3, and the push-button switch is connected to GPIO port pin 4. The button output is pulled-high by the software and therefore it is normally at logic 1. Pressing the button changes the button output to logic 0. Figure 2.17 shows the circuit built on a breadboard.

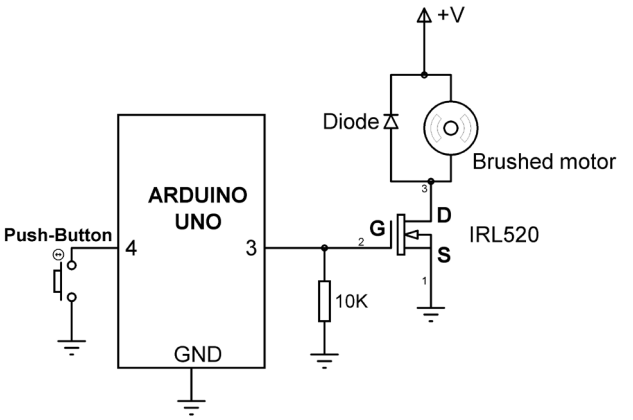

Figure 2.16 Arduino Uno circuit diagram

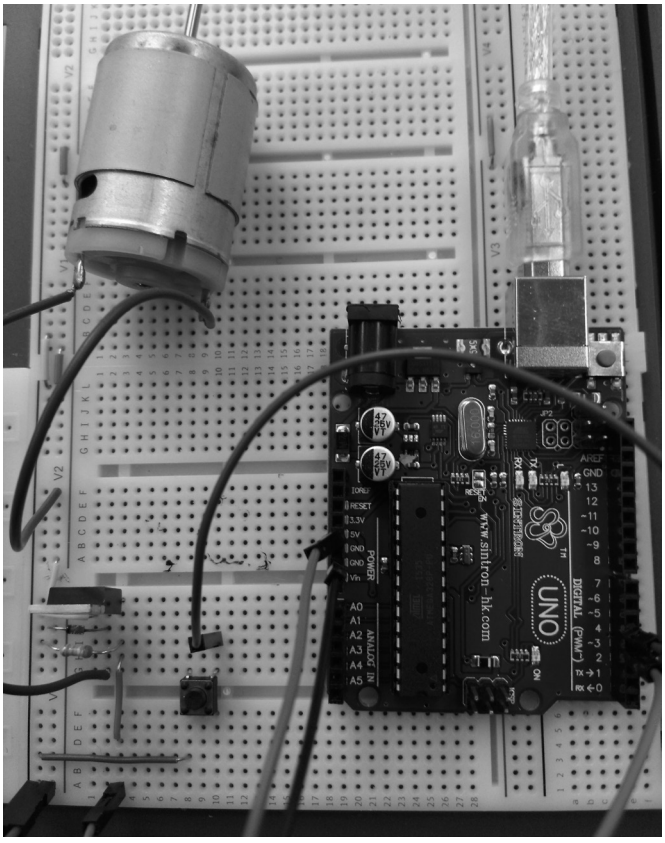

Figure 2.17 Circuit built on a breadboard

2.3.3 Circuit Diagram – Raspberry Pi

Figure 2.18 shows the circuit diagram of the project where port pin 2 is connected to the MOSFET transistor and the push-button switch is connected to port pin 4. The input pin

where the button is connected to is at logic 1, pulled high by software. The project constructed on a breadboard is shown in Figure 2.19.

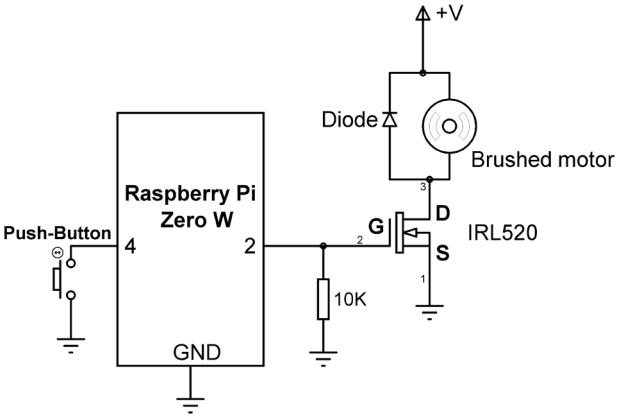

Figure 2.18 Raspberry Pi circuit diagram

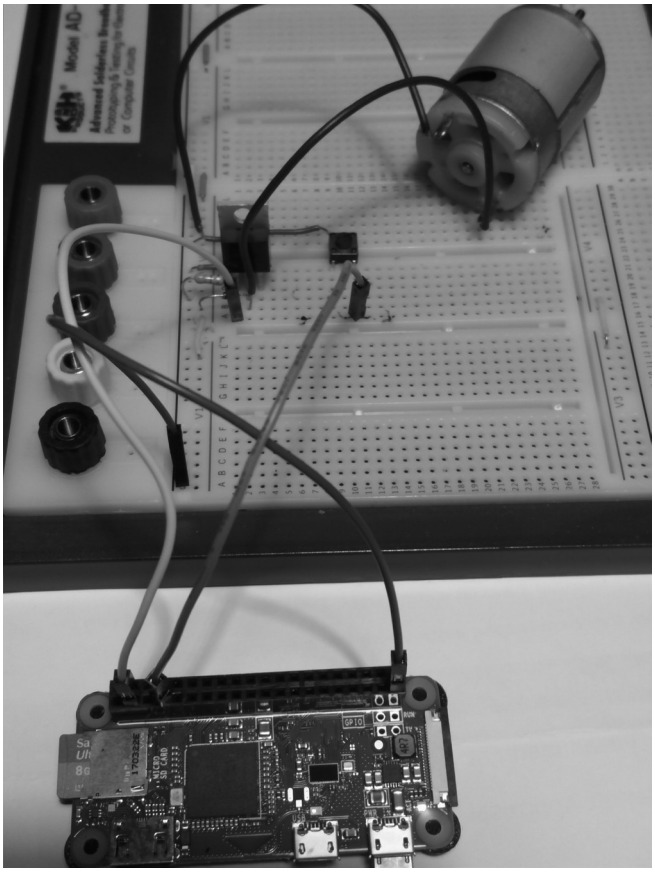

Figure 2.19 Raspberry Pi project on a breadboard

2.3.4 Program Listing – Arduino Uno

Arduino Uno offers six PWM outputs which are connected to three timers on the processor. These are:

Pin 5 and 6	- connected to Timer0
Pin 9 and 10	- connected to Timer1
Pin 11 and 3	- connected to Timer2

The frequency of the PWM depends upon the speed of the Timer/Counter which depends upon counter's clock divided by a pre-scaler value. The pre-scaler is 3-bits wide and is stored in the three least significant bits of the Timer/Counter registers CS02, CS01, and CS00. The Timer/Counter registers are TCCR0B, TCCR1B, and TCCR2B. Notice that the pairs of PWM pins have the same frequency. For example, PWM on pins 11 and 3 are controlled by Timer/Counter TCCR2B, pins 9 and 10 are controlled by TCCR1B, and pins 5 and 6 are controller by TCCR0B.

The default values of the PWM frequencies at boot time are as follows:

Pins 5 and 6:	976.56 Hz
Pins 9 and 10:	490.20 Hz
Pins 11 and 3:	490.20 Hz

In order to change the PWM frequency, we must re-load the three least significant bits of the required Timer/Counter registers. The values and corresponding PWM frequencies for PWM pins 11 and 3 are given below (these statements must be included in the setup routine of the program):

```
TCCR2B = TCCR2B & B11111000 | B00000001 frequency = 31372.55 Hz
TCCR2B = TCCR2B & B11111000 | B00000010 frequency = 3921.16 Hz
TCCR2B = TCCR2B & B11111000 | B00000011 frequency = 980.39 Hz
TCCR2B = TCCR2B & B11111000 | B00000100 frequency = 490.20 Hz (DEFAULT)
TCCR2B = TCCR2B & B11111000 | B00000101 frequency = 245.10 Hz
TCCR2B = TCCR2B & B11111000 | B00000110 frequency = 122.55 Hz
TCCR2B = TCCR2B & B11111000 | B00000111 frequency = 30.64 Hz
```

In this program the PWM frequency is set to 980.39 Hz by using the following statement in the setup routine:

```
TCCR2B = TCCR2B & B11111000 | B00000011
```

Figure 2.20 shows the Arduino program listing (program: Motor2). Compile the program as usual and upload to your Arduino Uno. At the beginning of the program port pins 3 and 4 are assigned to names Motor and Button respectively. Inside the setup routine Motor is configured as output and Button is configured as input and this pin is pulled up so the input is normally at logic 1.
On Arduino Uno the duty cycle ranges from 0 to 255 where 0 corresponds to 0% and 255

corresponds to 100%. Inside the program loop the state of Button is checked. If the Button is not pressed, the duty cycle is set to 100% (corresponding to 255), otherwise the duty cycle is set to 25% (corresponding to 64).

```
/*****************************************************************
*              TWO SPEED MOTOR CONTROL
*              ======================
*
* This is a very simple motor speed control. Pin 3 of the Arduino
* is connected to a small brushed DC motor through a power MOSFET
* switch. Also, pin 4 is connected to a push-button switch. The
* output of the switch is normally at logic 1 (pulled up) and it
* goes to logic 0 when the button is pressed. The motor normally
* spins at full speed and when the button is pressed the speed
* falls by 25%. Motro voltage has PWM waveform that has a 100%
* duty cycle when the switch is not pressed, and 50% duty cycle
* when the switch is pressed.
*

* File  : Motor2
* Date  : August 2017
* Author: Dogan Ibrahim
*****************************************************************/
#define Motor 3                                 // Motor pin
#define Button 4                                // Button pin

//
// Configure pin 3 where motro is connected as output
//
void setup()
{
  pinMode(Motor, OUTPUT);                       // Motor is output
  pinMode(Button, INPUT_PULLUP);                // Button is input
  TCCR2B = TCCR2B & B11111000 | B00000011;      // PWM freq = 980.39Hz
}

//
// Generate PWM waveform at motor pin with a 100% duty cycle. When
// the button is pressed, change the duty cycle to 25%. Choose the
// PWM frequency as 1 kHz.
//
void loop()
{
   if(digitalRead(Button) == 0)
     analogWrite(Motor, 64 );               // 25% duty cycle
```

```
    else
      analogWrite(Motor, 255);              // 100% duty cycle
}
```

Figure 2.20 Arduino program listing

Figure 2.21 shows the voltage waveform sent to the motor when the button is pressed (i.e. 25% duty cycle). This graph was obtained using a PCSGU250 PC based oscilloscope. The horizontal axis was 1ms/division and the vertical axis was 3V/division. Notice that the PWM frequency is approximately 1 kHz.

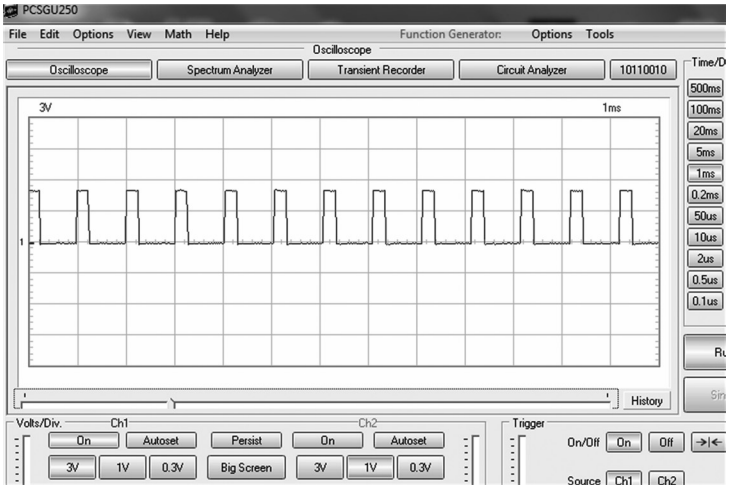

Figure 2.21 Voltage waveform sent to the motor

2.3.5 Program Listing – Raspberry Pi

Figure 2.22 shows the Raspberry Pi Python program listing. Python PWM library supports the following functions:

p = GPIO.PWM(channel, frequency)	Configure channel for PWM with specified frequency
p.start(DC)	Start PWM with the specified Duty Cycle
p.ChangeFrequency(frequency)	Change PWM frequency
p.ChangeDutyCycle(DC)	Change Duty Cycle

At the beginning of the program the GPIO library is imported, port pin 2 is configured as output and port pin 4 is configured as input with the pin pulled high by software. The PWM is started with frequency 1000Hz (1kHz) at duty cycle of 100% at pin 4 where the motor is connected to. When the button is pressed the duty cycle is changed to 25%.

```
#-----------------------------------------------------------------------
#                          TWO SPEED MOTOR CONTROL
#                          =======================
#
# This is a very simple motro speed control. Port 2 of the Rpi ZW
# is connected to a small brushed DC motor through a power MOSFET
# switch. Also, pin 4 is connected to a push-button switch. The
# output of the switch is normally at logic 1 (pulled up) and it
# goes to logic 0 when the button si pressed. The motor normally
# spins at ull speed and when the button is pressed teh speed falls
# by 25%/ Motor voltage hasPWM waveform that has 100% duty cycle
# when the switch is no pressed, and 25% duty cycle when the switch
# is pressed.
#
#
# File  : motor2.py
# Date  : August 2017
# Author: Dogan Ibrahim
#-----------------------------------------------------------------------
import RPi.GPIO as GPIO

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
Motor = 2
Button = 4

GPIO.setup(Motor, GPIO.OUT)
GPIO.setup(Button, GPIO.IN, pull_up_down=GPIO.PUD_UP)

#
# Generate PWM waveform at motor pin with a 100% duty cycle. When the
# switch is pressed, change the duty cycle to 25%. The frequency of the
# PWM waveform is set to 1000Hz
#
p = GPIO.PWM(Motor, 1000)
p.start(100)

while True:
  if GPIO.input(Button) == 0:
    p.ChangeDutyCycle(25)
  else:
    p.ChangeDutyCycle(100)
```

Figure 2.22 Python program listing

2.4 PROJECT 3 – Varying the Motor Speed

This is a again a very simple project where a small brushed DC motor is connected to the microcontroller through a power MOSFET transistor switch as in the previous project. In addition, a potentiometer is connected to one of the analog inputs of the microcontroller. In this project the speed of the motor is varied by moving the potentiometer arm.

2.4.1 Block Diagram

Figure 2.23 shows the block diagram of the project. A motor driver (MOSFET transistor) and a potentiometer are connected to the microcontroller.

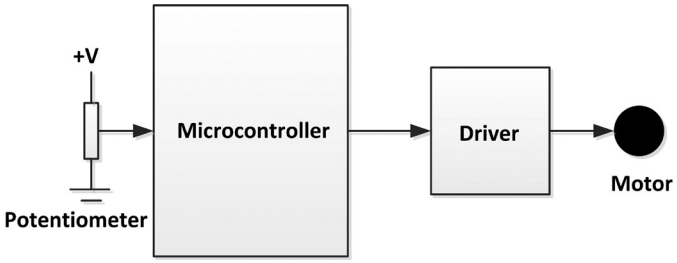

Figure 2.23 Block diagram of the project

The DC motor in this project is controlled using PWM waves as in the previous project. By varying the potentiometer arm the analog voltage read by the microcontroller is varied and this in turn changes the PWM duty cycle of the voltage applied to the motor, thus causing the motor speed to change. Arduino Uno ADC (analog-to-digital converter) is 10-bits wide and therefore it has 1024 steps (0 to 1023). The ADC reference voltage is +5V, making each step 4.9mV.

2.4.2 Circuit Diagram – Arduino Uno

The Arduino Uno circuit diagram of the project is shown in Figure 2.24. The MOSFET transistor is connected to GPIO port 3, and a 10K potentiometer is connected to analog input A0 of the Arduino Uno. Figure 2.25 shows the circuit built on a breadboard.

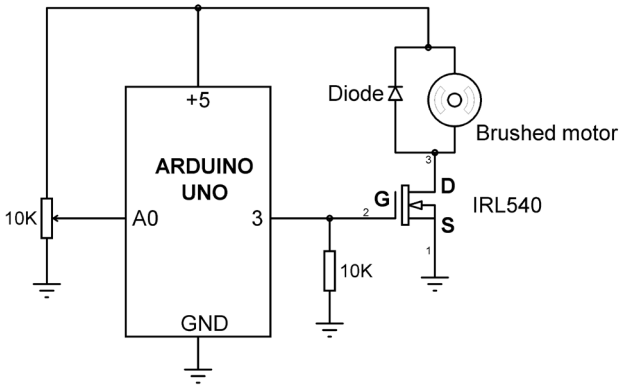

Figure 2.24 Arduino Uno circuit diagram

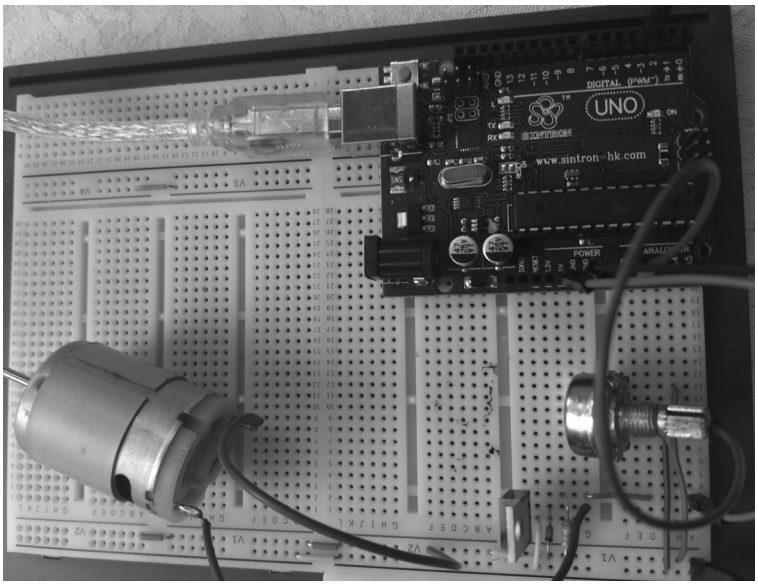

Figure 2.25 Circuit built on a breadboard

2.4.3 Circuit Diagram – Raspberry Pi

Raspberry Pi Zero W does not have any analog input ports. In this project we will be using an external analog-to-digital converter (ADC) chip to read the analog voltage of the potentiometer. The MCP3002 ADC chip is used in this project. This is a dual SPI based ADC converter having the following features:

- 10-bit resolution (0 to 1023 quantization levels)
- On-chip sample and hold
- SPI bus compatible
- Wide operating voltage (+2.7V to +5.5V)
- 75 Ksps sampling rate
- 5nA standby current, 50µA active current

The MCP3002 is a successive approximation 10-bit ADC with on-chip sample and hold amplifier. The device is programmable to operate as either differential input pair or as dual single-ended inputs. The device is offered in 8-pin package. Figure 2.26 shows the pin configuration of the MCP3002.

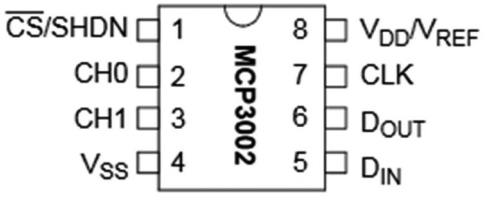

Figure 2.26 Pin configuration of the MCP3002

The pin definitions are as follows:

Vdd/Vref:	Power supply and reference voltage input
CH0:	Channel 0 analog input
CH1:	Channel 1 analog input
CLK:	SPI clock input
DIN:	SPI serial data in
DOUT:	SPI serial data out
CS/SHDN:	Chip select/shutdown input

In this project the supply voltage and the reference voltage are set to +3.3V. Thus, the digital output code is given by:

Digital output code = 1024 x Vin / 3.3
or, Digital output code = 310.30 x Vin

each quantization level corresponds to 3300mV/1024 = 3.22mV. Thus, for example, input data “00 0000001” corresponds to 3.22mV, “00 0000010” corresponds to 6.44mV and so on.

The MCP3002 ADC has two configuration bits: SGL/DIFF and ODD/SIGN. These bits follow the sign bit and are used to select the input channel configuration. The SGL/DIFF is used to select single ended or pseudo-differential mode. The ODD/SIGN bit selects which channel is used in single ended mode and is used to determine polarity in pseudo-differential mode. In this project we are using channel 0 (CH0) in single ended mode. According to the MCP3002 data sheet, SGL/DIFF and ODD/SIGN must be set to 1 and 0 respectively.

Figure 2.27 shows the circuit diagram of the project. The potentiometer is connected to CH0 of the ADC. CS, Dout, CLK, and Din pins of the ADC are connected to the SPI pins (see Appendix D) CE0 (pin 24), MISO (pin 21), SCLK (pin 23), and MOSI (pin 19) pins of the Raspberry Pi Zero W. Figure 2.28 shows the project constructed on a breadboard.

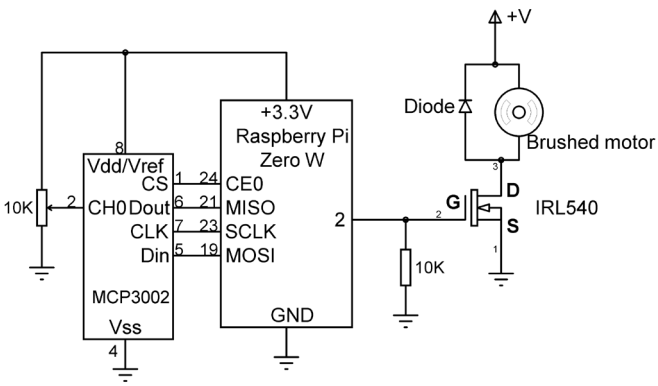

Figure 2.27 Raspberry Pi Zero W circuit diagram]

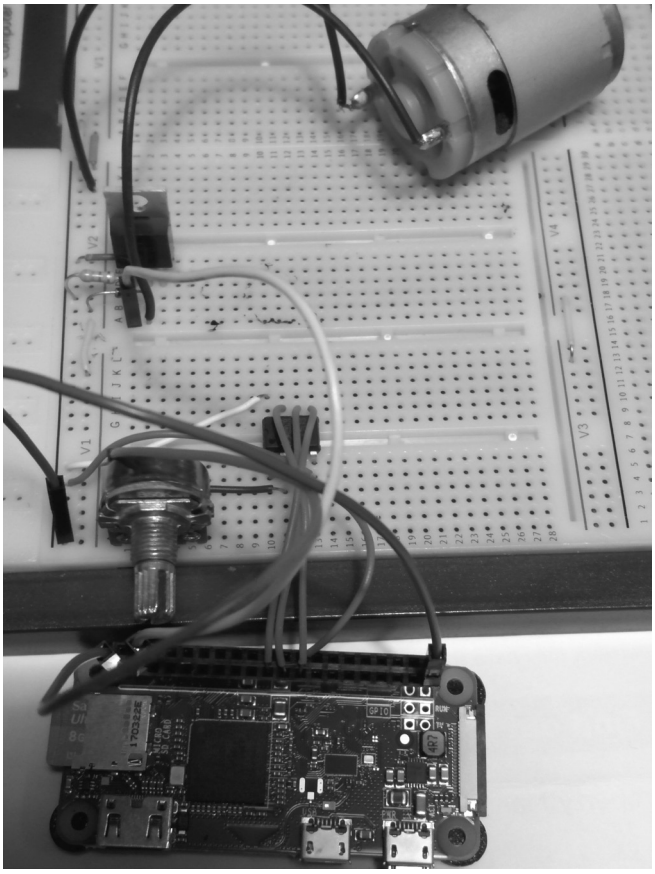

Figure 2.28 Raspberry Pi project on a breadboard

2.4.4 Program Listing – Arduino Uno

The frequency of the PWM voltage is set to 980.39 Hz as in the previous project. Figure 2.29 shows the program listing (program: Motor3). Compile the program as usual and upload to your Arduino Uno. At the beginning of the program port pins 3 and A0 are assigned to names Motor and Pot respectively. Inside the setup routine Motor is configured as digital output and Pot is configured as analog input.

Inside the program loop the analog voltage at the potentiometer arm is read as a digital value between 0 and 1023. Since the duty cycle numbers range from 0 to 255, the potentiometer reading is mapped to the duty cycle by dividing it by 4. For example, a potentiometer reading of 400 corresponds to a duty cycle number of 100 (which corresponds to 25% duty cycle). Similarly, a reading of 1023 approximately corresponds to 100% duty cycle.

```
/*****************************************************************
*              VARYING THE MOTOR CONTROL
*              ========================
*
* This is a very simple motor speed control. Pin 3 of the Arduino
* is connected to a small brushed DC motor through a power MOSFET
* switch. Also, analog pin A0 is connected to a potentiometer arm.
* The speed of the motor is varied by moving the potentiometer arm
*
*
* File  : Motor3
* Date  : August 2017
* Author: Dogan Ibrahim
*****************************************************************/
#define Motor 3                                 // Motor pin
#define Pot A0                                  // Button pin

//
// Configure pin 3 where motro is connected as output
//
void setup()
{
  pinMode(Motor, OUTPUT);                      // Motor is output
  TCCR2B = TCCR2B & B11111000 | B00000011;    // PWM freq = 980.39Hz
}


//
// Generate PWM waveform at motor pin with a varying duty cycle
// where the duty cycle is set by the potentiometer between 0
// and 255 (i.e. 0% to 100%)
//
void loop()
{
   int PotValue = analogRead(Pot);            // Read Pot value
   PotValue = PotValue / 4;                   // Convert to duty cycle
   analogWrite(Motor, PotValue);              // Set the duty cycle
}
```

Figure 2.29 Arduino program listing

2.4.5 Program Listing – Raspberry Pi

Figure 2.30 shows the Raspberry Pi Python program listing (program: motor3.py).

Function **get_adc_data** is used to read the analog data, where the channel number (channel_no) is specified in the function argument as 0 or 1. Notice that we have to send the start bit, followed by the SGL/DIFF and ODD/SIGN bits and the MSBF bit to the chip.

It is recommended to send leading zeroes on the input line before the start bit. This is often done when using microcontroller based systems that must send 8 bits at a time.

The following data can be sent to the ADC (SGL/DIFF = 1 and ODD/SIGN = channel_no) as bytes with leading zeroes for more stable clock cycle. The general data format is:

0000 000S DCM0 0000 0000 0000

Where, S = start bit, D = SGL/DIFF bit, C = ODD/SIGN bit, M = MSBF bit

For channel 0: 0000 0001 1000 0000 0000 0000 (0x01, 0x80, 0x00)

For channel 1: 0000 0001 1100 0000 0000 0000 (0x01, 0xC0, 0x00)

Notice that the second byte can be sent by adding 2 to the channel number (to make it 2 or 3) and then shifting 6 bits to the left as shown above to give 0x80 or 0xC0.

The chip returns 24 bit data (3 bytes) and we must extract the correct 10 bit ADC data from this 24 bit data. The 24 bit data is in the following format (“X” is don't care bit):

XXXX XXXX XXXX DDDD DDDD DDXX

Assuming that the returned data is stored in 24 bit variable ADC, we have:

ADC[0] = “XXXX XXXX”
ADC[1] = ”XXXX DDDD”
ADC[2] = “DDDD DDXX”

Thus, we can extract the 10 bit ADC data with the following operations:

(ADC[2] >> 2) so, low byte = “00DD DDDD”

and

(ADC[1] & 15) << 6) so, high byte = “DD DD00 0000”

Adding the low byte and the high byte we get the 10 bit converted ADC data as:

DD DDDD DDDD

The SPI bus on the Raspberry Pi supports the following functions:

Function	**Description**
open (0,0)	Open SPI bus 0 using CE0
open (0,1)	Open SPI bus 0 using CE1

close()	disconnect the device from the SPI bus
writebytes([array of bytes])	Write an array of bytes to SPI bus device
readbytes(len)	Read **len** bytes from SPI bus device
xfer2([array of bytes])	Send an array of bytes to the device with CEx asserted at all times
xfer([array of bytes])	Send an array of bytes de-asserting and asserting CEx with every byte transmitted

The module **spidev** must be imported at the beginning of the program before any of the above functions are called. Also, you must enable the SPI interface on your RPi ZW in the configuration menu. The steps are:

- Get into command mode (e.g. from Putty)
- Enter the following command:

  pi@raspberrypi:~ $ **sudo raspi-config**

- Select the Interface Options
- Enable SPI interface
- Finis and exit the configuration menu

At the beginning of the program in Figure 2.30 modules RPi.GPIO and spidev are imported to the program and an instance of the SPI is created. The program then configures port 2 where the motor is connected to as an output port. Function **get_adc_data** reads the analog voltage at the potentiometer arm and returns a digital value between 0 and 1023. Inside the main program loop the duty cycle of the PWM waveform is set based on the value read by the ADC. Duty cycle is set such that 0% corresponds to ADC value 0, and 100% corresponds to ADC value 1023. As a result, the speed of the motor changes as the potentiometer arm is moved.

```
#-----------------------------------------------------------------
#                          VARYING THE MOTOR SPEED
#                          =======================
# This is a very simple motor speed control. A brushed DC motor is
# connected to port pin 2 of the Raspberry Pi Zero W through a
# MOSFET switch. Also, a potentiometer is connected to the RPi ZW
# through the MCP3002 ADC. CH0 of the ADC is used in this project.
# The speed of the motor is varied by moving the potentiometer arm
#
```

```
#
# File  : motor3.py
# Date  : August 2017
# Author: Dogan Ibrahim
#-------------------------------------------------------------------------
import RPi.GPIO as GPIO
import spidev
#
# Create SPI instance and open the SPI bus
#
spi = spidev.SpiDev()
spi.open(0,0)       # We are using CE0 for CS

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
Motor = 2

GPIO.setup(Motor, GPIO.OUT)

#
# This function returns the ADC data read from the MCP3002
#
def get_adc_data(channel_no):
  ADC = spi.xfer2([1, (2 + channel_no) << 6, 0])
  rcv = ((ADC[1] & 15) << 6) + (ADC[2] >> 2)
  return rcv



#
# Generate PWM waveform at motor pin with a varying duty cycle
# where the duty cycle is set by the potentiometer. The frequency
# of the PWM waveform is set to 1000Hz. The MCP3002 ADC is a 10
# bit converter (0 to 1023 levels). The duty cycle of the PWM is
# changed from 0% to 100% proportional to the potentiometer value
#
p = GPIO.PWM(Motor, 1000)
p.start(100)

while True:
  adc = get_adc_data(0)
  Duty = adc * 100 / 1023
  p.ChangeDutyCycle(Duty)
```

Figure 2.30 Python program listing

2.5 PROJECT 4 – Changing the Motor Direction

This project shows how the direction of the motor can be changed. In this project a push-button switch is connected to the microcontroller. Pressing the button simply reverses the direction of the motor.

2.5.1 Block Diagram

Figure 2.31 shows the block diagram of the project. A *direction control driver circuit* is used to control the direction of the motor. The details of this circuit are explained below.

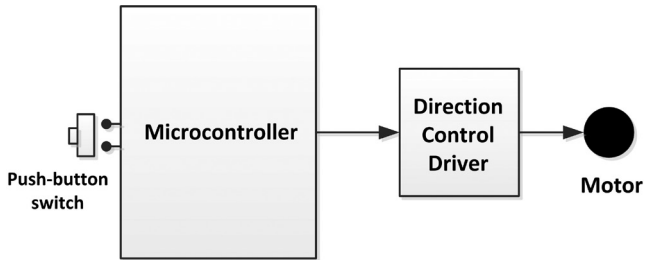

Figure 2.31 Block diagram of the project

The speed of a DC motor can be changed by changing the applied voltage level. The direction of rotation can be changed by simply reversing the polarity of the supply voltage. In DC motor direction control applications an H bridge circuit is used to change the polarity of the voltage applied to the motor and hence change the direction of rotation. Figure 2.32 shows the basic operation of an H bridge circuit. Here, we have 4 switches labelled A,B,C,D and the motor is connected in the middle of the circuit. By controlling the switches we can easily change the polarity of the voltage applied to the motor. For example, by clocking switches A and D and opening B and C, the motor will rotate clockwise. Similarly, by closing switches B and C and opening A and D the motor will rotate in anti-clockwise direction. This is illustrated below (0 and 1 correspond to switch open and close conditions respectively):

A	B	C	D	Motor rotation
0	0	0	0	No rotation
1	0	0	1	Clockwise
0	1	1	0	Anti-clockwise

It is clear from the above table that switches A and D and also B and C must be operated together.

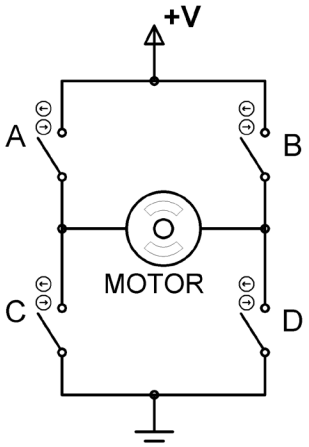

Figure 2.32 H bridge with switches

In real motor direction control applications the switches are replaced by transistors. Figure 2.33 shows an H bridge circuit built using bipolar transistors. In this circuit 4 signals are required to control the motor direction as explained above. A simpler circuit using only two control signals is shown in Figure 2.34. By using logic inverters we can make sure that only one transistor in either side of the motor is turned ON. For example, when A is set to logic 1, the transistor at the top left is turned ON and the one at the bottom left is OFF. NPN and PNP bipolar transistors can also be used in H bridge circuits requiring two control lines as shown in Figure 2.35.

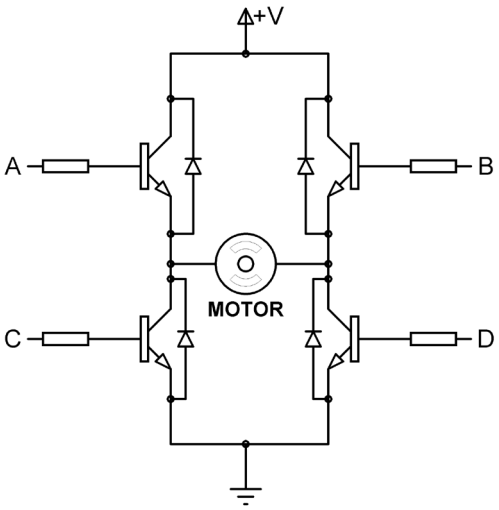

Figure 2.33 Simple H bridge circuit built using bipolar transistors

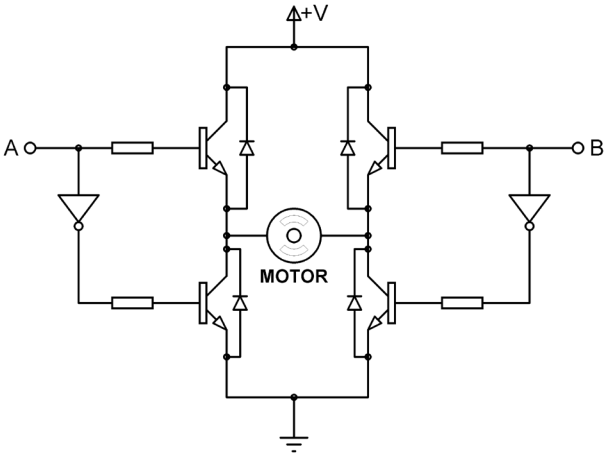

Figure 2.34 H bridge circuit using only 2 control signals

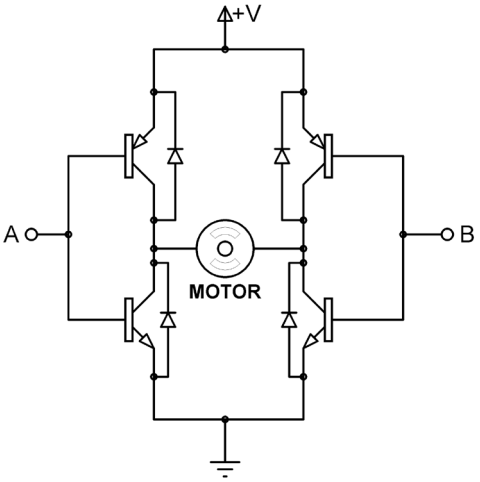

Figure 2.35 H bridge circuit with NPN and PNP transistors

An H bridge circuit built using power MOSFET transistors is shown in Figure 2.36. Notice that diodes are used in the H bridge circuits to protect the transistors from the back emf. Also, in a 4 wire control, A and C or B and D must not be enabled at the same time as this will short the power supply to the ground and possibly damage the bridge transistors.

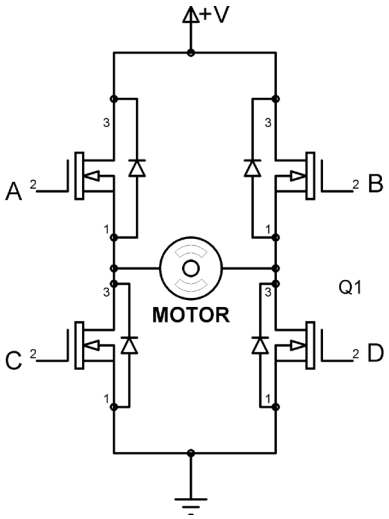

Figure 2.36 H bridge circuit built using MOSFET transistors

2.5.2 Circuit Diagram – Arduino Uno

The Arduino Uno circuit diagram of the project is shown in Figure 2.37. Here, port pins 0, 1, 2, 3 of Arduino are connected to H bridge pins A, B, C, D respectively. The push-button switch is connected to port pin 4 of Arduino.

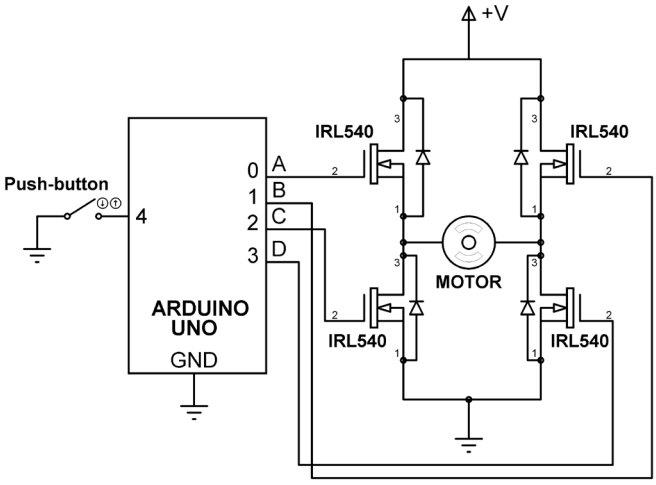

Figure 2.37 Arduino Uno circuit diagram

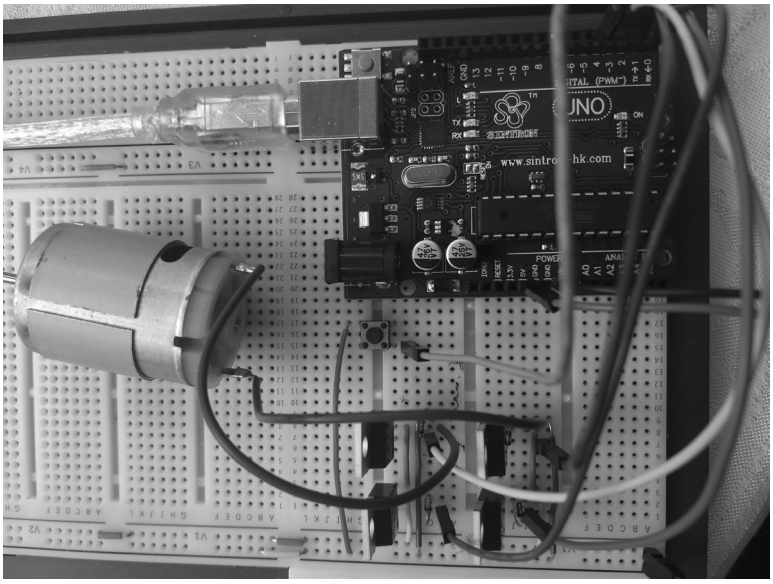

Figure 2.38 Circuit built on a breadboard

2.5.3 Circuit Diagram – Raspberry Pi

Figure 2.39 shows the circuit diagram of the project where pins 2,3,4, and 17 are connected to H bridge inputs A,B,C and D respectively. The push-button switch is connected to port pin 27. Pressing the button forces the button output to logic 0. The Raspberry Pi project built on a breadboard is shown in Figure 2.40.

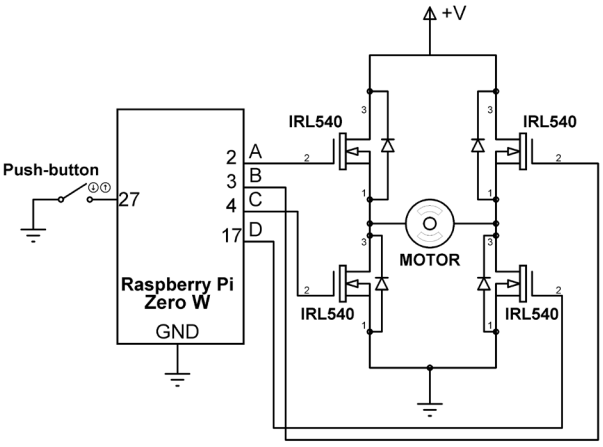

Figure 2.39 Raspberry Pi circuit diagram

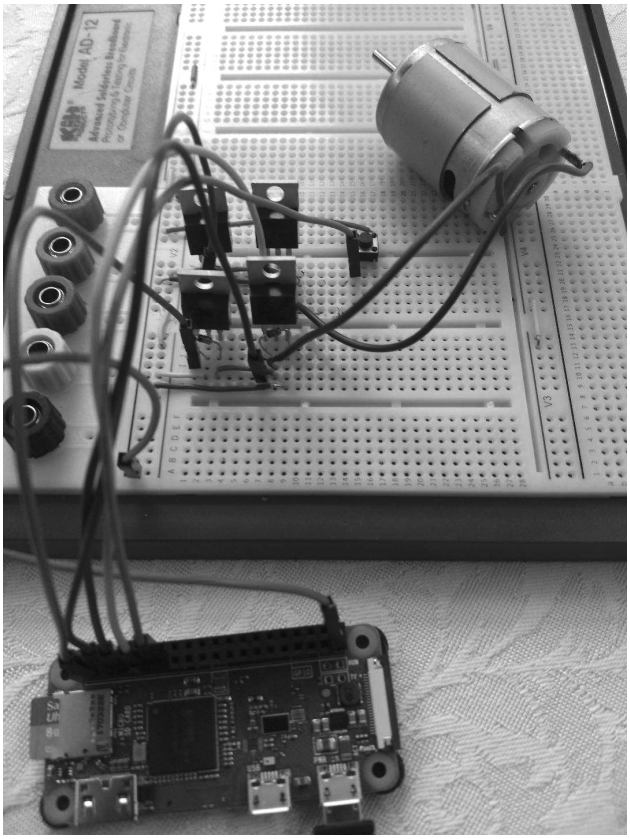

Figure 2.40 Raspberry Pi project on a breadboard

2.5.4 Program Listing – Arduino Uno

The program listing is shown in Figure 2.41 (program: Motor4). At the beginning of the program A, B, C and D are assigned to port pins 0,1,2, and 3 respectively. Also, Button is assigned to port pin 4. In the setup routine A,B,C,D are configured as outputs and Button is configured as input with pull-up option so that the state of the button is normally at logic 1 and goes to 0 when the button is pressed. Also, the motor is set to be idle.

Inside the program loop the state of the button is checked and if the button is pressed (Button = 0) the motor is spinned anti-clockwise by calling function **Anticlockwise**. If on the other hand the button is not pressed (Button = 1) then function **Clockwise** is called to spin the motor clockwise.

In a clockwise rotation MOSFETS at pins A and D are turned ON and the other two are turned OFF. Similarly, in an anticlockwise rotation MOSFETS at pins B and C are turned ON and the other two are turned OFF.

```
/*****************************************************************
*              MOTOR DIRECTION CONTROL
*              ======================
*
* In this project the DC motor is connected to the Arduino Uno
* through a MOSFET based H bridge. In addition, a push-button
* switch is connected to port pin 4 of the Arduino Uno. Pressing
* the button changes the direction of rotation.
*
* The H bridge is controlled from port pins 0,1,2,3 of the Arduino
*
*
* File  : Motor4
* Date  : August 2017
* Author: Dogan Ibrahim
*****************************************************************/
#define A 0                                     // H bridge A input
#define B 1                                     // H bridge B input
#define C 2                                     // H bridge C input
#define D 3                                     // H bridge D input
#define Button 4                                // Button input

//
// Configure H bridge pins as outputs and the Button pin as input
//
void setup()
{
  pinMode(A, OUTPUT);                           // A is output
  pinMode(B, OUTPUT);                           // B is output
  pinMode(C, OUTPUT);                           // C is output
  pinMode(D, OUTPUT);                           // D is output
  pinMode(Button, INPUT_PULLUP);                // Button is input
  StopMotor();                                  // Stop the motor
}


//
// This function stops the motor
//
void StopMotor()
{
  digitalWrite(A, LOW);                         // A = 0
  digitalWrite(B, LOW);                         // B = 0
  digitalWrite(C, LOW);                         // C = 0
  digitalWrite(D, LOW);                         // D = 0
}
```

```
//
// This function turns the motor clockwise
//
void Clockwise()
{
  digitalWrite(B, LOW);
  digitalWrite(C, LOW);
  digitalWrite(A, HIGH);
  digitalWrite(D, HIGH);
}

//
// This function turns the motor anticlockwise
//
void Anticlockwise()
{
  digitalWrite(A, LOW);
  digitalWrite(D, LOW);
  digitalWrite(B, HIGH);
  digitalWrite(C, HIGH);
}


//
// Motor spins in one direction. When the button is pressed the
// motor spins in the reverse direction
//
void loop()
{
   if(digitalRead(Button) == 1)
     Clockwise();
   else
     Anticlockwise();
}
```

Figure 2.41 Arduino program listing

2.5.5 Program Listing – Raspberry Pi

Figure 2.42 shows the Raspberry Pi Python program listing. At the beginning of the program the GPIO library is imported and names A,B,C,D are assigned to H bridge gate inputs which are connected to RPi ZW ports 2,3,4,17 respectively. Port pin 27 is connected to the push-button.

Inside the program loop the state of the button is checked and if the button is pressed (Button = 0) the motor is spinned anti-clockwise by calling function **Anticlockwise**. If on

the other hand the button is not pressed (Button = 1) then function **Clockwise** is called to spin the motor clockwise.

In a clockwise rotation MOSFETS at pins A and D are turned ON and the other two are turned OFF. Similarly, in an anticlockwise rotation MOSFETS at pins B and C are turned ON and the other two are turned OFF.

```
#                          MOTOR DIRECTION CONTROL
#                          ======================
#
# In this project the DC motor is connected to the RPi ZW through a
# MOSFET based H bridge. In addition, a push-button switch is connected
# to port pin 27 of the RPi. Pressing the button changes teh direction
# of rotation.
#
# The H bridge is connected to pins 2,3,4,17 of the RPi
#
# File  : motor4.py
# Date  : August 2017
# Author: Dogan Ibrahim
#-------------------------------------------------------------------------
import RPi.GPIO as GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
A = 2
B = 3
C = 4
D = 17
Button = 27

GPIO.setup(A, GPIO.OUT)
GPIO.setup(B, GPIO.OUT)
GPIO.setup(C, GPIO.OUT)
GPIO.setup(D, GPIO.OUT)
GPIO.setup(Button, GPIO.IN, pull_up_down=GPIO.PUD_UP)

#
# This function stops the motor
#
def StopMotor():
  GPIO.output(A, 0)
  GPIO.output(B, 0)
  GPIO.output(C, 0)
  GPIO.output(D, 0)
  return
```

```
#
#This function turns the motor clockwise
#
def Clockwise():
  GPIO.output(B, 0)
  GPIO.output(C, 0)
  GPIO.output(A, 1)
  GPIO.output(D, 1)
  return

#
# This function turns the motor anticlockwise
#
def Anticlockwise():
  GPIO.output(A, 0)
  GPIO.output(D, 0)
  GPIO.output(B, 1)
  GPIO.output(C, 1)
  return
#
# Motor spins in one direction. When the button is pressed the motor
# spins in the reverse direction
#
StopMotor()

while True:
  if GPIO.input(Button) == 1:
    Clockwise()
  else:
    Anticlockwise()
```

Figure 2.42 Python program listing

2.6 PROJECT 5 – Using H Bridge Integrated Circuit

This project shows how the direction of the motor can be changed easily by using an H bridge integrated circuit. In this project a push-button switch is connected to the microcontroller. Pressing the button simply reverses the direction of the motor.

2.6.1 Block Diagram

The block diagram of the project is as in Figure 2.31. In this project the L293 (From Texas Instruments. SN754410 is a similar integrated circuit) H bridge integrated circuit is used. The chip has two H bridges, one on the left side of the chip and one on the right side, and therefore it can drive two motors independently. The chip operating voltage is 4.5V to 36V and it can drive up to 1 Ampere of current (L293D can supply up to 600mA of current).

Figure 2.43 shows the pin configuration of L293. The pins have the following functions:

Pin 1 (1,2EN): motor enable pin
Pin 2,7 (1A,2A): motor logic control
Pin 3,6 (1Y,2Y): motor drive pins
Pin 4,5 (GND): power supply ground
Pin 12,13 (GND): power supply ground
Pin 16 (VCC1): VCC1 power supply for the chip (4.5 to 7V)
Pin 8 (VCC2): VCC1 to 36V motor voltage

The pins on the right hand side of the chip have similar functions for another motor. Table 2.1 shows the motor control logic (for one motor only):

EN	1A	2A	Function
1	0	1	Turn right
1	1	0	Turn left
1	0	0	Fast motor stop
1	1	1	Fast motor stop
0	X	X	Fast motor stop

Table 2.1 Motor control logic

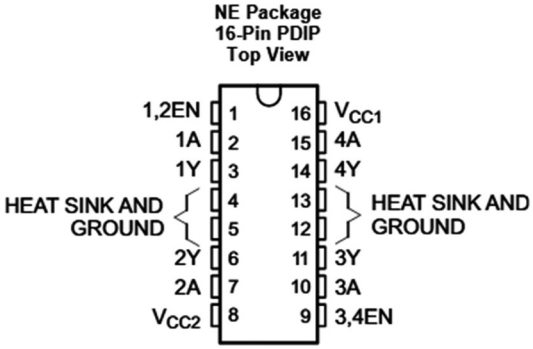

Figure 2.43 Pin configuration of the L293

2.6.2 Circuit Diagram – Arduino Uno

Figure 2.44 shows the circuit diagram of the project. Since there is only one motor, the pins on the left hand side of the chip are used only. The motor is enabled always by connecting pin EN to logic 1 (+5V). Control pins 1A and 2A are connected to ports 0 and 1 of the Arduino. The push-button switch is connected to port pin 4. The motor is connected across pins 3 (1Y) and 6 (2Y) of the L293 chip and an external 6V power supply is used to power the motor through pin 8 (VCC2) of the L293. Figure 2.45 shows the circuit built on a breadboard.

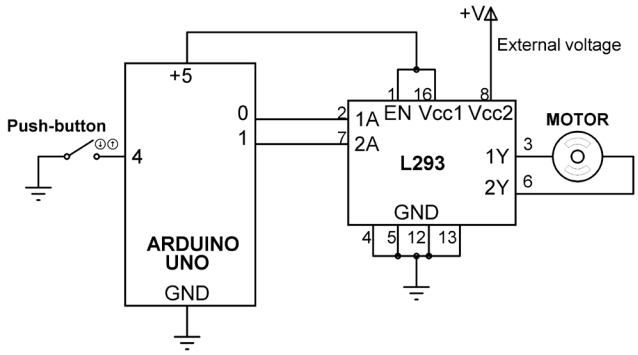

Figure 2.44 Arduino Uno circuit diagram

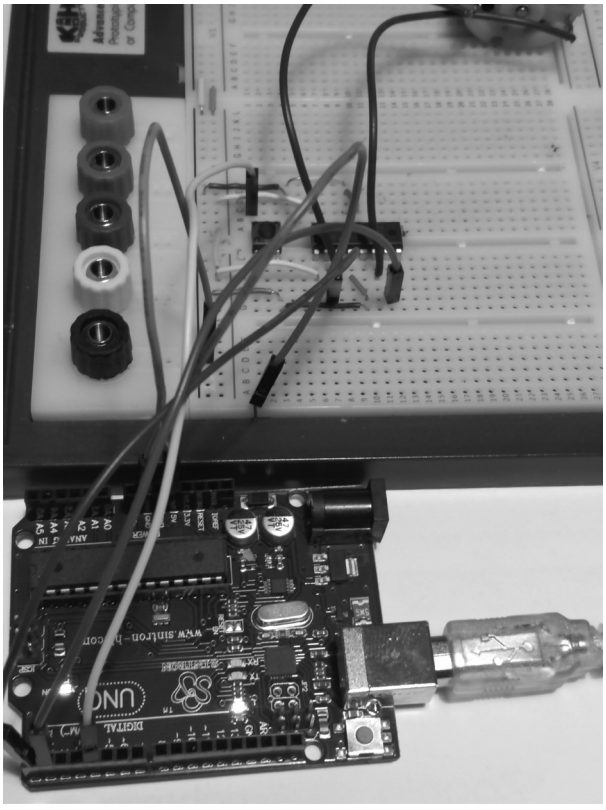

Figure 2.45 Arduino Uno circuit built on a breadboard

2.6.3 Circuit Diagram – Raspberry Pi

Figure 2.46 shows the Raspberry Pi Zero W circuit diagram of the project. As with the Arduino Uno circuit, only the pins on the left hand side of L293 are used. Control pins 1A and 2A are connected to RPi ZW port pins 2 and 3 respectively. The button is connected to port pin 4 and is pulled-up in software. The motor is powered from an external power supply connected to pin 8 of the L293 chip. Figure 2.47 shows the circuit built on a breadboard.

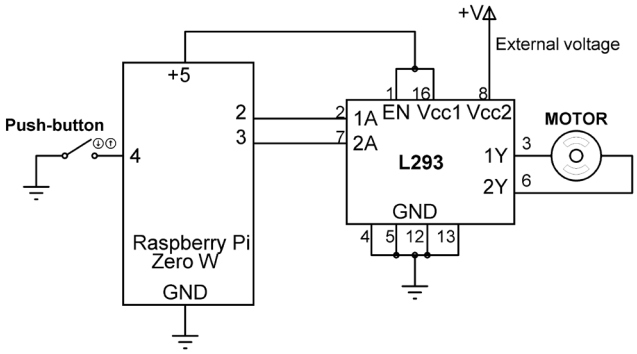

Figure 2.46 Raspberry Pi circuit diagram

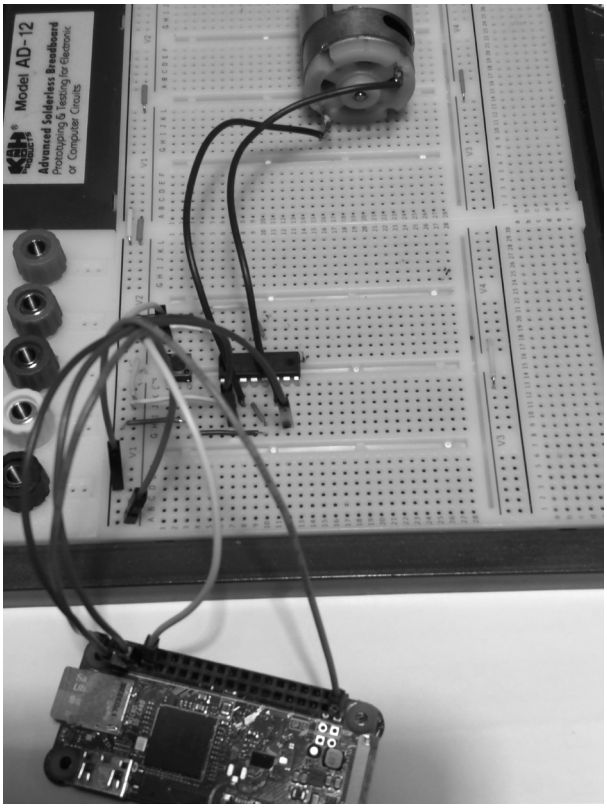

Figure 2.47 Raspberry Pi circuit built on a breadboard

2.6.4 Program Listing – Arduino Uno

The program listing is shown in Figure 2.48 (program: motor5). At the beginning of the program A1 and A2 are assigned to port pins 0 and 1 respectively, and Button is assigned to pin 4. Inside the setup routine port pins 0 and 1 are configured as outputs and pin 4 is configured as input with pull-up resistor. The motor is stopped to start with by clearing 1A and 2A pins of L293.

Function **Clockwise** sets 1A to 0 and 2A to 1 so that the motor rotates clockwise. Similarly, function **Anticlockwise** sets 1A to 1 and 2A to 0 so that the motor rotates anticlockwise. Inside the main program loop the state of Button is checked. If Button is 1 (i.e. the button is not pressed) then the motor spins clockwise. If on the other hand the Button is pressed then the motor spins anticlockwise.

```
/*****************************************************************
*              MOTOR DIRECTION CONTROL - H BRIDGE IC
*              ====================================
*
* In this project the DC motor is connected to the Arduino Uno
* through the integrated circuit L293 H bridge. In addition, a push-button
* switch is connected to port pin 4 of the Arduino Uno. Pressing
* the button changes the direction of rotation.
*
* Arsduino Port pins 0 and 1 are connected to L293 pins 1A and 2A
* respectively. Motor roates one direction when 1A=0, 2A =1, and
* reverse direction when 1A = 1, 2A = 0. Power to the motor is
* supplied externally.
*
* File  : Motor5
* Date  : August 2017
* Author: Dogan Ibrahim
*****************************************************************/
#define A1 0                                    // H bridge 1A input
#define A2 1                                    // H bridge 2A input
#define Button 4                                // Button input

//
// Configure H bridge pins as outputs and the Button pin as input
//
void setup()
{
  pinMode(A1, OUTPUT);                          // 1A is output
  pinMode(A2, OUTPUT);                          // 2A is output
  pinMode(Button, INPUT_PULLUP);                // Button is input
  StopMotor();                                  // Stop the motor
}

//
// This function stops the motor
//
void StopMotor()
{
  digitalWrite(A1, LOW);                         // 1A = 0
  digitalWrite(A2, LOW);                         // 2A = 0
```

```
}
//
// This function turns the motor clockwise
//
void Clockwise()
{
  digitalWrite(A1, LOW);
  digitalWrite(A2, HIGH);
}

//
// This function turns the motor anticlockwise
//
void Anticlockwise()
{
  digitalWrite(A1, HIGH);
  digitalWrite(A2, LOW);
}
//
// Motor spins in one direction. When the button is pressed the
// motor spins in the reverse direction
//
void loop()
{
   if(digitalRead(Button) == 1)
     Clockwise();
   else
     Anticlockwise();
}
```

Figure 2.48 Arduino program listing

2.6.5 Program Listing – Raspberry Pi

The Raspberry Pi Zero W program listing of the project is shown in Figure 2.49 (program: motor5.py). At the beginning of the program RPi.GPIO library is imported, A1 and A2 are assigned to motor pins 1A and 2A respectively, and Button is assigned to port pin 4. A1 and A2 are then configured as outputs and Button is configured as input with pull-up resistor. Function **Clockwise** spins the motor clockwise by setting 1A to 0 and 2A to 1. Similarly, function **Anticlockwise** spins the motor anticlockwise by setting 1A to 1 and 2A to 0. Inside the main program loop the state of Button is checked and the motor is turned clockwise if the Button is not pressed, otherwise it is turned anticlockwise.

```
#-----------------------------------------------------------------
#                     MOTOR DIRECTION CONTROL - H BRIDGE IC
#                     ====================================
#
# In this project the DC motor is connected to the RPi ZW through a
```

```
# L293 H bridge integrated circuit. In addition, a push-button switch
# is connected to port pin 4 of the RPi. Pressing the button changes
# the direction of rotation.
#
# L293 pins 1A and 2A are connected to RPi ZW pins 2 and 3 and the
# motor is connected to pins 1Y and 2Y of the chip. The motor is
# powered using an external power supply, connected to pin 8 of the
# L293 chip
#
# File  : motor5.py
# Date  : August 2017
# Author: Dogan Ibrahim
#-------------------------------------------------------------------------
import RPi.GPIO as GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
A1 = 2
A2 = 3
Button = 4

GPIO.setup(A1, GPIO.OUT)
GPIO.setup(A2, GPIO.OUT)
GPIO.setup(Button, GPIO.IN, pull_up_down=GPIO.PUD_UP)

#
# This function stops the motor
#
def StopMotor():
  GPIO.output(A1, 0)
  GPIO.output(A2, 0)
  return

#
#This function turns the motor clockwise
#
def Clockwise():
  GPIO.output(A1, 0)
  GPIO.output(A2, 1)
  return

#
# This function turns the motor anticlockwise
#
def Anticlockwise():
  GPIO.output(A1, 1)
  GPIO.output(A2, 0)
```

```
  return
#
# Motor spins in one direction. When the button is pressed the motor
# spins in the reverse direction
#
StopMotor()

while True:
  if GPIO.input(Button) == 1:
    Clockwise()
  else:
    Anticlockwise()
```

Figure 2.49 Raspberry Pi program listing

2.7 PROJECT 6 – Motor Speed and Direction Control Using H Bridge Integrated Circuit

This project shows how the speed and direction of a DC motor can be changed easily by using an H bridge motor controller integrated circuit. In this project the motor is controlled as follows:

- Turn clockwise for 5 seconds at high speed
- Wait 2 seconds
- Turn clockwise for 5 second at low speed
- Wait 2 seconds
- Turn anticlockwise for 5 seconds at high speed
- Wait 2 seconds
- Turn anticlockwise for 5 seconds at low speedThe above process is repeated indefinitely until stopped manually.

2.7.1 Block Diagram

The block diagram of the project is shown in Figure 2.50. In this project the LMD18200 H bridge motor controller integrated circuit is used. The chip has a built-in H bridge where the speed and the direction of rotation of a motor can be controlled. The speed is controlled by sending PWM waves to the chip, and the direction of rotation is controlled by setting or clearing a direction control bit. LMD18200 has the following features:

- Up to 3A continuous current
- Supply voltage up to 55V
- TTL and CMOS compatible inputs
- Motor speed and direction control
- Thermal shutdown
- Motor stop (break) input

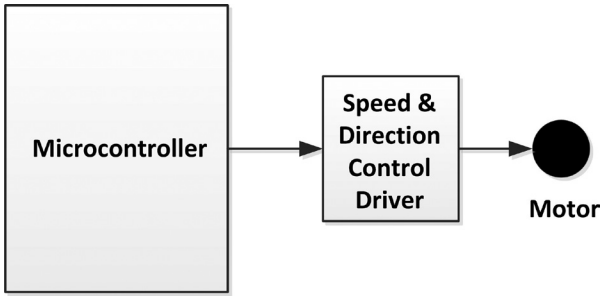

Figure 2.50 Block diagram of the project

Figure 2.51 shows the pin configuration of LMD18200. The pins have the following functions:

Pin 1 (Bootstrap input): Bootstrap capacitor input. A 10 nF capacitor should be connected to this input
Pin 2 (Output): Motor connection
Pin 3 (Direction input): Controls the direction of the motor
Pin 4 (Brake input): This input is used to break a motor by shorting its terminals
Pin 5 (PWM input): Speed control input
Pin 6 (Vs): Power supply
Pin 7 (Ground): Power supply ground
Pin 8 (Current sense output): Sourcing current sensing output
Pin 9 (Thermal flag output): Thermal warning flag output
Pin 10 (Output): Motor connection
Pin 11 (Bootstrap 2 input): Bootstrap capacitor input. A 10 nF capacitor should be connected to this input

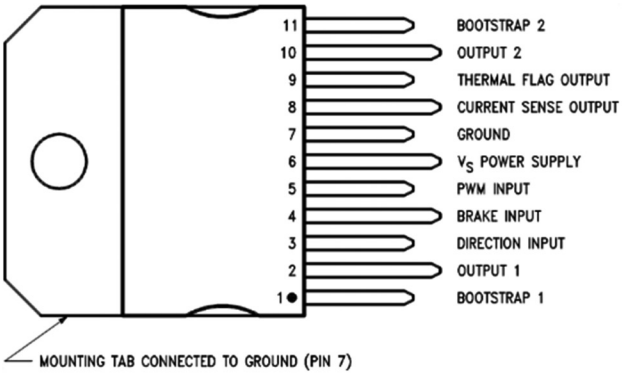

Figure 2.51 Pin configuration of LMD18200

LMD18200 controller chip is available as a low-cost small module (see Figure 2.52) with an on-board heatsink, capacitors, and two screw terminal type connectors for making connection to a microcontroller and a motor. The module has the following screw pin configuration:

Left Screw Terminal (J2)

GND: Microcontroller power supply ground
PWM: Speed control input (at logic levels)
DIR: Direction control input
BRAKE: Brake control input (logic 1 to brake)

Right Screw Terminal (J3)

V+: External motor power supply
GND: Power supply ground
OUT1: Motor connection
OUT2: Motor connection

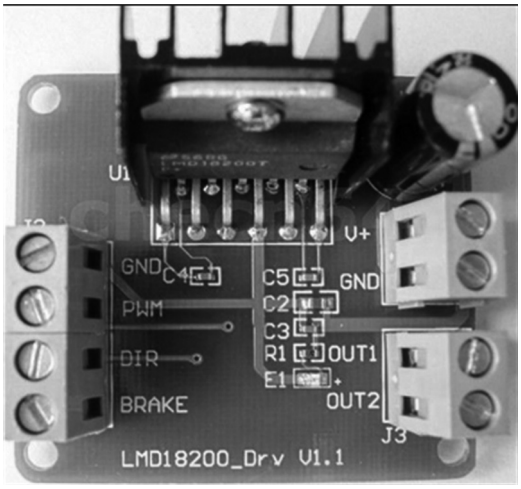

Figure 2.52 LMD18200 module

Table 2.2 shows the operational logic table of the LMD18200.

PWM	DIR	BRAKE	Active output drivers
1	1	0	Rotate in one direction
1	0	0	Rotate in reverse direction
1	1	1	Break
1	0	1	Break
0	X	1	Break

Table 2.2 LMD18200 operational logic table

2.7.2 Circuit Diagram – Arduino Uno

The LMD18200 module is used in this project. Figure 2.53 shows the circuit diagram of the project. The motor is connected to screw terminals OUT1 and OUT2 and an external +12V power supply is connected to the V+ input. GPIO port pins 0 and 3 are connected to the DIR and PWM inputs of the LMD18200 module. Figure 2.54 shows the circuit built using jumpers

to connect the Arduino Uno to the screw terminals on the LMD18200 module.

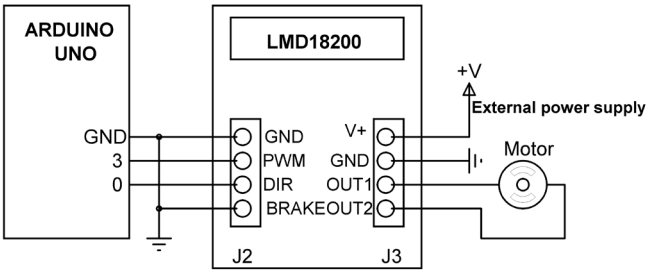

Figure 2.53 Arduino Uno circuit diagram

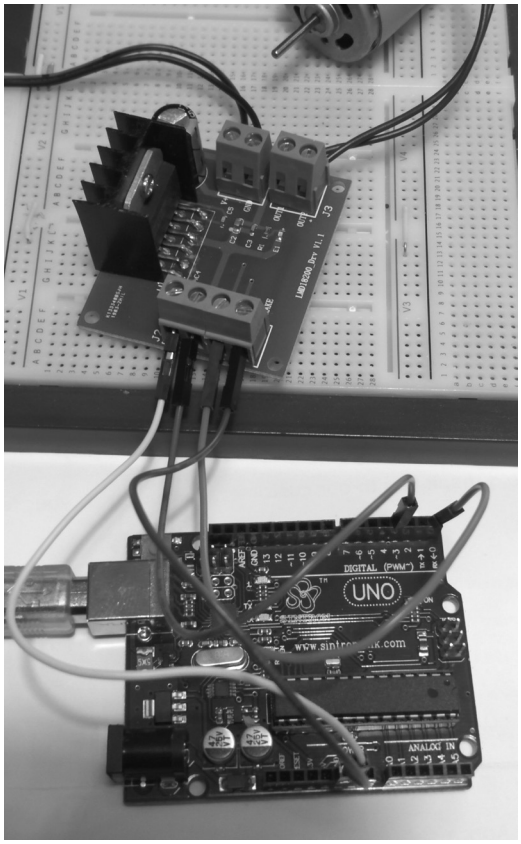

Figure 2.54 Arduino Uno connection to the LMD18200 module

2.7.3 Circuit Diagram – Raspberry Pi

Figure 2.55 shows the circuit diagram for the Raspberry Pi version of the project. The LMD18200 module is used here as well. Port pins 2 and 3 are connected to the DIR and PWM inputs of the LMD18200 module. Motor is powered from an external +12V power supply. Figure 2.56 shows the circuit built using jumpers to connect the Raspberry Pi Zero

W to the screw terminals on the LMD18200 module.

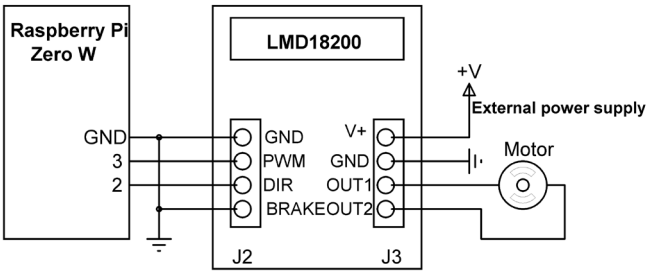

Figure 2.55 Raspberry Pi circuit diagram

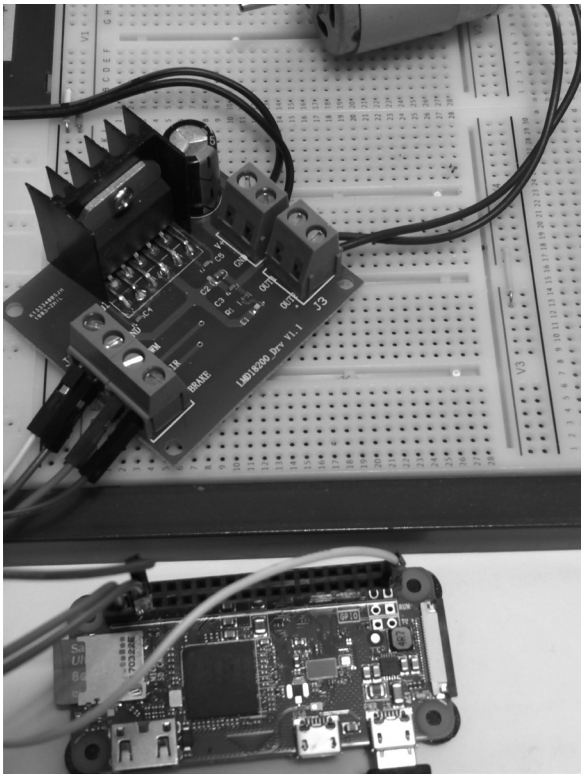

Figure 2.56 Raspberry Pi connection to the LMD18200 module

2.7.4 Program Listing – Arduino Uno

The program listing is shown in Figure 2.57 (program: motor6). At the beginning of the program DIR and PWM are assigned to port pins 0 and 3 respectively. Inside the setup routine the ports are configured as output and the PWM frequency is set to nearly 1 kHz (980.39 Hz). Function **Rotate** has two character arguments: **direction** and **speed**. If the direction is **c** (i.e. clockwise) then the DIR pin of the LMD18200 module is set LOW. If on the other hand the direction is **a** (i.e. anticlockwise) then the DIR pin of the LMD18200 module

is set HIGH. If the speed is **f** (i.e. fast) then PWM voltage with 100% (i.e. a value of 255) duty cycle is sent to the PWM pin. If on the other hand the speed is **s** (i.e. slow) then the PWM duty cycle is set to 25% (i.e. a value of 64)

```
/*****************************************************************
*         MOTOR SPEED AND DIRECTION CONTROL USING AN IC
*         =============================================
*
* This is a simple motor speed and direction control project
* using the LMD18200 motor control module. Pins 0 and 3 of the
* Arduino Uno are connected to pins DIR and PWM of the LMD18200
* respectively. A +12V external power supply is used to power the
* motor.
*
*
*
* File  : Motor6
* Date  : August 2017
* Author: Dogan Ibrahim
*****************************************************************/
#define DIR 0                                  // DIR pin
#define PWM 3                                  // PWM pin

//
// Configure DIR as output
//
void setup()
{
  pinMode(DIR, OUTPUT);                         // Direction is output
  pinMode(PWM, OUTPUT);                         // PWM is output
  TCCR2B = TCCR2B & B11111000 | B00000011;    // PWM freq = 980.39Hz
}

//
// This function rotates the motor. Argument direction can be 'c'
// for clockwise or 'a' for anticlockwise. Argument speed can be
// 'f' for fast or 's' for slow
//
void Rotate(char direction, char speed)
{
  if(direction == 'c')
    digitalWrite(DIR, LOW);
  else
    if(direction == 'a')digitalWrite(DIR, HIGH);

  if(speed == 'f')
```

```
    analogWrite(PWM, 255);
  else
    if(speed == 's')analogWrite(PWM, 64);
}

//
// The motor is rotated as follows:
// 5 seconds clockwise and fast
// 5 seconds clockwise and slow
// 5 seconds anticlockwise and fast
// 5 seconds anticlockwise and slow
//
void loop()
{
     Rotate('c', 'f');                    // clockwise, fast
     delay(5000);
     Rotate('c', 's');                    // clockwise, slow
     delay(5000);
     Rotate('a', 'f');                    // anticlockwise, fast
     delay(5000);
     Rotate('a', 's');                    // anticlockwise, slow
     delay(5000);
}
```

Figure 2.57 Arduino Uno program listing

2.7.5 Program Listing – Raspberry Pi

The program listing is shown in Figure 2.58 (program: motor6.py). At the beginning of the program RPi.GPIO and time modules are imported to the program. DIR and PWM are assigned to port pins 2 and 3 respectively and these pins are configured as outputs. Function **Rotate** has two character arguments: **direction** and **speed**. If the direction is **c** (i.e. clockwise) then the DIR pin of the LMD18200 module is set LOW. If on the other hand the direction is **a** (i.e. anticlockwise) then the DIR pin of the LMD18200 module is set HIGH. If the speed is **f** (i.e. fast) then PWM voltage with 100% (i.e. a value of 100) duty cycle is sent to the PWM pin. If on the other hand the speed is **s** (i.e. slow) then the PWM duty cycle is set to 25% (i.e. a value of 25). 5 seconds of delay is inserted between each output.

```
#-------------------------------------------------------------------
#                MOTOR SPEED AND DIRECTION CONTROL USING AN IC
#                ===================== ======================
#
# This is a simple motor speed and direction control project
# using the LMD18200 motor control module. Pins 2 and 3 of the
# Raspberry Pi Zero W are connected to pins DIR and PWM of the
# LMD18200 respectively. A +12V external power supply is used
# to power the motor.
```

```
#
#
# File  : motor6.py
# Date  : August 2017
# Author: Dogan Ibrahim
#-------------------------------------------------------------------------
import RPi.GPIO as GPIO
import time

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
DIR = 2
PWM = 3

GPIO.setup(DIR, GPIO.OUT)
GPIO.setup(PWM, GPIO.OUT)

#
# Generate PWM waveform at PWM pin with frequency of 1000 Hz
#
p = GPIO.PWM(PWM, 1000)
p.start(100)

#
# This function rotates the motor with the given direction
# and speed
#
def Rotate(direction, speed):
  if direction == 'c':
    GPIO.output(DIR, 0)
  else:
    if direction == 'a':
      GPIO.output(DIR, 1)

  if speed == 'f':
   p.ChangeDutyCycle(100)
  else:
   if speed == 's':
     p.ChangeDutyCycle(25)
  return

while True:
  Rotate('c', 'f')
  time.sleep(5)
  Rotate('c', 's')
  time.sleep(5)
```

```
Rotate('a', 'f')
time.sleep(5)
Rotate('a', 's')
time.sleep(5)
```

Figure 2.58 Raspberry Pi program listing

2.8 PROJECT 7 – Using a Rotary Encoder - Displaying the Speed of a Motor (Arduino Uno)

This project shows how a rotary encoder can be used to sense the speed of a motor. The speed is then displayed on the Serial Monitor of the Arduino IDE.

2.8.1 Block Diagram

The block diagram of the project is shown in Figure 2.59. In this project a small geared brushed DC motor with a built-in rotary encoder is used (Pololu's motor and encoder). Figure 2.60 shows a picture of the motor where the rotary encoder is attached to the back shaft of the motor.

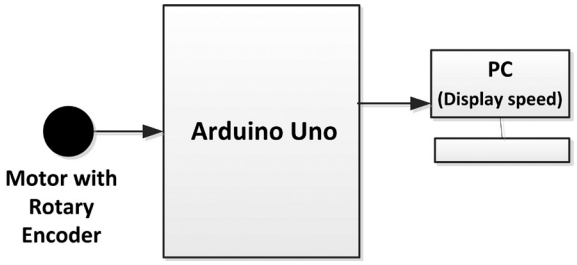

Figure 2.59 Block diagram of the project

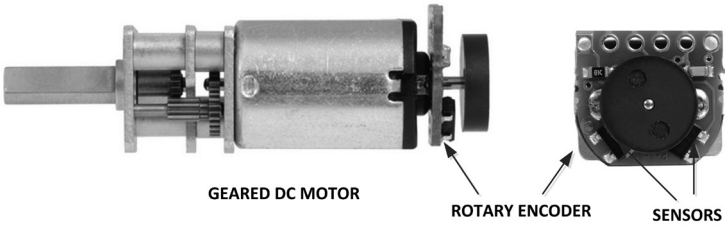

Figure 2.60 Picture of the motor used (Pololu.com)

A rotary encoder (or a shaft encoder) is a device that converts the angular position of a rotating device such as a motor into an electrical signal. Rotary encoders are used in motor control applications to sense the speed of the motor or the position of the motor shaft. There are basically two types of rotary encoders: optical and Hall Effect. Optical rotary encoders work with optical principles where light shines onto a photodiode through slits (or holes) in a metal or any other form of disc. By counting the number of holes passing in-front of the photodiode in a given time we can calculate the speed of the motor.

In this project a Hall Effect based rotary encoder is used. Two static magnetic sensors (called Phase A and Phase B) are used on the encoder board together with a rotating magnetic disc. The sensors give out pulses as the motor is rotated. By counting the pulses in a given time we can calculate the speed of the motor. Figure 2.61 shows the output waveform of the encoder used in this project. These types of encoders are also known as quadrature encoders where there are four possible output states. The direction of rotation is easily determined by finding the order that the output signals change from 0 to 1. For example, if the Phase A signal rises when Phase A signal is LOW then the motor rotates in one direction. If on the other hand Phase B signal rises when Phase A signal is LOW then the motor rotates in the opposite direction (see Figure 2.62).

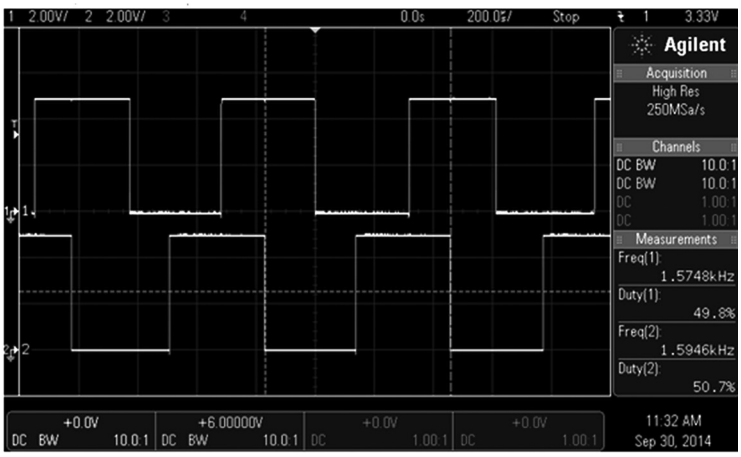

Figure 2.61 Output waveforms of the rotary encoder (Pololu.com)

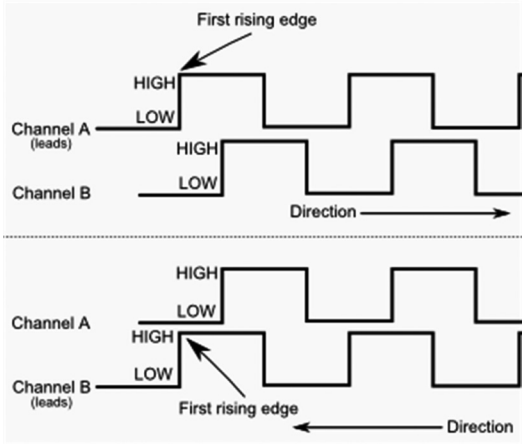

Figure 2.62 Determining the direction of rotation (www.robotoid.com)

The specifications of the motor used in this project are as follows (depends on the voltage applied):

- 6V DC operation
- Shaft speed 15000 RPM (at 6V)
- 380:1 gear ratio
- Speed after gears 41 RPM (at 6V)
- Motor torque 2.5Kg.cm
- Current 170 mA
- Built in Hall Effect with two sensors
- Disc with 7 pole pairs
- Hall Effect resolution (7 x 2 x 380 = 5320)
- Weight 18 grams

The speed of the motor is easily found by counting the number of pulses either from one or from both encoders in a given time. For example, if only the rising edges of phase A are counted every second then the speed of the motor shaft after the gears in RPM can be calculated by dividing this count by the Hall Effect resolution and multiplying by 60 (60 seconds in a minute). i.e.

Motor shaft speed (after gears) = Pulses * 60 / 5320

2.8.2 Circuit Diagram

The pin configuration of the motor is shown in Figure 2.63 and is as follows:

Red cable:	Motor + power supply
Black cable:	Encoder +3.3V or +5V power supply
Yellow cable:	Encoder A Phase
Green cable:	Encoder B Phase
Blue cable:	Encoder ground
White cable:	Motor ground

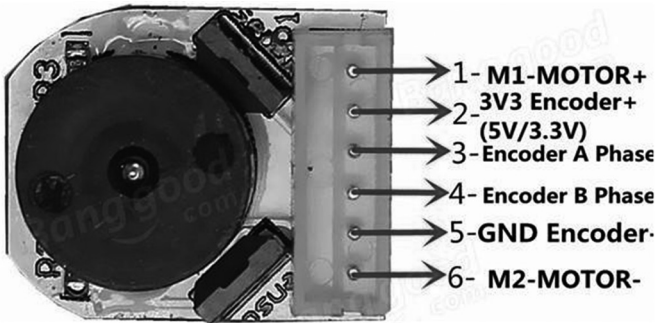

Figure 2.63 Motor pin configuration (Pololu.com)

The circuit diagram of the project is shown in Figure 2.64. Only one of the encoder phases is used in this example. The motor power is supplied externally using a +6V DC power supply. The connection between the motor and the Arduino Uno are as follows:

Motor Pin	Arduino Uno Port
Encoder +	+5V
Encoder A Phase	2
Encoder B Phase	No connection
Encoder ground	GND

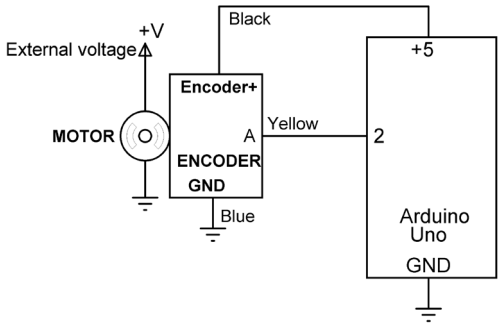

Figure 2.64 Circuit diagram of the project

Encoder pulses are received using one of the external interrupt inputs of the Arduino Uno.

Notice that the encoder output signal is very noisy and it may give rise to higher counts. It is recommended to connect a capacitor from the encoder pin to ground to eliminate this noise.

2.8.3 Program Listing

Arduino Uno has two external interrupt pins at GPIO ports 2 and 3. Pin 2 corresponds to interrupt number 0 and pin 3 corresponds to interrupt number 1. When the **attachInterrupt (interrupt number,....,....)** function is used to attach an interrupt to a function we have to specify the interrupt number as the first argument of the function. Alternatively, we can use function **attachInterrupt(digitalPinToInterrupt(pin number),.....,...)** and specify the GPIO pin number as the external interrupt pin.

Figure 2.65 shows the program listing of the project (program: motor7). PhaseA is assigned to pin number 2 and this pin is attached to interrupt service routine called **EncoderISR**. Variable interval is set to 1 second. Inside the **EncoderISR** the value of variable **Count** is incremented by one.

Function **millis()** returns the number of milliseconds that elapsed since the program started running on the Arduino Uno. Inside the program loop the elapsed milliseconds since the last time is calculated and if this is greater than or equal to the interval (1 second) then it is assumed that a second has elapsed. At the end of one second the speed of the motor shaft after the gears is determined in RPM by multiplying the **Count** by 60 and dividing by the Hall Effect resolution. This is then displayed on the Serial Monitor of the Arduino IDE. Notice that the Arduino **delay** function is not used here to create one second timing interval for the **Count** since it gives inaccurate results when this function is interrupted by an external or a timer interrupt. Instead, the **mills()** function is used to create a one second timing

interval. Figure 2.66 shows the speed displayed on the Serial Monitor.

```
/*****************************************************************
*         MOTOR SPEED SENSE USING ROTARY ENCODER
*         ======================================
*
* In this project the speed of a motor is measured using a Hall
* Effect sensor with a magnetic disc attached to the shaft of the
* motor.
*
* The speed (RPM) of the motor is found by counting the number
* of encoder pulses in a given time period (e.g. in a second).
*
* The rotary encoder ouput is connected to pin 2 of the Arduino
* Uno and the pulses generate external interrupts which are
* counted inside the interrupt service routine EncoderISR.
*
*
* File  : Motor7
* Date  : August 2017
* Author: Dogan Ibrahim
*****************************************************************/
#define PhaseA 2                                  // Phase A
volatile unsigned long Count = 0;                 // Encoder count
unsigned long Pulses;                             // Encoder pulses
unsigned long RPM;                                // Motor speed
unsigned long previousMillis = 0;
const long interval = 1000;                       // milliseconds
//
// This is the interrupt service routine which is called everytime
// a pulse is generated from the encoder output rising (going from
// LOW to HIGH)
//
void EncoderISR()
{
  Count++;
}

//
// Configure Encoder Phase A as input. Also, attach the Encoder A
// output to external interrupt 0 (GPIO pin 2). The interrupt
// service routine is named EncoderISR and is when an interrupt
// occurs (an encoder pulse is generated). External interrupts
// are generated on the rising edge (LOW to HIGH) of the encoder
//
void setup()
```

```
{
  Serial.begin(9600);
  pinMode(PhaseA, INPUT);                          // Phase A is input
  attachInterrupt(digitalPinToInterrupt(2), EncoderISR, RISING);
}

//
// Wait until a second has elapsed and then calculate the speed
// of the motor in RPM and display the speed on the Serial Monitor.
// The interval is set to 1000ms (1 second). When
//           CurrentMillis-previousMillis = interval
// then it is assumed that a second has elapsed
//
void loop()
{
     unsigned long CurrentMillis = millis();
     if(CurrentMillis - previousMillis >= interval)
     {
        Pulses = Count;                               // Copy of Count
        previousMillis = CurrentMillis;
        Count = 0;                                    // Clear Count
        RPM = Pulses * 60 /5320;                      // Calculate RPM
        Serial.println(RPM);                          // Display motor speed
      }
}
```

Figure 2.65 Program listing of the project

Figure 2.66 Motor speed displayed on the Serial Monitor

In this project only one sensor is used. For more accurate results you may like to use both sensors by connecting the PhaseB sensor output to external interrupt input pin 3 and then incrementing Count in both interrupt service routines.

2.9 PROJECT 8 - Displaying Motor Speed on LCD (Arduino Uno)

This project is similar to the previous project, but here the motor speed is displayed on an LCD in RPM. An I2C type LCD is used in this project.

2.9.1 Block Diagram

Figure 2.67 shows the block diagram of the project.

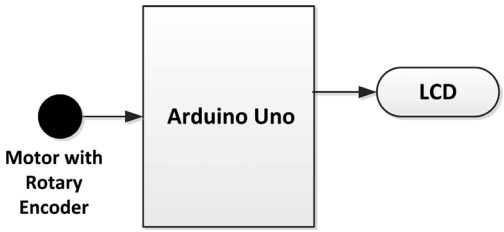

Figure 2.67 Block diagram of the project

2.9.2 Circuit Diagram

The circuit diagram of the project is shown in Figure 2.68. The LCD has 4 pins: GND, +V, SDA, and SCL. GND and +V pins are connected to ground and +5V supply voltages of the Arduino respectively. SDA is connected to pin A4 and SCL is connected to pin A5. PhaseA output of the rotary encoder is connected to pin 2 of the Arduino as in the previous project.

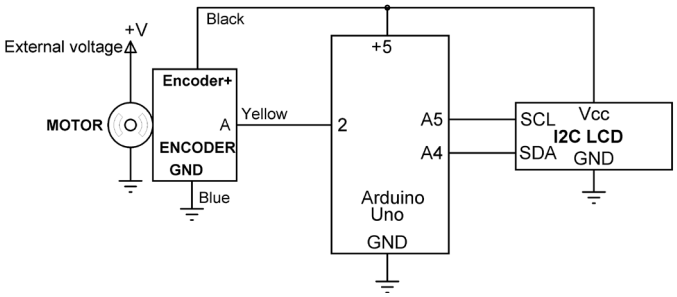

Figure 2.68 Circuit diagram of the project

The LCD used in this project is based on the I2C (or I^2C) interface. I2C is a multi-slave, multi-master, single-ended serial bus used to attach low-speed peripheral devices to microcontrollers. The bus consists of only two wires called SDA and SCL where SDA is the data line and SCL is the clock line and up to 1008 slave devices can be supported on the bus. Both lines must be pulled up to the supply voltage by suitable resistors. The clock signal is always generated by the bus master. The devices on the I2C bus can communicate at 100 kHz or 400 kHz.

Figure 2.69 shows the front and back of the I2C based LCD. Notice that the LCD has a small board mounted at its back to control the I2C interface. The LCD contrast is adjusted through the small potentiometer mounted on this board. A jumper is provided on this board to disable the backlight if required.

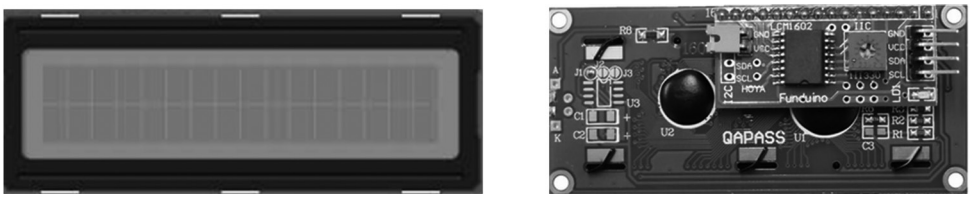

Figure 2.69 I2C based LCD (front and back views)

2.9.3 Program Listing

The I2C LCD library supports the following commands:

clear()	clear the LCD and position cursor to first position
home()	home the cursor to first position
setCursor(x, y)	set cursor to column x, row y (index starts from 0)
print()	display on LCD
display()	show characters on the display (default)
noDisplay()	do not show any characters on the display
blink()	start blinking the cursor
noBlink()	stop blinking the cursor
cursor()	display the cursor indicator
noCursor()	do not show the cursor indicator
scrollDisplayLeft()	scroll the display left by one character position
scrollDisplayRight()	scroll the display right by one character position
leftToRight()	text to flow to the right from the cursor, as if the display is left-justified (default)
rightToLeft()	text to flow to the left from the cursor, as if the display is right-justified.
noBacklight()	disable backlight
backlight()	enable backlight
getBacklight()	get backlight value
autoscroll()	moves all the text one space to the left each time a letter is added (default)
noAutoscroll()	disabled auto scrolling

Before programming the Arduino Uno for the I2C LCD, it is necessary to download and include the I2C LCD library in our Arduino IDE folder (if it not already there). The steps to do this are given below:

- Create a folder named **LiquidCrystal_I2C** under the folder named libraries in your Arduino IDE folder (see Figure 2.70).

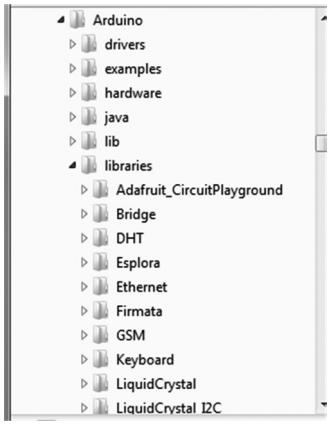

Figure 2.70 Create folder LiquidCrystal_I2C

- Go to the following link to download the I2C LCD library:

  https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library

- Click button **Clone or download** (see Figure 2.71) and copy all the files to the **LiquidCrystal_I2C** folder as shown in Figure 2.72.

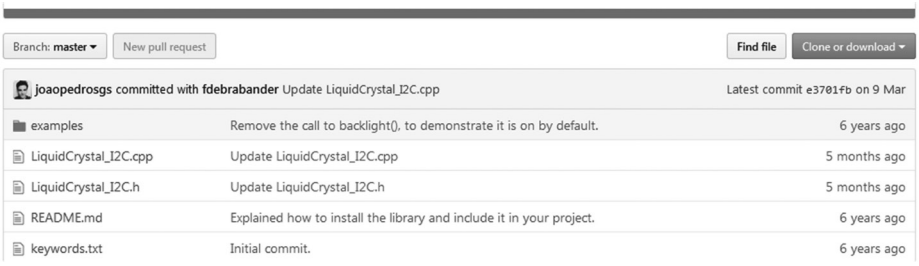

Figure 2.71 Download the I2C library files

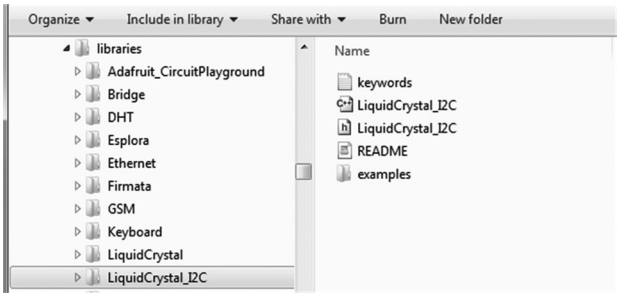

Figure 2.72 Files in the LiquidCrystal_I2C folder

- Start the Arduino IDE. Go to **File -> Examples** and you should see **Liquid-Crystal_I2C** examples in the drop down menu if the library has been installed correctly.

The program listing is shown in Figure 2.73 (program: motor8). At the beginning of the program libraries **Wire** (I2C library) and **LiquidCrystal_I2C** are included in the program. Then the address of the LCD (0x27) and its configuration (16 columns by 2 rows) are defined in the I2C LCD library. Inside the setup routine the program then initializes the I2C LCD library and turns ON the backlight.

Inside the main program loop the encoder pulses are read as in the previous project and displayed on the second row of the LCD. The text **Motor Speed (RPM)** is displayed on the first row of the LCD. External interrupt on pin 2 is then assigned to interrupt service routine EncoderISR as in the previous project. Figure 2.74 shows the output on the LCD.

```
/*****************************************************************
*              MOTOR SPEED DISPLAY ON LCD
*              ==========================
*
* In this project the speed of a motor is measured using a Hall
* Effect sensor with a magnetic disc attached to the shaft of the
* motor as in the previous project. An I2C LCD is connected to
* the Arduino UNO and the motor speed is displayed on the LCD in
* RPM.
*
*
* File  : Motor8
* Date  : September 2017
* Author: Dogan Ibrahim
*****************************************************************/
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

#define PhaseA 2                                  // Phase A
volatile unsigned long Count = 0;                 // Encoder count
unsigned long Pulses;                             // Encoder pulses
unsigned long RPM;                                // Motor speed
unsigned long previousMillis = 0;
const long interval = 1000;                       // milliseconds
//
// Set the LCD address to 0x27 and the configuration to
// 16 chars and 2 rows display
//
LiquidCrystal_I2C lcd(0x27, 16, 2);         // LCD address 0x27

//
// This is the interrupt service routine which is called everytime
// a pulse is generated from the encoder output rising (going from
// LOW to HIGH)
//
```

```
void EncoderISR()
{
  Count++;
}

//
// Configure Encoder Phase A as input. Also, attach the Encoder A
// output to external interrupt 0 (GPIO pin 2). The interrupt
// service routine is named EncoderISR and is when an interrupt
// occurs (an encoder pulse is generated). External interrupts
// are generated on the rising edge (LOW to HIGH) of the encoder
//
void setup()
{
  lcd.begin();                                    // Initialize LCD
  lcd.backlight();                                // Turn ON backlight
  lcd.clear();                                    // Clear LCD
  pinMode(PhaseA, INPUT);                         // Phase A is input
  attachInterrupt(digitalPinToInterrupt(2), EncoderISR, RISING);
}

//
// Wait until a second has elapsed and then calculate the speed
// of the motor in RPM and display the speed on the Serial Monitor.
// The interval is set to 1000ms (1 second). When
//           CurrentMillis-previousMillis = interval
// then it is assumed that a second has elapsed
//
void loop()
{
     unsigned long CurrentMillis = millis();
     if(CurrentMillis - previousMillis >= interval)
     {
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Motor Speed(RPM)");
        Pulses = Count;                                // Copy of Count
        previousMillis = CurrentMillis;
        Count = 0;                                     // Clear Count
        RPM = Pulses * 60 /5320;                       // Calculate RPM
        lcd.setCursor(0, 1);
        lcd.print(RPM);
      }
}
```

Figure 2.73 Program listing of the project

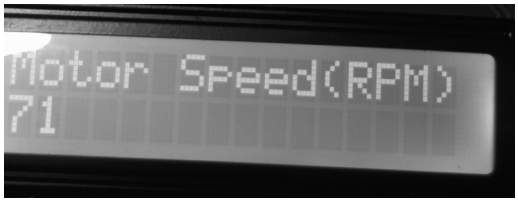

Figure 2.74 Output on the LCD

2.10 PROJECT 9 - Identification of the DC Motor (Arduino Uno)

In this project we will plot the step time response of our DC motor with the encoder and then find the time response parameters from this plot.

2.10.1 Block Diagram

The block diagram of this project is same as Figure 2.59.

2.10.2 Circuit Diagram

The circuit diagram of this project is same as Figure 2.64

2.10.3 Program Listing

In this project the encoder sampling time is reduced to 5ms so that we can plot the step input time response of the motor. In addition, the speed unit is changed from RPM to radians/sec by multiplying the speed by 2π and dividing by 60.

Figure 2.75 shows the program listing (program: motor9). Notice here that the encoder time interval is reduced to 5ms.

```
/*****************************************************************
*               MOTOR STEP RESPONSE
*               ===================
*
* In this project the speed of a motor is measured using a Hall
* Effect sensor with a magnetic disc attached to the shaft of the
* motor.
*
* The time response of the motor is plotted by applying a 6V
* step voltage and then plotting the speed in rad/sec using the
* Arduino Serial Plotter. The time interval of the encoder readinmg
* is set to 5ms.
*
*
* File  : Motor9
* Date  : August 2017
* Author: Dogan Ibrahim
*****************************************************************/
#define PhaseA 2                                  // Phase A
volatile unsigned long Count = 0;                 // Encoder count
```

```
unsigned long Pulses;                              // Encoder pulses
unsigned long RADS;                                // Motor speed
unsigned long previousMillis = 0;
const long interval = 5;                           // milliseconds
//
// This is the interrupt service routine which is called everytime
// a pulse is generated from the encoder output rising (going from
// LOW to HIGH)
//
void EncoderISR()
{
  Count++;
}

//
// Configure Encoder Phase A as input. Also, attach the Encoder A
// output to external interrupt 0 (GPIO pin 2). The interrupt
// service routine is named EncoderISR and is when an interrupt
// occurs (an encoder pulse is generated). External interrupts
// are generated on the rising edge (LOW to HIGH) of the encoder
//
void setup()
{
  Serial.begin(9600);
  pinMode(PhaseA, INPUT);                          // Phase A is input
  attachInterrupt(digitalPinToInterrupt(2), EncoderISR, RISING);
}

//
// Wait until a second has elapsed and then calculate the speed
// of the motor in RPM and display the speed on the Serial Monitor.
// The interval is set to 5ms
//
void loop()
{
     unsigned long CurrentMillis = millis();
     if(CurrentMillis - previousMillis >= interval)
     {
        Pulses = Count;                                  // Copy of Count
        previousMillis = CurrentMillis;
        Count = 0;                                       // Clear Count
        RADS = Pulses * 200*2*3.1415 /5320;              // Speed in rad/s
        Serial.println(RADS);
      }
}
```

Figure 2.75 Program listing of the project

The time response of the motor is plotted using the **Serial Plotter** menu option of the Arduino IDE. Here, the Y axis is the motor speed in radians/sec and the horizontal axis is the time in units of 5ms (X axis ticks depend upon the interval of the **Serial.println** statements used in the program). Figure 2.76 shows the time response of the motor.

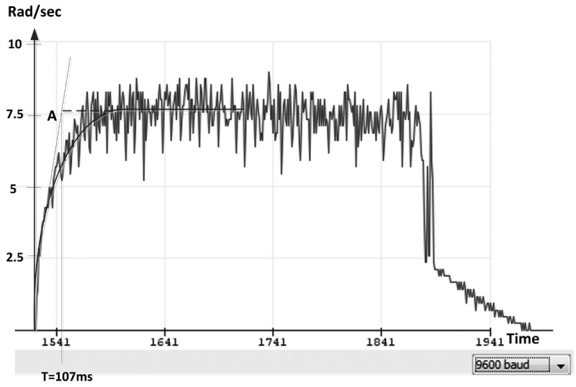

Figure 2.76 Time response of the motor

The time response of a first order system is found by drawing a line through the initial slope and then intersecting this line with the final value of the time response. The time at the intersection point (point **A** in Figure 2.76) is the time constant of the system. In this example, the time constant is found to be T = 107ms.

The time response of the motor is shown in Figure 1.12 in Chapter 1 and is repeated below:

$$W(t) = \frac{V_O}{K_E}\left(1 - e^{-t/T}\right) \tag{1.12}$$

Where T is the time constant and V_O is the applied step voltage . When the speed settles down, i..e at large values of time, equation (1.12) reduces to:

$$W(t) = \frac{V_O}{K_E}$$

The test was carried out with a step input voltage of 6V. From Figure 2.76 the final value of the speed is 7.5 rad/sec, therefore,

$$K_E = \frac{V_O}{W(t)} = \frac{6}{7.5} = 0.8$$

The time response of the motor system can therefore be written as:

$$W(t) = \frac{V_O}{0.8}\left(1 - e^{-t/0.107}\right)$$

or, $W(t) = 1.25V_O\left(1 - e^{-t/0.107}\right.$

which corresponds to the motor transfer function in the Laplace domain of:

$$\frac{W(s)}{V_O(s)} = \frac{0.208}{0.107\,s + 1}$$

2.11 PROJECT 10 – Closed Loop PID Speed Control of a DC Motor

This is a closed loop speed control of a motor. Feedback is applied through the encoder and the speed of the motor is controlled using PID type controller.

2.11.1 Block Diagram

The block diagram of the project is shown in Figure 2.77. The same motor and encoder as in the previous project are used here. Here, Arduino Uno is the digital controller. A MOSFET transistor is used to drive the motor. The motor speed is measured using an encoder as in the previous project and the speed is fed back to the Arduino. The Arduino implements the PID algorithm to keep the motor speed at the desired set value. The desired value is set (hardcoded) in software. An LCD is used to show the desired and the actual speeds. Top row of the LCD shows the desired speed while the bottom row shows the actual speed in radians/second.

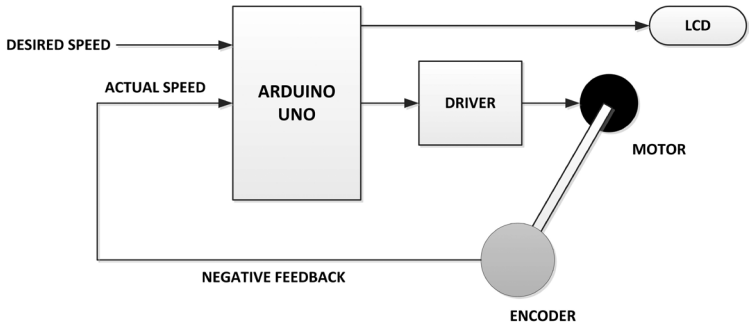

Figure 2.77 Block diagram of the project

2.11.2 Circuit Diagram

Figure 2.78 shows the circuit diagram of the project. The Gate terminal of the MOSFET transistor is connected to pin 3 of the Arduino where this pin is configured to generate PWM waveform so that the average voltage applied to the motor can be varied as the duty cycle is changed. PhaseA encoder output is connected to port pin 2 of the Arduino Uno as before. An I2C LCD is connected to pins A4 and A5 of the Arduino as in the previous project. The LCD displays the desired and the actual speeds, which should be the same if the controller is working correctly.

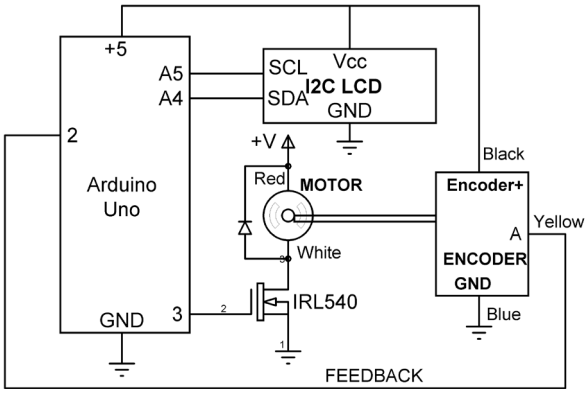

Figure 2.78 Circuit diagram of the project

2.11.3 Program Listing

PID (Proportional+Integral+Derivative) is a type of controller used in most automatic control systems. The controller receives the desired and the actual values of the variable to be controlled, calculates the error term which is the difference between the two, and then implements a control algorithm to achieve the required control action.

PID controllers used to be analog where operational amplifiers and similar electronic devices were used to implement the PID functions. The control action in the analog form of the PID algorithm is as follows:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}$$

Where e(t) is the error term, K_p is the proportional constant, K_i is the integral constant, and K_d is the derivative constant of the controller. There are many tuning algorithms that can be used to estimate the optimum values of these constants. Nowadays, the PID algorithm is implemented in software on microcontrollers where the digital form of this controller is used. The digital PID controller has the following digital form, represented in terms of Z-transforms (The details of the PID algorithm are beyond the scope of this book. Interested readers can find lots of information on PID controllers on the Internet and in many automatic control books):

$$U(z) = K_p E(z) + \frac{K_i}{1 - z^{-1}} E(z) + K_d \left(1 - z^{-1}\right) E(z)$$

Figure 2.79 show the program listing (program: motor10). At the beginning of the program the I2C LCD libraries are included, PhaseA is assigned to pin 2 and PWM to pin 3 respectively. The PID parameters are then defined and the sampling time is set to 100ms, and the desired speed is set to 3 radians/second. In this project only PI (proportional+integral) controller is used. In this type controller the steady state error is always zero. After several trials and errors the constants K_p and K_i were set to 5.0 and 3.0 respectively which gave satisfactory time response. Inside the setup routine PhaseA is configured as input and

PWM is configured as output. External interrupt from pin 2 is attached to interrupt service routine EncoderISR.

Inside the program loop the speed of the motor is read as radians/second and stored in variable **Speed**. The error between the desired speed and the actual speed is calculate and stored in variable Error. Then the output of the PID algorithm is calculated and is stored in variable pid. In PID controller applications wind-up errors can occur because the integral term increases constantly if there is an error. Since this output can be very large as the integrator term accumulates the error, it is necessary to limit it. The remainder of the program limits the PID output and sends PWM voltage to the motor to control its speed.
The PID algorithm is implemented by the following code:

```
Error = DesiredSpeed – Speed;            // Error term
Integrator = Integrator + Error;         // Integrator term
pid = kp*Error + ki*Integrator;          // PID output
PID = (int)pid;                          // Integer form
if(PID > 0)                              // If positive
{
    if(PID > 255) analogWrite(PWM, 255); // Send 100% duty cycle
  else analogWrite(PWM, PID);            // Send normal duty cycle
}
if(PID < 0)                              // If negative
{
   if(PID < -255)analogWrite(PWM,0);     // Send 0% duty cycle
   else
  {
     PID = abs(PID);                     // Make it positive
     analogWrite(PWM,255-PID);           // Send normal duty cycle
  }
}
```

Figure 2.80 shows the speed response of the motor, plotted using the Serial Plotter of the Arduino IDE. As it can be seen from this figure, the speed settles down to the desired value of 3 radians/second.

The **Serial.begin** and **Serial.println** statement are used so that the time response can be plotted. These statements can be removed in a working system to increase the throughput.

```
/*****************************************************************
*             CLOSED LOOP MOTOR SPEED CONTROL
*             ===============================
*
* This is a closed loop motor control project. The encoder reading
* is used as feedback and a PID algorithm is used to control the
* speed of the motor.
*
```

```
* PhaseA of the encoder output is connected to pin 2 of the Arduino.
* Pin 3 of the Arduino is connected to a MOSFET transistor which
* drives the motor by sending PWM pulses to it. An I2C LCD is
* connected to pins A4 and A5 of the Arduino Uno. The LCD shows the
* desired and the actual speeds. PID algorithm is used to control
* teh speed of the motor. The desired speed is hardcoded in the
* program.
*
*
* File  : Motor10
* Date  : September 2017
* Author: Dogan Ibrahim
*****************************************************************/
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

#define PhaseA 2                                    // Phase A
#define PWM 3                                       // MOSFET gate
volatile unsigned long Count = 0;                   // Encoder count
unsigned long Pulses;                               // Encoder pulses
float Speed;                                        // Motor speed
float DesiredSpeed = 3;                             // Desired speed
float Integrator = 0.0;
unsigned long previousMillis = 0;
const long interval = 100;                          // 100 milliseconds
float ki = 3.0;                                     // Integrator constant
float kp = 5.0;                                     // Proportional constant
float pid, Error;
int PID;
//
// Set the LCD address to 0x27 and the configuration to
// 16 chars and 2 rows display
//
LiquidCrystal_I2C lcd(0x27, 16, 2);                 // LCD address 0x27

//
// This is the interrupt service routine which is called everytime
// a pulse is generated from the encoder output rising (going from
// LOW to HIGH)
//
void EncoderISR()
{
  Count++;
}

//
```

```
// Configure Encoder Phase A as input. Also, attach the Encoder A
// output to external interrupt 0 (GPIO pin 2). The interrupt
// service routine is named EncoderISR and is when an interrupt
// occurs (an encoder pulse is generated). External interrupts
// are generated on the rising edge (LOW to HIGH) of the encoder
//
void setup()
{
  Serial.begin(9600);                       // Remove for better performance
  lcd.begin();                                          // Initialize LCD
  lcd.backlight();                                      // Turn ON backlight
  lcd.clear();                                          // Clear LCD
  TCCR2B = TCCR2B & B11111000 | B00000011;              // Set PWM frequency
  pinMode(PhaseA, INPUT);                               // Phase A is input
  pinMode(PWM, OUTPUT);                                 // PWM output
  attachInterrupt(digitalPinToInterrupt(2), EncoderISR, RISING);
}


//
// Wait until 100ms has elapsed and then calculate the speed
// of the motor. The sampling tiem is set to 100ms. This loop
// implements the PID algorithm
//
void loop()
{
     unsigned long CurrentMillis = millis();
     if(CurrentMillis - previousMillis >= interval)
     {
        lcd.clear();                                    // Clear lcd
        Pulses = Count;                                 // Copy of Count
        previousMillis = CurrentMillis;
        Count = 0;                                      // Clear Count
        Speed = Pulses *2*10*3.1415 /5320;              // Actual speed
//
// Implement the PID algorithm
//
        Error = DesiredSpeed - Speed;                   // Error term
        Integrator = Integrator + Error;
        pid = kp*Error + ki*Integrator;                 // PID output
        PID = (int)pid;
        if(PID > 0)
        {
          if(PID > 255) analogWrite(PWM, 255);
          else analogWrite(PWM, PID);
        }
```

```
        if(PID < 0)
        {
          if(PID < -255)analogWrite(PWM,0);
          else
          {
            PID = abs(PID);
            analogWrite(PWM,255-PID);
          }
        }
        lcd.setCursor(0, 0);                              // Set cursor at 0,0
        lcd.print(DesiredSpeed);                          // Display desired
speed
        lcd.setCursor(0,1);                               // Set cursor at 0,1
        lcd.print(Speed);                                 // Actual speed
        Serial.println(Speed);              // Remove for better performance
      }
}
```

Figure 2.79 Program listing of the project

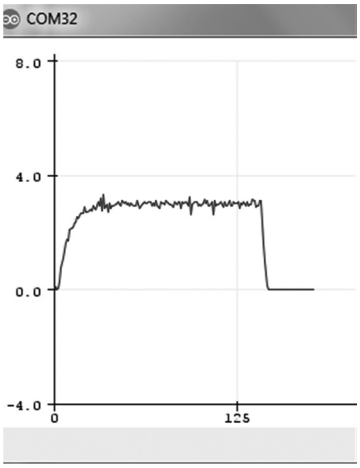

Figure 2.80 Speed response of the motor

The project can be modified and can be made more user friendly if the desired speed is read from an external potentiometer instead of hardcoding it in the program.

2.12 PROJECT 11 – Using a Rotary Encoder - Displaying the Speed of a Motor (Raspberry Pi Zero W)

This project shows how a rotary encoder can be used to sense the speed of a motor. The speed is then displayed on the monitor.

2.12.1 Block Diagram

The block diagram of the project is shown in Figure 2.81. In this project the same small

geared brushed DC motor as the one used in the last few Arduino Uno projects is used.

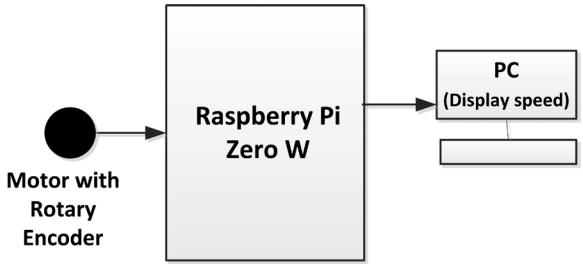

Figure 2.81 Block diagram of the project

2.12.2 Circuit Diagram

Each GPIO pin, when configured as a general-purpose input, can be configured as an interrupt source to the RPi ZW. Several interrupt generation sources are configurable:

- Level-sensitive (high or low)
- Rising or falling edge

Level interrupts maintain the interrupt status until the level has been cleared by system software. Rising edge detection is when the signal at the input pin goes from 0 to 1. Similarly, falling edge detection is when the input signal goes from 1 to 0.

The circuit diagram of the project is shown in Figure 2.82. Only one of the encoder phases is used in this example. The motor power is supplied externally using a +6V DC power supply. The connection between the motor and the RPi ZW are as follows (since any input pin can be configured for external interrupt, pin 2 is chosen in this example project):

Motor Pin	**Raspberry Pi Zero W Port**
Encoder +	+5V
Encoder A Phase	2
Encoder B Phase	No connection
Encoder ground	GND

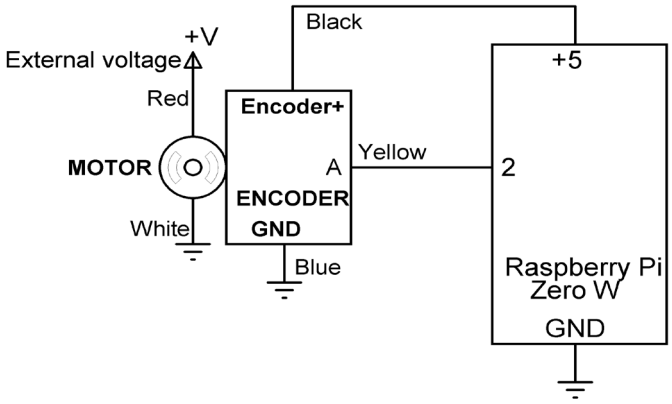

Figure 2.82 Circuit diagram of the project

Encoder pulses are received using one of the external interrupt inputs of the RPi ZW.

2.12.3 Program Listing

Figure 2.83 shows the program listing of the project (program: motor7.py). At the beginning of the program the RPi.GPIO, time, and threading libraries are imported to the program. PhaseA is assigned to pin number 2 and this pin is attached to interrupt handler routine called **EncoderISR**. Inside the **EncoderISR** the value of variable **Count** is incremented by one.

The main program is executed in a function called **Encoder**. Here, the total number of pulses counted in one second is used to calculate the motor speed in RPM. The result is then displayed on the PC monitor. In this program the threading function is used to call function **Encoder** every second. This function has the following format:

```
threading.Timer(next_call - time.time(), Encoder).start()
```

Function **time.time()** returns the time as a floating point number expressed in seconds since the epoch, in UTC. Variable **next_call** is incremented by one at every iteration. In the above statement, function **Encoder** is executed exactly every second at the same point in the second.

```
#-----------------------------------------------------------------
#                MOTOR SPEED SENSE USING ROTARY ENCODER
#                ======================================
#
# In this project teh speed of a motor is measured using a Hall
# Effect sensor with a magnetic disc attached to the shaft of the
# motor.
#
# The speed (RPM) of the motor is found by counting the number of
# encoder pulses in a given time period (e.g. in a second).
```

```
#
# The rotary encoder output is connected to pin 2 of the RPi ZW
# and the pulses generate external interrupts which are counted
# inside the interrupt service routine EncoderISR
#
#
# File  : motor7.py
# Date  : August 2017
# Author: Dogan Ibrahim
#-------------------------------------------------------------------------
import RPi.GPIO as GPIO
import time
import threading

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
PhaseA = 2
global Count
Count = 0
GPIO.setup(PhaseA, GPIO.IN)
#
# This is the interrupt service routine. The program jumps
# here when a pulse is received from the encoder
#
def EncoderISR(dummy):
  global Count
  Count = Count + 1

#
# Attach function EncoderISR to pin 2 so that when pin 2
# goes 0 to 1 (rising) the program generates an interrupt
# and goes to function EncoderISR
#
GPIO.add_event_detect(PhaseA, GPIO.RISING, callback=EncoderISR)

next_call = time.time()
#
# Speed calculation routine. Here the encoder pulses are counted
# in one second and then the motor speed is calculated in RPM
# and displayed on the PC monitor using the print statement
#
def Encoder():
  global Count, next_call
  Pulses = Count
  RPM = Pulses * 60 / 5320
  print(RPM)
```

```
    next_call = next_call + 1
    threading.Timer(next_call-time.time(),Encoder).start()
    Count = 0

#
# Start the speed calculation routine
#
Encoder()
while True:
    pass
```

Figure 2.83 Program listing of the project

2.13 PROJECT 12 - Displaying Motor Speed on LCD (Raspberry Pi Zero W)

This project is similar to the previous project, but here the motor speed is displayed on an LCD in RPM. An I2C type LCD is used in this project.

2.13.1 Block Diagram

Figure 2.84 shows the block diagram of the project.

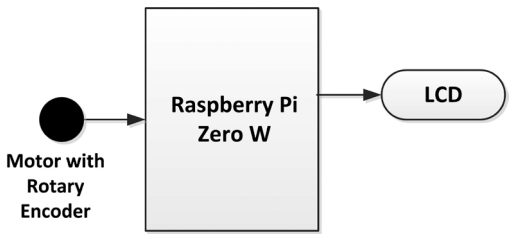

Figure 2.84 Block diagram of the project

2.13.2 Circuit Diagram

The circuit diagram of the project is shown in Figure 2.85. Pin 3 (GPIO2) and 5 (GPIO3) of the RPi ZW are the I2C data and control pins SDA and SCL respectively. The LCD has 4 pins: GND, +V, SDA, and SCL. SDA and SCL of the LCD are connected to pins 3 and 5 of the RPi ZW respectively. PhaseA output of the rotary encoder is connected to pin 7 (GPIO4) of the RPi ZW.

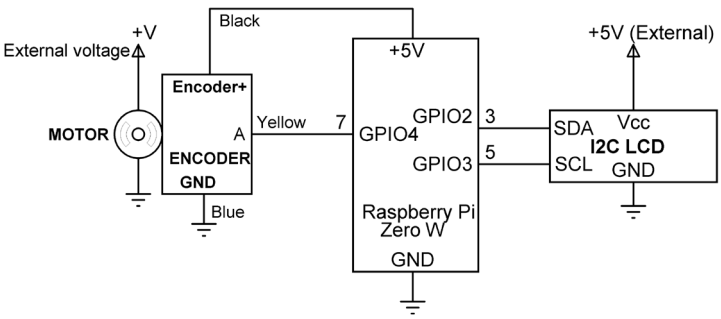

Figure 2.85 Circuit diagram of the project

Notice that the LCD is powered from an external +5V power supply since the +5V supply of the RPi ZW does not have enough current to drive the LCD. Also note that there is no problem mixing the +3.3V GPIO pins of the RPi ZW with the +5V of the I2C LCD. This is because the RPi ZW is the I2C master device and the SDA and SCL lines are pulled up to +3.3V through resistors. SCL line is the clock which is always output from the master device. The slave device (I2C LCD here) only pulls down the SDA line when it acknowledges the receipt of data and it does not send any data to the master device. Therefore, there are no voltage level problems as long as the RPi ZW I2C output pins can drive the I2C LCD inputs, which is the case here.

2.13.3 Program Listing

Before using the I2C pins of the RPi ZW we have to enable the I2C peripheral interface on the device. The steps for this are as follows:

- Start the configuration menu from the command prompt:

  pi@raspberrypi:~ $ **sudo raspi-config**

- Go down the menu to **Interface Options**
- Go down and select **I2C**
- Enable the I2C interface
- Select **Finish** to complete

Now we have to install the I2C library on our Rpi ZW. The steps are as follows:

- Enter the following commands from the command menu:

  pi@raspberrypi:~ $ **sudo apt-get update**
  pi@raspberrypi:~ $ **sudo apt-get install –y python-smbus i2c-tools**
  pi@raspberrypi:~ $ **sudo reboot**

- Enter the following command to test the installation. You should see **i2c_ bcm2835** listed:

  pi@raspberrypi:~ $ **lsmod | grep i2c_**

- Modify the config file as follows:

  pi@raspberrypi:~ $ **sudo nano /etc/modules**

  Add the following lines (if they are not already there):

i2c-bcm2835
i2c-dev

Exit from nano by typing Ctrl X and Y to save the file. You can check the Contents of this file by entering the command:

pi@raspberrypi:~ $ **cat /etc/modules**

- Reboot the RPi ZW by entering:

pi@raspberrypi:~ $ **sudo reboot**

- Connect your LCD to the RPi ZW device and enter the following command to check whether or not the LCD is recognizedby the RPi ZW:

pi@raspberrypi:~ $ **sudo i2cdetect –y 1**

You should see a table similar to the one shown below. A number in the table means that the LCD has been recognizes correctly and the I2C slave address of the LCD is shown in the table. In this example the LCD address is 27:

```
     1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- 27 -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

We should now install an I2C LCD library so that we can send commands and data to our LCD. There are many Python libraries available for the I2C type LCDs. The one chosen here is called the **RPi_I2C_driver**. This library is installed as follows:

- Go to the following web link:

https://gist.github.com/DenisFromHR/cc863375a6e19dce359d

- Scroll down to section **RPi_I2C_driver.py**. Click **Raw** at the top right hand side of the screen and save the file in a folder (e.g. Desktop) with the name **RPi_I2C_driver.py** (the easiest option might be to copy the file into the Notebook and then save it as RPi_I2C_driver.py).

- Start your Raspberry Pi Zero W in command mode.

- Start the WinSCP file copy utility (you should install it if you already do not have it) on your PC and copy file **RPi_I2C_driver.py** to folder **usr/lib/python2.7** on your Raspberry Pi Zero W.

- Check to make sure that the file is copied successfully. You should see the file listed with the command:

```
pi@raspberrypi: ~ $ ls /usr/lib/python2.7
```

We are now ready to write our program. Figure 2.86 shows the program listing (program: motor8.py). At the beginning of the program libraries RPi.GPIO, timer, threading, and the LCD driver library RPi_I2C_driver have been imported to the program. The heading MOTOR SPEED is then displayed on the LCD. Encoder pulses are attached to interrupt service routine **EncoderISR** where the pulses are counted. Inside the **Encoder** routine the motor speed is calculated and then displayed on the LCD in RPM.

```
#-----------------------------------------------------------------
#                DISPLAY MOTOR SPEED ON LCD
#                ==========================
#
# In this project the speed of a motor is measured using a Hall
# Effect sensor with a magnetic disc attached to the shaft of the
# motor.
#
# The speed (RPM) of the motor is found by counting the number of
# encoder pulses in a given time period (e.g. in a second).
#
# The motor speed is displayed on an I2C type LCD in RPM
#
# File  : motor8.py
# Date  : August 2017
# Author: Dogan Ibrahim
#-------------------------------------------------------------------------
import RPi.GPIO as GPIO
import time
import threading
import RPi_I2C_driver
LCD = RPi_I2C_driver.lcd()
LCD.lcd_display_string("MOTOR SPEED",1)
time.sleep(2)

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
PhaseA = 4
global Count
Count = 0
```

```
GPIO.setup(PhaseA, GPIO.IN)
#
# This is the interrupt service routine. The program jumps
# here when a pulse is received from the encoder
#
def EncoderISR(dummy):
  global Count
  Count = Count + 1

#
# Attach function EncoderISR to pin 2 so that when pin 2
# goes 0 to 1 (rising) the program generates an interrupt
# and goes to function EncoderISR
#
GPIO.add_event_detect(PhaseA, GPIO.RISING, callback=EncoderISR)

next_call = time.time()
#
# Speed calculation routine. Here the encoder pulses are counted
# in one second and then the motor speed is calculated in RPM
# and displayed on the PC monitor using the print statement
#
def Encoder():
  global Count, next_call
  Pulses = Count
  RPM = Pulses * 60 / 5320
  LCD.lcd_clear()
  rpm = str(RPM)
  LCD.lcd_display_string(rpm,1)
  next_call = next_call + 1
  threading.Timer(next_call-time.time(),Encoder).start()
  Count = 0

#
# Start the speed calculation routine
#
Encoder()
while True:
  pass
```

Figure 2.86 Program listing

The I2C LCD library supports the following functions (see the I2C LCD library documentation for more details):

lcd_clear()	clear LCD and set to home position
lcd_display_string(text, row)	display text at LCD row

lcd_write_char(c)	display character
lcd_write(cmd)	write command cmd to LCD
lcd.backlight(1/0)	enable/disable LCD backlight
lcd_display_string_pos(text,row,col)	display text at given row,column

2.14 PROJECT 13 – Using Timer Interrupts to Calculate the Motor Speed (Arduino Uno)

This project is similar to Project 7 where the speed of the motor is calculated and displayed on the Serial Monitor. In this project Timer1 of the Arduino Uno is used to generate interrupts every second. Inside the interrupt service routine the motor speed is calculated and displayed. Using timer interrupts give very accurate results since the encoder pulses are read exactly at every second.

2.14.1 Block Diagram

The block diagram of the project is as in Figure 2.59. Same DC motor as in Project 7 is used here.

2.14.2 Circuit Diagram

The circuit diagram of the project is as in Figure 2.64. Only one of the encoder phases is used. **Note that the encoder output signal is very noisy and it may give rise to higher counts. It is recommended to connect a capacitor from the encoder pin to ground to eliminate this noise.**

2.8.3 Program Listing

Arduino Uno has 3 timers called timer0, timer1 and timer2. Each of these timers have a counter that is incremented by one on each clock pulse. Once a timer counter reaches the value stored in a compare match register it will clear to zero on the next clock pulse and at the same time generate an interrupt (if interrupts are enabled). Arduino Uno clock frequency is 16 MHz which has a period about 63ns. A pre-scaler register is used to divide the clock frequency by a factor of 8, 64, 256 or 1024 so that longer interrupt times can be obtained. Timer0 and timer 2 are 8 bit timers, meaning they can store a maximum counter value of 255. Timer1 is a 16 bit timer which can store a maximum counter value of 65535. By selecting a pre-scaler value we can set the time to interrupt as desired.

The number to be loaded into the compare match register can be calculated using the following formula (compare match register is zero indexed):

Compare match register value = [16,000,000 Hz / (pre-scaler * desired interrupt frequency)] – 1

For example, to generate interrupts every second (1 Hz), assuming a pre-scaler of 1024 is used, the value to be loaded into the compare match register is calculated as follows:

Compare match register value = [16,000,000 / (1024 * 1)] – 1 =15624

The required pre-scaler value is selected by the three least significant bits CS12, CS11, and

CS10 of the TCCRxB register (x is 0 for timer0, 1 for timer1, and 2 for timer2) as shown in Table 2.3

CS12	CS11	CS10	Description
0	0	0	Timer stopped
0	0	1	No pre-scaling
0	1	0	1/8 pre-scaler
0	1	1	1/64 pre-scaler
1	0	0	1/256 pre-scaler
1	0	1	1/1024 pre-scaler
1	1	0	External clock (falling edge)
1	1	1	External clock (rising edge)

Table 2.3 Pre-scaler selection

Timer registers must be configured correctly in order to generate interrupts at the required times. The registers to be configured in Timer1 are: TCCR1A, TCCR1B, OCR1A, and TIMSK1 (timer0 registers are: TCCR0A, TCCR0B, OCR0A and TIMSK0. Similarly, timer2 registers are: TCCR2A, TCCR2B, OCR2A, and TIMSK2). As an example, the code for configuring timer1 to generate interrupts every second is given below:

```
TCCR1A = 0;                                    // Clear TCCR1A
TCCR1B = 0;                                    // Clear TCCR1B
TCNT1 = 0;                                     // Clear counter
OCR1A = 15624;                                 // Set for 1 second
TCCR1B |= (1 << WGM12);
TCCR1B |= (1 << CS12) | (1 << CS10);           // Set pre-scaler to 1024
TIMSK1 |= (1 << OCIE1A);                       // Enable Timer1 interrupts
```

Timer1 interrupt service routine is called: ISR(TIMER1_COMPA_vect) and the program jumps to this routine whenever a timer1 interrupt occurs.

Figure 2.87 shows the program listing (program: motor11). At the beginning of the program PhaseA is assigned to port 2. Inside the setup routine the Serial Monitor is initialized, and timer1 is configured to generate interrupts every second so that the program jumps to function **ISR(TIMER1_COMPA_vect)** every second. Also, **attachInterrupt** is called to attach the encoder interrupts to function **EncoderISR** so that the program jumps to this function to increment the encoder pulse count whenever the encoder signal goes from 0 to 1 (rising). Inside the timer1 interrupt service routine the program calculates and then displays the motor speed on the Serial Monitor of the Arduino IDE. Notice that the main program inside the loop does not do any processing.

```
/*****************************************************************
*         MOTOR SPEED DETECTION USING TIMER INTERRUPTS
*         ============================================
*
* In this project the speed of a motor is measured using a Hall
* Effect sensor with a magnetic disc attached to the shaft of the
* motor.
*
* The speed (RPM) of the motor is found by counting the number
* of encoder pulses in a given time period (e.g. in a second).
*
* In this program Timer1 is configured to interrupt every second
* and the speed is calculated inside the timer interrupt service
* routine
*
*
* File  : Motor11
* Date  : August 2017
* Author: Dogan Ibrahim
*****************************************************************/
#define PhaseA 2                                  // Phase A
volatile unsigned long Count = 0;                 // Encoder count
unsigned long Pulses;                             // Encoder pulses
unsigned long RPM;                                // Motor speed
//
// This is the interrupt service routine which is called everytime
// a pulse is generated from the encoder output rising (going from
// LOW to HIGH)
//
void EncoderISR()
{
  Count++;
}

//
// Configure Encoder Phase A as input. Also, attach the Encoder A
// output to external interrupt 0 (GPIO pin 2). The interrupt
// service routine is named EncoderISR and is when an interrupt
// occurs (an encoder pulse is generated). External interrupts
// are generated on the rising edge (LOW to HIGH) of the encoder
// This routine also configures the Timer1 interrupts so that they
// occur every second
//
void setup()
{
  Serial.begin(9600);
```

```
  TCCR1A = 0;                                    // Clear TCCR1A
  TCCR1B = 0;                                    // Clear TCCR1B
  TCNT1 = 0;                                     // Clear counter
  OCR1A = 15624;                                 // Set for 1 second
  TCCR1B |= (1 << WGM12);
  TCCR1B |= (1 << CS12) | (1 << CS10);           // Set prescaler to 1024
  TIMSK1 |= (1 << OCIE1A);                       // Enable Timer1 interrupts
  pinMode(PhaseA, INPUT);                        // Phase A is input
  attachInterrupt(digitalPinToInterrupt(2), EncoderISR, RISING);
}

//
// Timer1 interrupt service routine. Inside this routine calculate
// and display the motor speed on the Serial Monitor. This routine
// is called every second
//
ISR(TIMER1_COMPA_vect)
{
  Pulses = Count;                                  // Get pulses
  Count = 0;                                       // Clear Count
  RPM = Pulses * 60 / 5320;                        // Calculate RPM
  Serial.println(RPM);                             // Display RPM
}

//
// Main program loop - do nothing and wait for interrupts
//
void loop()
{

}
```

Figure 2.87 Program listing

2.15 PROJECT 14 – Mobile Robot Basic Control (Arduino Uno)

In this project a small mobile robot is used. The aim of this project is to show how the basic robot movement functions such as moving forward, moving backwards, turning left and turning right can be done using an Arduino Uno. In this project the motor performs the following movements:

- Go forward for 3 seconds
- Wait 2 seconds
- Go in reverse direction for 3 seconds
- Wait for 2 seconds
- Turn right for 3 seconds
- Wait for 2 seconds
- Turn left for 3 seconds

- Wait for 3 seconds
- Stop the motor

2.15.1 Block Diagram

The robot used in this project is a very low-cost (around $10) mobile robot having two motors driving two large wheels, and a caster wheel placed at the front of the robot. The robot is sold as a kit with all the parts shown in Figure 2.88.

Figure 2.88 The robot kit

The robot can be built in about half an hour using the instructions supplied. Figure 2.89 shows the robot in built form.

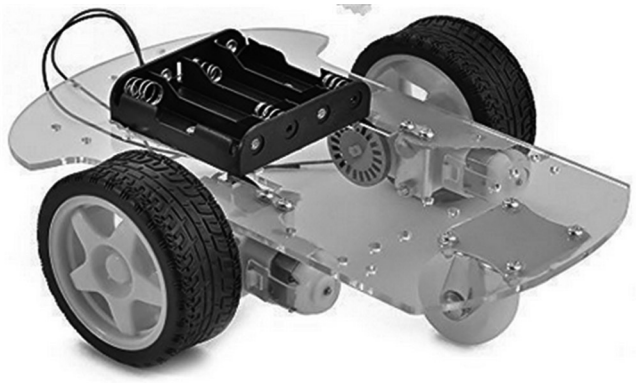

Figure 2.89 Robot in built form

In this project we want to control the robot using an Arduino Uno and because of this the Arduino Uno is mounted on the robot chassis using two small screws. Additionally a small breadboard is fixed using Velcro and mounted on the robot chassis. 4 x NiZn AA batteries are used as the power supply. The reason for using NiZn batteries is because each battery provides 1.6V and therefore the total voltage is 4 x 1.6 = 6.4V. Arduino 5V regulator re-

quires at least 6V for its operation. Using standard alkaline AA batteries may not supply the required voltage (we could also use a PP3 9V battery).

The robot motors are controlled using an L293 IC as described earlier in this book (see Section 2.6). This IC was mounted on the breadboard. Figure 2.90 and Figure 2.91 show the robot assembly with the Arduino, breadboard, batteries, and the jumper wires.

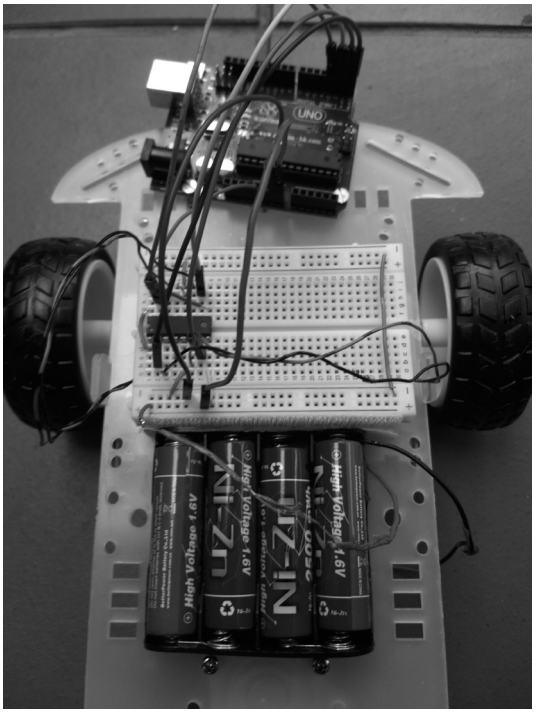

Figure 2.90 The robot assembly (front)

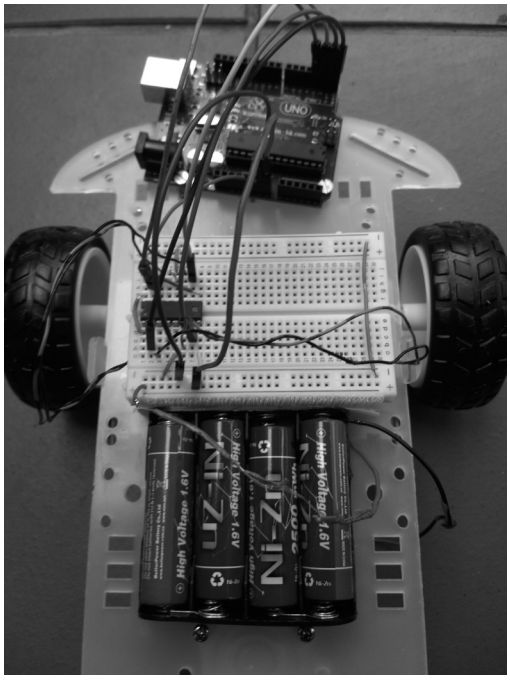

Figure 2.91 The robot assembly (back)

The block diagram of the project is shown in Figure 2.92.

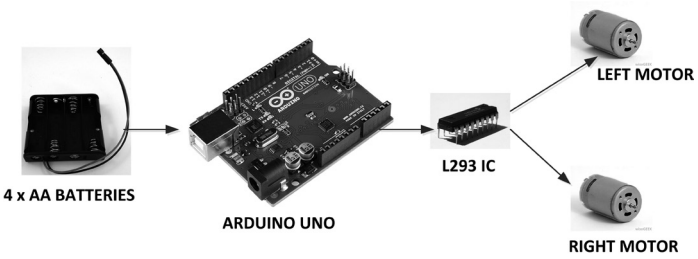

Figure 2.92 Block diagram of the project

2.15.2 Circuit Diagram

The circuit diagram of the project is shown in Figure 2.93. The Arduino Uno is powered from an external battery pack. The + and - pins of the battery pack are connected to Vin and Gnd power supply inputs of the Arduino Uno. Notice that you must never connect the battery pack to the +5V pin of the Arduino Uno since this voltage is high and can damage your Arduino.

The power supply of the logic side (Vcc1) of the L293 H bridge chip is connected to the +5V pin of the Arduino Uno. The motor supply voltage of L293 (pin Vcc2) is connected directly to the battery + pin since Arduino cannot supply enough current to drive the motors.

L293 includes two independent motor driver circuits: 1A, 2A ,1Y, 2Y and 3A, 4A, 3Y, 4Y. The control inputs 1A, 2A, 3A and 4A are connected to Arduino ports 0,1,2, and 3 respectively. 1Y and 2Y are connected across the terminals of the left wheel motor. Similarly, 3Y and 4Y are connected across the terminals of the right wheel motor.

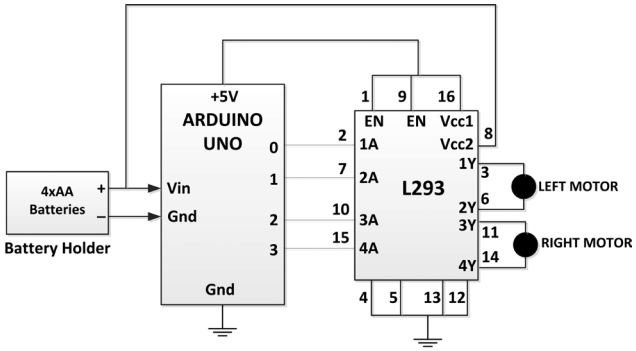

Figure 2.93 Circuit diagram of the project

2.15.3 Program Listing

Figure 2.94 shows the program listing (program: robot1). At the beginning of the program L1A and L2A are assigned to ports 0 and 1, and R3A and R4A are assigned to ports 3 and 4 respectively. Inside the setup routine L1A,L2A,R3A and R4A are configured as outputs.

The following functions are developed to control the motor:

StopMotor: This function stops both motors, therefore stopping the robot, by sending logic 0 to all the control inputs of L293.

FORWARD: This function moves the robot forward. This is achieved by configuring both motors to rotate in the same direction. The function accepts an argument which sets the time in seconds the motors should run. The robot is stopped at the end of the function.

REVERSE: This function moves the robot in reverse direction. This is achieved by configuring both motors to rotate in the same direction (opposite to the forward direction). Again, the duration the motors should run is specified in the function argument. The robot is stopped at the end of the function.

LEFT: This function turns the robot left. This is achieved by stopping the left hand side motor and activating the right hand side motor. The motor run time is specified as an argument. The robot is stopped at the end of the function. Another way to turn left is to reverse the left motor and set the right motor to go forward. This gives a sharper turn with smaller radius.

RIGHT: This function turns the robot right. This is achieved by stopping the right hand side motor and activating the left hand side motor. The motor run time is specified as an argument. The robot is stopped at the end of the function. Another way to turn right is to

reverse the right motor and sset the left motor to go forward. This gives a sharper turn with smaller radius.

StopRobot: This function simply waits forever while the motor is stopped.

The robot moves as specified in the following statements:

```
FORWARD(3);          // Go FORWARD 3 seconds
delay(2000);         // Wait 2 secs
REVERSE(3);          // REVERSE 3 seconds
delay(2000);         // Wait 2 secs
RIGHT(3);            // RIGHT 3 seconds
delay(2000);         // Wait 2 secs
LEFT(3);             // LEFT 3 secs
delay(2000);         // Wait 2 secs
StopRobot();         // Stop the robot
```

```
/*****************************************************************
*                 MOBILE ROBOT CONTROL
*                 ====================
*
* In this project a mobile robot is connected to the Arduino Uno.
* The robot is controlled as follows:
*
* Forward 3 seconds
* Stop for 2 seconds
* Reverse for 3 seconds
* Stop for 2 seconds
* Right turn and go for 3 seconds (keep turning right)
* Stop for 2 seconds
* Left turn for 3 seconds (keep turning left)
* Stop
*
* The aim of this project is to show how the basic movements of a
* robot can be done using an Arduino Uno
*
*
* File  : robot1
* Date  : September 2017
* Author: Dogan Ibrahim
*****************************************************************/
#define L1A 0                                   // LEFT motor 1A
#define L2A 1                                   // LEFT motor 2A
#define R3A 2                                   // RIGHT motor 3A
#define R4A 3                                   // RIGHT motor 4A
```

```
//
// Configure 1A, 2A, 3A, 4A as outputs
//
void setup()
{
  pinMode(L1A, OUTPUT);
  pinMode(L2A, OUTPUT);
  pinMode(R3A, OUTPUT);
  pinMode(R4A, OUTPUT);
}

//
// This function stops the motor
//
void StopMotor()
{
  digitalWrite(L1A, LOW);
  digitalWrite(L2A, LOW);
  digitalWrite(R3A, LOW);
  digitalWrite(R4A, LOW);
}

//
// This function moves the robot forward for tim seconds. Both
// motors rotate in the same direction
//
void FORWARD(char tim)
{
  digitalWrite(L1A, LOW);
  digitalWrite(L2A, HIGH);
  digitalWrite(R3A, LOW);
  digitalWrite(R4A, HIGH);
  delay(tim*1000);
  StopMotor();
}

//
// This function reverses the motor for tim seconds. Both motors
// rotate in the same direction
//
void REVERSE(char tim)
{
  digitalWrite(L1A, HIGH);
  digitalWrite(L2A, LOW);
  digitalWrite(R3A, HIGH);
  digitalWrite(R4A, LOW);
```

```
  delay(tim*1000);
  StopMotor();
}

//
// This function turns the motor LEFT for tim seconds. Left motor
// is stopped and the right motor is activated
//
void LEFT(char tim)
{
  digitalWrite(L1A, LOW);
  digitalWrite(L2A, LOW);
  digitalWrite(R3A, LOW);
  digitalWrite(R4A, HIGH);
  delay(tim*1000);
  StopMotor();
}

//
// This function just waits (used to terminate the loop)
//
void StopRobot()
{
  while(1);
}

// This function turns the motor RIGHT for tim seconds. Right motor
// is stopped and the left motor is activated
//
void RIGHT(char tim)
{
  digitalWrite(L1A, LOW);
  digitalWrite(L2A, HIGH);
  digitalWrite(R3A, LOW);
  digitalWrite(R4A, LOW);
  delay(tim*1000);
  StopMotor();
}
//
// Main program loop. Control the robot
//
void loop()
{
  FORWARD(3);                                    // Go FORWARD 3 seconds
  delay(2000);                                   // Wait 2 secs
  REVERSE(3);                                    // REVERSE 3 seconds
```

```
  delay(2000);                                    // Wait 2 secs
  RIGHT(3);                                       // RIGHT 3 seconds
  delay(2000);                                    // Wait 2 secs
  LEFT(3);                                        // LEFT 3 secs
  delay(2000);                                    // Wait 2 secs
  StopRobot();                                    // Stop the robot
}
```

Figure 2.94 Program listing

2.16 PROJECT 15 – Obstacle Avoiding Robot (Arduino Uno)

In this project a small mobile robot is used as in Project 14. In addition, an ultrasonic transmitter/receiver module is used to sense the distance of an object in-front of the robot as the robot moves forward. If the distance is less than 50cm (can be changed if required) then the robot turns left until it finds a clearance further than 50cm with no objects and it then goes forward. The robot continues to work in this manner.

2.16.1 Block Diagram

The block diagram of the project is shown in Figure 2.95.

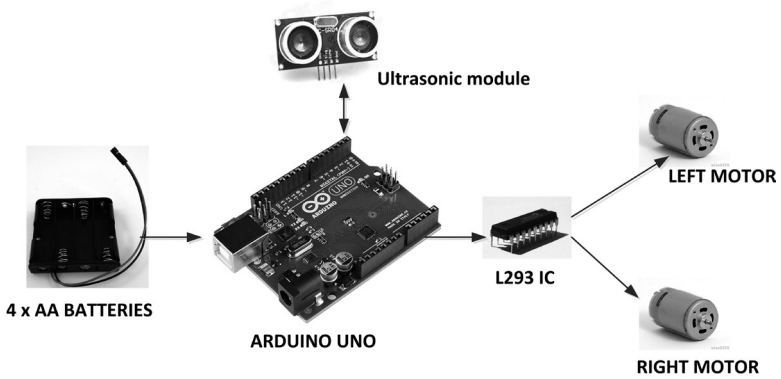

Figure 2.95 Block diagram of the project

2.16.2 Circuit Diagram

The circuit diagram of this project is similar to the one given in Figure 2.93 but here an HC-SR04 type ultrasonic transmitter/receiver module is added to the circuit (see Figure 2.96). This module has the following specifications:

- Operating voltage (current): 5V (2mA) operation
- Detection distance: 2cm – 450cm
- Input trigger signal: 10us TTL
- Sensor angle: not more than 15 degrees

The sensor module has the following pins:

Vcc: +V power
Trig: Trigger input
Echo: Echo output
Gnd: Power ground

Figure 2.96 Ultrasonic transmitter/receiver module

The principle of operation of the ultrasonic sensor module is as follows:

- A 10us trigger pulse is sent to the module
- The module then sends eight 40kHz square wave signals and automatically detects the returned (echoed) pulse signal
- If an echo signal is returned the time to receive this signal is recorded
- The distance to the object is calculated as:

Distance to object (in metres) = (time to received echo in seconds * speed of sound) / 2

The speed of sound is 340 m/s, or 0.034 cm/μs

Therefore,

Distance to object (in cm) = (time to received echo in μs) * 0.034 / 2

or,

Distance to object (in cm) = (time to received echo in μs) * 0.017

Figure 2.97 shows the principle of operation of the ultrasonic sensor module. For example, if the time to receive the echo is 294 microseconds then the distance to the object is calculated as:

Distance to object (cm) = 294 * 0.017 = 5 cm

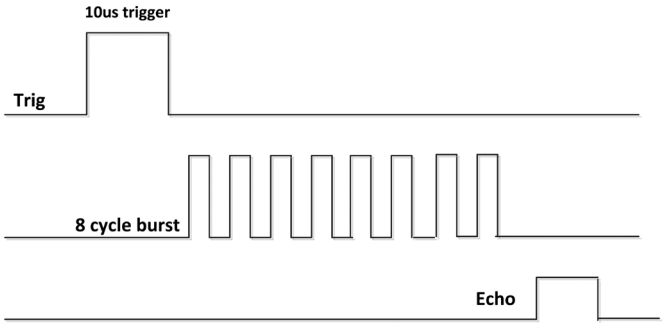

Figure 2.97 Operation of the ultrasonic sensor module

Figure 2.98 shows the circuit diagram of the project. L293 and the wheel motors are connected as in the previous project. The ultrasonic sensor is powered from the Arduino Uno and the Trig and Echo pins of the ultrasonic sensor are connected to Arduino port pins 5 and 6 respectively.

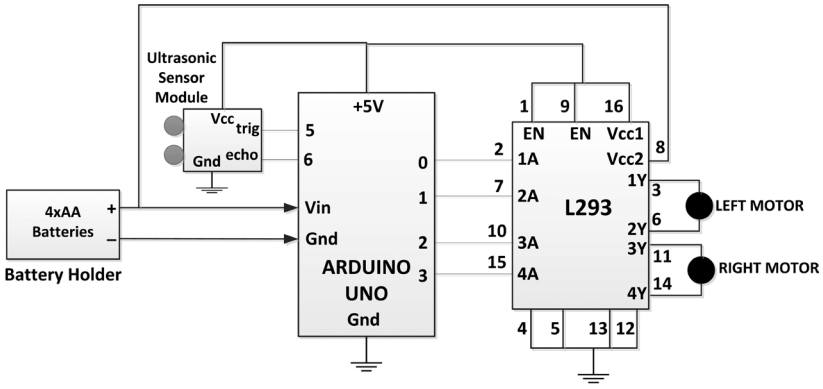

Figure 2.98 Circuit diagram of the project

Figure 2.99 shows the robot assembly with the ultrasonic sensor module mounted in a hole near the front of the robot chassis using glue (it might be preferable to mount the sensor in the middle right in-front of the robot).

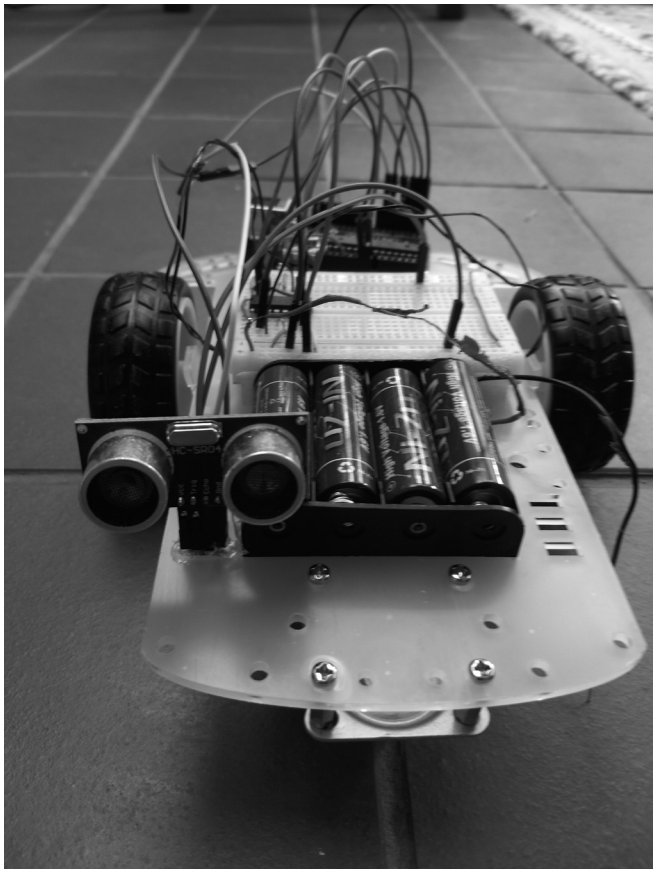

Figure 2.99 Robot assembly with the ultrasonic module

2.16.3 Program Listing

The program listing is shown in Figure 2.100 (program: robot2). At the beginning of the program L1A and L2A are assigned to ports 0 and 1, and R3A and R4A are assigned to ports 3 and 4 respectively. Also, ultrasonic sensor pins trig and echo are assigned to ports 5 and 6 respectively. Inside the setup routine L1A,L2A,R3A, R4A, and trig are configured as outputs, and echo is configured as input. Allowable distance to any object is set to 50cm.

Functions **FORWARD** and **LEFT** move the robot forward and turn to left as described in the previous project. Function **GetDistance** reads the distance to an object. The trigger pulse is sent initially for 10 microseconds. The program then uses the built-in function **pulseIn** and waits until the echo is received from the sensor. The time duration to echo is returned in microseconds by the **pulseIn** function. The distance to the object is then calculated in cm by multiplying the time duration by 0.017 and this value is returned to the main program loop. Inside the main program loop the distance to the objects is calculated and if the distance is less than or equal to 50cm then the robot turns left until the it finds a clearance with a distance more than 50cm. When this clearance is found the robot moves forward. This process is repeated forever until stopped manually by the user.

```
/*****************************************************************
*                 OBSTACLE AVOIDING ROBOT
*                 =======================
*
* This is an obstacle avoiding robot project. An ultrasonic sensor
* module is used to detect the distance in-front of the robot to
* the objects as the motor moves forward. If the distance to the
* object is 50cm or less then the robot turns left until it finds a
* clearance with no objects within 50cm and it then goes forward.
* This process continues forever until stopped manually
*
*
* File  : robot2
* Date  : September 2017
* Author: Dogan Ibrahim
*****************************************************************/
//
// Motor pin definitions
//
#define L1A 0                                       // LEFT motor 1A
#define L2A 1                                       // LEFT motor 2A
#define R3A 2                                       // RIGHT motor 3A
#define R4A 3                                       // RIGHT motor 4A

//
// Ultrasonic sensor module definitions
//
#define trig 5                                          // Trigger pin
#define echo 6                                          // Echo pin
float distance;
float allowed_distance = 50.0;                          // Allowed distance to
object

//
// Configure 1A, 2A, 3A, 4A as outputs. Also trig is an output and
// echo is an input. Stop the robot to start with
//
void setup()
{
  pinMode(L1A, OUTPUT);
  pinMode(L2A, OUTPUT);
  pinMode(R3A, OUTPUT);
  pinMode(R4A, OUTPUT);
  pinMode(trig, OUTPUT);
  pinMode(echo, INPUT);
  StopRobot();
```

```
}

//
// This function stops the robot
//
void StopRobot()
{
  digitalWrite(L1A, LOW);
  digitalWrite(L2A, LOW);
  digitalWrite(R3A, LOW);
  digitalWrite(R4A, LOW);
  delay(2000);
}

//
// This function moves the robot forward
//
void FORWARD()
{
  digitalWrite(L1A, LOW);
  digitalWrite(L2A, HIGH);
  digitalWrite(R3A, LOW);
  digitalWrite(R4A, HIGH);
}

//
// This function turns the motor LEFT
//
void LEFT()
{
  digitalWrite(L1A, LOW);
  digitalWrite(L2A, LOW);
  digitalWrite(R3A, LOW);
  digitalWrite(R4A, HIGH);
}

//
// This function reads the distance to the object and returns the
// distance in cm
//
float GetDistance()
{
   long duration;
   float DistanceToObject;
   digitalWrite(trig, LOW);                    // Clear trig
   delay(80);         // WAit to settle
```

```
   digitalWrite(trig, HIGH);                    // Set trig HIGH
   delayMicroseconds(10);                       // for 10 microseconds
   digitalWrite(trig, LOW);                     // Clear trig
   duration = pulseIn(echo, HIGH);              // Read the echo duration
   DistanceToObject = duration * 0.017;         // Distance in cm
   return (DistanceToObject);                   // Return distance
}
//
// Main program loop. Control the robot to avoid obstacles
//
void loop()
{
  distance = GetDistance();                     // Get distance to object
  if(distance <= allowed_distance)              // If obstacle too close
  {
     LEFT();                                    // Turn left
  }
  else FORWARD();                               // Go forward
}
```

Figure 2.100 Program listing

Notice that in this program the left turn was done by stopping the left hand motor and setting the right hand motor to go forward. In a small area it is more efficient to left turn by reversing the left hand motor and setting the right hand motor to go forward. This gives a much smaller radius of return, and is therefore suitable for small areas or if the allowable object distance is small.

2.17 PROJECT 16 – Line Following Robot (Arduino Uno)

Line following robots are popular projects among microcontroller programmers, especially in robotics applications. In these applications a wide and dark black line is drawn on a large paper (usually in the form of a circle) and the robot moves on this line.

In this project a small mobile robot is used as in the previous projects. Two Light Dependent resistors (LDRs) are mounted in-front of the robot, close to the floor. The robot follows a dark black line drawn on a large paper.

2.17.1 Block Diagram

Line following robots can be designed using two different types of sensors: IR transmitter/receiver sensor modules, and LDR sensors. IR transmitter/receiver sensor modules (see Figure 2.101 for an example sensor module TCRT5000) transmit IR signals and the reflected signal is detected by an IR sensor. The dark line absorbs the transmitted light and only a small amount of the signal is reflected back. The white part of the line on the other hand reflects most of the IR light. The IR receiver is like a switch and usually gives out a logic HIGH signal when the black line is detected. The robot is then programmed to follow the black line. IR transmitter/receiver sensor modules have the advantages that they are not affected much from the ambient lighting conditions.

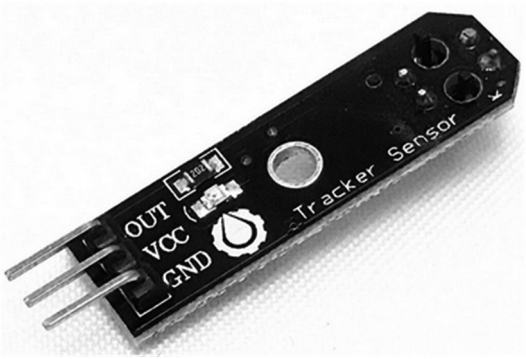

Figure 2.101 TCRT5000 IR transmitter/receiver module

LDR are light dependent resistors whose resistances depend upon the ambient light. When exposed to light, the resistance is low, and when it is dark the resistance increases sharply. Figure 2.102 shows a typical resistance-light curve of a typical LDR.

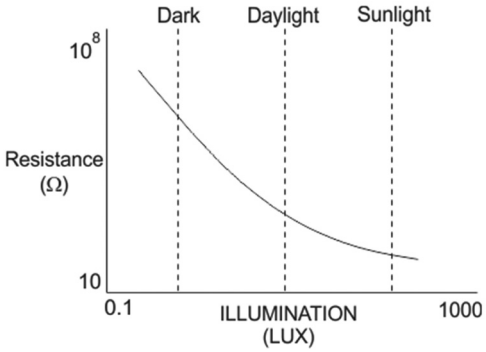

Figure 2.102 Typical LDR resistance-light curve

In this project two small LDRs are used to detect the dark line and then the robot is controlled to follow the dark line. LDR has the advantage that it is much cheaper than an IR sensor module, but it has the disadvantage that it is very sensitive to ambient light conditions, and LDRs are very slow to respond to light changes. Because of this, it is advisable to shine a bright white LED on the area where the robot is moving to minimize the effects of ambient light. When an LDR is above the dark line its resistance increases sharply, and when on the white paper, its resistance decreases. The LDRs are mounted in-front of the robot close to the floor and are positioned on the robot chassis that one LDR is on the left of the line and the other LDR is on the right of the line. The LDRs can be in 3 different positions with respect to the dark line. Figure 2.103 shows these cases. In case 1, the robot moves forward since the dark line is between the two LDRs. In case II, the left LDR is on the dark line and the robot must be moved to the left. In case III, the right LDR is on the dark line and the robot must be moved to the right.

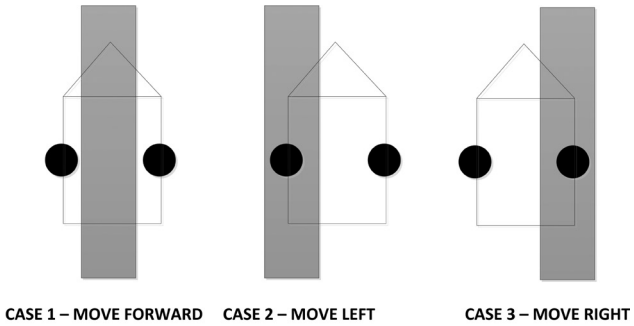

Figure 2.103 Robot movements to follow the dark line

The block diagram of the project is shown in Figure 2.104. A bright white LED is mounted in-front of the robot to brighten the path so that the effects of the ambient light can be minimized. The LDRs should be mounted about 5mm above the surface.

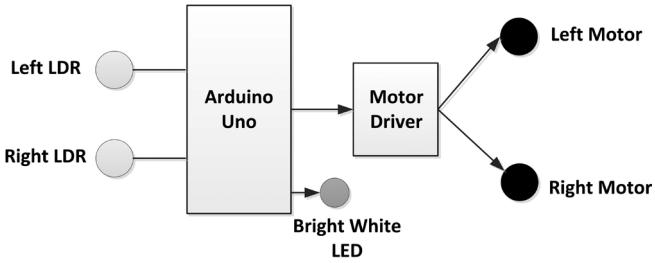

Figure 2.104 Block diagram of the project

2.17.2 Circuit Diagram

The circuit diagram of this project is shown in Figure 2.105. The two LDRs are connected to the analog inputs of the Arduino Uno through 15K pull-up resistor circuits. Left LDR is connected to analog input A1 and the right LDR is connected to analog input A0. The white bright LED is turned on continuously. The chosen LED was connected directly to the +ve terminal of the battery. It has a forward current of 30mA and a voltage drop of 4V. The current limiting resistor for the LED was chosen as 68 Ohms. The remainder of the circuit is same as in the previous projects where the wheel motors are driven using the L293 IC. The motors were powered from the 6V battery. Notice that the robot moves very fast and the sensors may not be fast enough to sense the dark line. If this is the case, it is recommended to slow down the motor, e.g power it from 3V (i.e. connect pin 8 of L293 to +3V instead of to +6V). An alternative is to use faster sensors such as IR transmitter/receiver modules as shown in Figure 2.101.

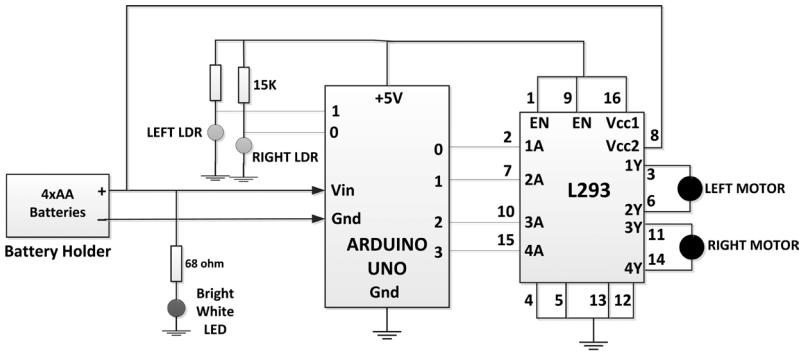

Figure 2.105 Circuit diagram of the project

Figure 2.106 and Figure 2.107 show the robot assembly with the two LDR sensors mounted in-front of the robot together with the white light. The LDR used in this project is shown in Figure 2.108. The LDRs are mounted at the end of small straws with the wires routed inside the straws.

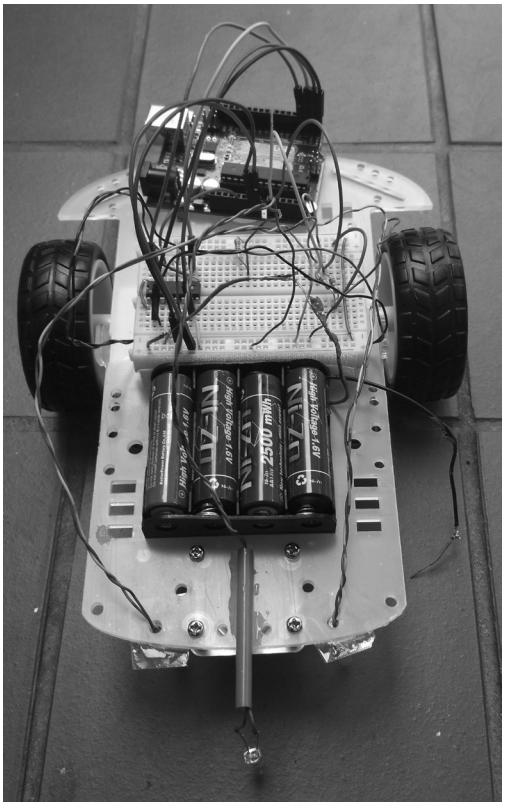

Figure 2.106 The robot assembly (top view)

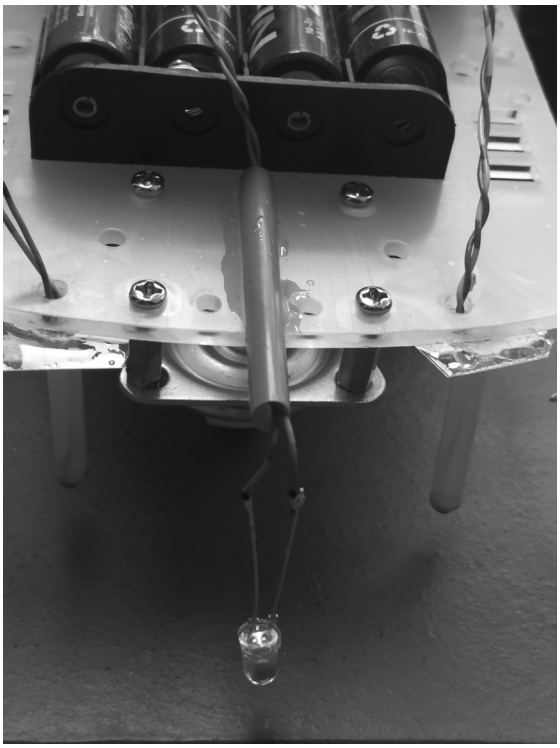

Figure 2.107 The robot assembly (front view)

Figure 2.108 LDR used in the project

2.17.3 Program Listing

The LDRs are used to sense the dark line. Before writing the program it was necessary to read the analog output from an LDR when it was on the dark line and also on the white paper. The reading was displayed using the Serial Monitor of the Arduino. The program was then written using this value for the detection of the dark line. The code of the test program is given below (make sure the white LED is ON, and keep a record of variable **valright** below when the LDR is on and off the dark line. You should repeat the test with the left sensor **valleft** since the two sensors may have slight response differences):

```
int valright;

void setup()
{
      Serial.begin(9600);
}

void loop()
{
      valright = analogRead(0);
      Serial.println(valright);
      delay(1000);
}
```

Figure 2.109 shows the path used to test the program where sticky black tape was used to make the path. The complete program listing is shown in Figure 2.110 (program: robot3). At the beginning of the program the motor connections are defined and they are set as outputs in the setup routine. Inside the main program loop analog values of both LDRs are read. It was found earlier during the tests that a dark line is detected if the analog reading was over 500. Therefore, if the left LDR is over the dark line (**valleft > 500**) then the robot is turned left. Similarly, if the right LDR is over the dark line (**valright > 500**) then the robot is turned right. If none of the sensors are on the dark line then the robot moves forward.

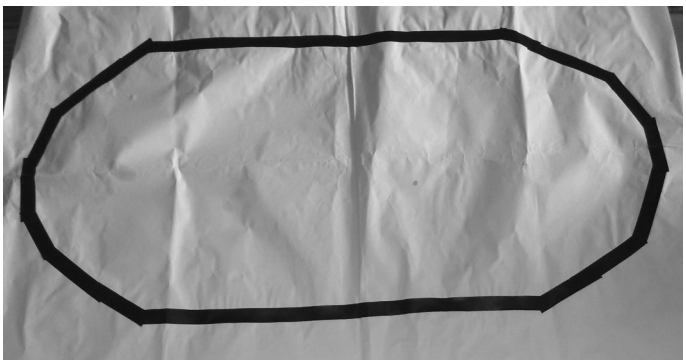

Figure 2.109 Path used to test the program

```
/*****************************************************************
*                  LINE FOLLOWING ROBOT
*                  ====================
*
* This is a line following robot project. In this project two LDR
* type sensors are monted in-front of the robot, close to the floor
* where the robot moves. The robot follows the black line drawn
* on the floor.
*
```

```
* File  : robot3
* Date  : September 2017
* Author: Dogan Ibrahim
*****************************************************************/
//
// Motor pin definitions
//
#define L1A 0                                   // LEFT motor 1A
#define L2A 1                                   // LEFT motor 2A
#define R3A 2                                   // RIGHT motor 3A
#define R4A 3                                   // RIGHT motor 4A

//
// Light Dependent Resistor (LDR) definitions
//
int valleft;
int valright;
//
// Configure 1A, 2A, 3A, 4A as outputs. Also, LDRs as inputs
//
void setup()
{
  pinMode(L1A, OUTPUT);
  pinMode(L2A, OUTPUT);
  pinMode(R3A, OUTPUT);
  pinMode(R4A, OUTPUT);
}

//
// This function moves the robot forward
//
void FORWARD()
{
  digitalWrite(L1A, LOW);
  digitalWrite(L2A, HIGH);
  digitalWrite(R3A, LOW);
  digitalWrite(R4A, HIGH);
}

//
// This function turns the motor LEFT
//
void LEFT()
{
  digitalWrite(L1A, HIGH);
  digitalWrite(L2A, LOW);
```

```
  digitalWrite(R3A, LOW);
  digitalWrite(R4A, HIGH);
}

//
// This function turns the motor RIGHT
//
void RIGHT()
{
  digitalWrite(L1A, LOW);
  digitalWrite(L2A, HIGH);
  digitalWrite(R3A, HIGH);
  digitalWrite(R4A, LOW);
}

//
// This function checks whether or not the robot is following the
// line. If the robot is to the left of the line then it is turned
// right, if it is to the right of the line then it is turned left,
// otherwise it goes forward
//
void loop()
{
   valright = analogRead(0);
   valleft = analogRead(1);

   if(valleft > 500)
    LEFT();
   else if(valright > 500)
    RIGHT();
   else FORWARD();
}
```

Figure 2.110 Program listing

This project can be improved by using a pair of IR transmitter/sensor modules instead of LDRs so that the detection of the dark line is not dependent upon the ambient light and also the IR sensors are much faster than the LDR sensors.

It is also possible to use more than 2 sensor modules for improved accuracy and smoother movement. As mentioned earlier, the motor should also be slowed down if the path is small and the sensors fail to detect the dark line as they pass over it. This is because the response of the LDR is very slow. Additionally, other marks such as a stop mark can be drawn on the line to stop the robot. A stop mark is usually a wide dark line drawn perpendicular to the normal path and it is detected when both sensors are on this dark line (see Figure 2.111).

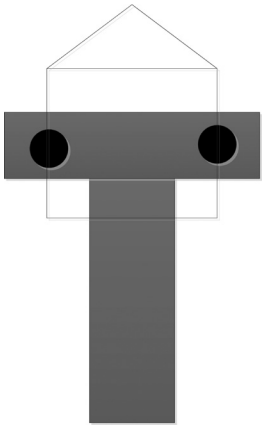

Figure 2.111 Stop mark on the robot path

2.18 PROJECT 17 – Mobile Robot Basic Control (Raspberry Pi Zero W)

This project is similar to Project 14 but here the Raspberry Pi Zero W is used as the controller instead of the Arduino Uno. The aim of this project is to show how to program the robot for simple movements.

In this project the motor performs the following movements:

- Go forward for 2 seconds
- Go reverse direction for 2 seconds
- Go forward for 2 seconds
- Turn left for 1 second
- Go forward for 2 seconds
- Turn right for 1 second
- Go reverse for 1 second
- Stop the motor

2.18.1 Block Diagram

In this project we want to control the robot using an RPi ZW and because of this the RPi ZW is mounted on the robot chassis. Additionally, a small breadboard is fixed using Velcro and mounted on the robot chassis. 4 x NiZn AA batteries are used to power the motor as in Project 14. Power to the RPi ZW is provided using a portable +5V charger with compatible micro USB connector (the ones used to charge Samsung mobile phones), mounted on top of the robot chassis.

The robot motors are controlled using an L293 IC as in Project 14. Figure 2.112 shows the block diagram of the project.

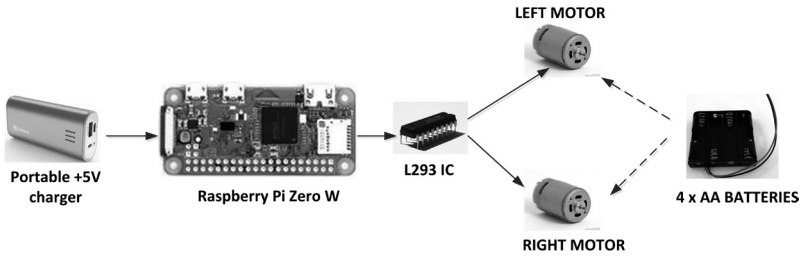

Figure 2.112 Block diagram of the project

2.18.2 Circuit Diagram

Figure 2.113 shows the circuit diagram of the project. The motors are powered from the battery pack, and the RPi ZW is directly powered from a portable +5V charger. 1Y, 2Y and 3Y, 4Y of the L293 IC are connected to the left and right wheel motors respectively. L293 control pins 1A, 2A, 3A and 4A are connected to RPi ZW port pins GPIO2 (pin 3), GPIO3 (pin 5), GPIO4 (pin 7), and GPIO17 (pin 11) respectively.

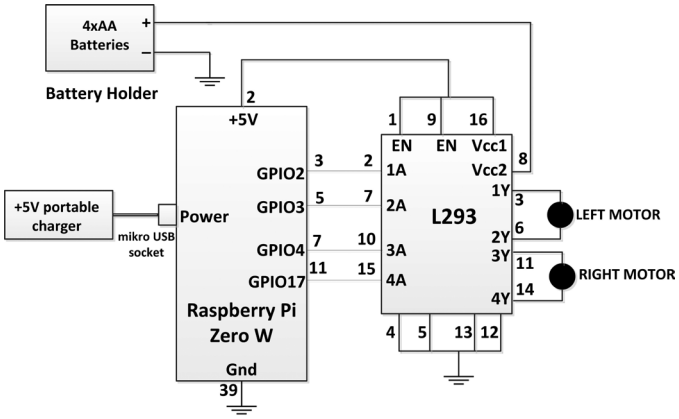

Figure 2.113 Circuit diagram of the project

Figure 2.114 shows the robot assembly with the RPi ZW, breadboard, battery pack, and the portable charger.

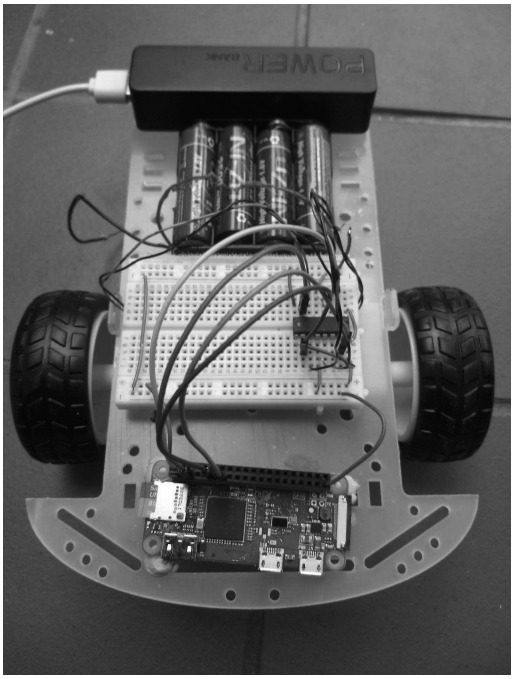

Figure 2.114 Robot assembly

2.18.3 Program Listing

Python is an interpretive language and user programs are not stored in the permanent memory of the processor. Programs are run interactively either from the command line or from the GUI menu. User programs are stored on an SD card or disk and must be run interactively after the processor is up and running. When the processor is rebooted or power is applied to the processor any user program does not run. It is important however, especially in robotics applications to run a user program independently and automatically after the processor is rebooted.

It is possible to run a Python program automatically at boot time. The steps are as follows:

- Edit file **rc.local** in folder **/etc/**

  pi@raspberrypi:~ $ **sudo nano /etc/rc.local**

- Write the full path and name of the program to run at boot time into this file. e.g. if the program name is **myfile.py** and it is in folder **/yourpath**, write the following statement inside file **rc.local**. Notice that the statement must be terminated with an ampersand character (**&**) so that the program runs in the background. If the ampersand is omitted and your program is in a loop then the RPi ZW will never complete its boot process:

  python /yourpath/myfile.py &

Notice that by default, unless changed, your path after logging in to the RPi ZW is **/home/pi**

Figure 2.115 shows the program listing of the project (program: robot1.py). The four basic motor movements are achieved as follows:

	FORWARD	REVERSE	LEFT	RIGHT
L1A	0	1	1	0
L2A	1	0	0	1
R3A	0	1	0	1
R4A	1	0	1	0

At the beginning of the program L293 pins are defined and configured as outputs. Functions are then created to move the motor forward, reverse, turn left, and turn right.

```
#-----------------------------------------------------------------
#                         BASIC ROBOT MOVEMENTS
#                         =====================
#
# This is a simple robot control project. In this project the RPi ZW
# is mounted on the robot chassis. The L293 IC is used as the motor
# driver. The aim of this project is to show how the basic robot
# movements can be done using the Python programming language
#
# In this project the robot moves as follows:
#
# Forward for 2 second
# Stop 2 seconds
# Reverse for 2 seconds
# Stop 2 seconds
# Forward for 2 seconds
# Stop 2 seconds
# Turn left for 1 second
# Stop 2 seconds
# Forward for 2 seconds
# Right for 1 second
# Stop 2 seconds
# Reverse for 1 second
# Stop
#
#
# File  : robot1.py
# Date  : August 2017
# Author: Dogan Ibrahim
#-----------------------------------------------------------------
import RPi.GPIO as GPIO
```

```
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
#
# L293 control pins
#
L1A = 2    # GPIO2
L2A = 3    # GPIO3
R3A = 17     # GPIO17
R4A = 27     # GPIO27
#
# Configure L293 control pins as outputs
#
GPIO.setup(L1A, GPIO.OUT)
GPIO.setup(L2A, GPIO.OUT)
GPIO.setup(R3A, GPIO.OUT)
GPIO.setup(R4A, GPIO.OUT)

#
# Stop the motor
#
def StopMotor():
  GPIO.output(L1A, 0)
  GPIO.output(L2A, 0)
  GPIO.output(R3A, 0)
  GPIO.output(R4A, 0)
  return

#
# Set forward movement
#
def FORWARD(tim):
  GPIO.output(L1A, 0)
  GPIO.output(L2A, 1)
  GPIO.output(R3A, 0)
  GPIO.output(R4A, 1)
  time.sleep(tim)
  return

#
# Set reverse movement
#
def REVERSE(tim):
  GPIO.output(L1A, 1)
  GPIO.output(L2A, 0)
  GPIO.output(R3A, 1)
```

```
  GPIO.output(R4A, 0)
  return

#
# Turn left
#
def LEFT(tim):
  GPIO.output(L1A, 1)
  GPIO.output(L2A, 0)
  GPIO.output(R3A, 0)
  GPIO.output(R4A, 1)
  time.sleep(tim)
  return

#
# Turn right
#
def RIGHT(tim):
  GPIO.output(L1A, 0)
  GPIO.output(L2A, 1)
  GPIO.output(R3A, 1)
  GPIO.output(R4A, 0)
  time.sleep(tim)
  return

#
# Start of the main program loop
#

FORWARD(2)
time.sleep(2)
REVERSE(2)
time.sleep(2)
FORWARD(2)
time.sleep(2)
LEFT(1)
FORWARD(2)
RIGHT(1)
time.sleep(2)
REVERSE(1)
StopMotor()

while True:
  pass
```

Figure 2.115 Program listing

The program name **robot1.py** was inserted into file **/etc/rc.local**. The contents of file /**etc/rc.local** is shown in Figure 2.116. The program starts automatically after reboot or when power is applied to the RPi ZW. Make sure that your program is free of errors before inserting it into file **/etc/rc.local**.

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi
#
# Insert our program path here
python /home/pi/robot1.py &
#
#
exit 0
```

Figure 2.116 rc.local file

After applying power to the robot, wait until the RPi ZW boots and the program should start automatically. You should remove your Python program name from file **/etc/rc.local** after testing and completing your project so that the program does not start every time you restart your RPi ZW.

2.19 PROJECT 18 – Obstacle Avoiding Robot (Raspberry Pi Zero W)

This project is similar to Project 15 but here the Raspberry Pi Zero W is used as the controller instead of the Arduino Uno. An ultrasonic transmitter/receiver module is used to sense the distance of an object in-front of the robot as the robot moves forward. In addition, a buzzer is connected to the robot. If the distance is less than 50cm (can be changed if required) then the robot sounds the buzzer and turns left until it finds a clearance further than 40cm with no objects and it then goes forward.

2.19.1 Block Diagram

The block diagram of the project is shown in Figure 2.117.

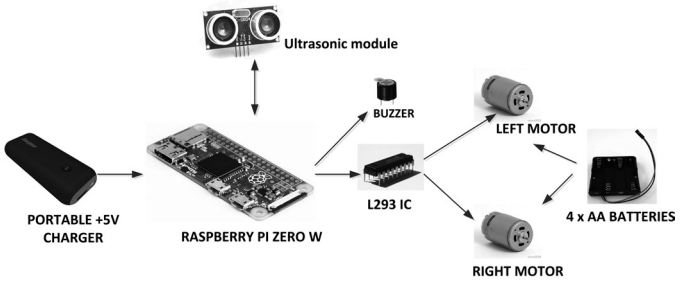

Figure 2.117 Block diagram of the project

2.19.2 Circuit Diagram

The circuit diagram of this project is shown in Figure 2.118. The ultrasonic sensor operates with +5V and its Vcc pin is connected to the +5V supply of the RPi ZW. The trig input of the ultrasonic sensor is connected to GPIO22 (pin 15) of the RPi ZW. The input pins of the RPi ZW are not +5V compatible and the voltage at any input pin must not exceed +3.3V otherwise the input circuitry can be damaged. The echo output of the ultrasonic sensor gives a +5V pulse when the echo signal is detected. This output is reduced to +3.3V using a resistive potential divider circuit consisting of a 1K and a 2K resistor and is then connected to GPIO4 (pin 7) of the RPi ZW. The voltage at the output of the potential divider resistor is (see Figure 2.118):

$$Vo = 5V \times 2K / (2K + 1K) = 3.3V$$

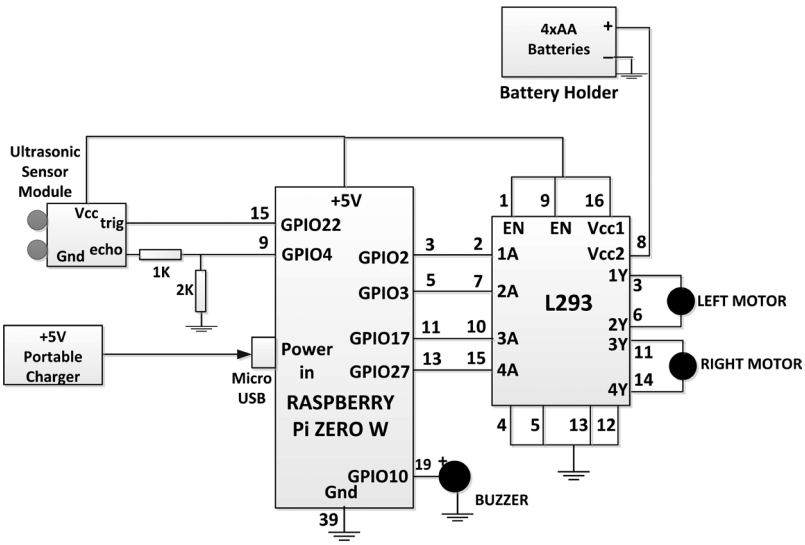

Figure 2.118 Circuit diagram of the project

Port pins GPIO2 (pin 3), GPIO3 (pin 5), GPIO17 (pin 11) and GPIO27 (pin 13) are connected to L293 pins 1A, 2A, 3A and 4A respectively. The wheel motors are connected to pins 3,6 and 11,14 of the L293 driver IC as before. An active buzzer is connected to GPIO10 (pin 19) port of the RPi ZW. The robot assembly is shown in Figure 2.119.

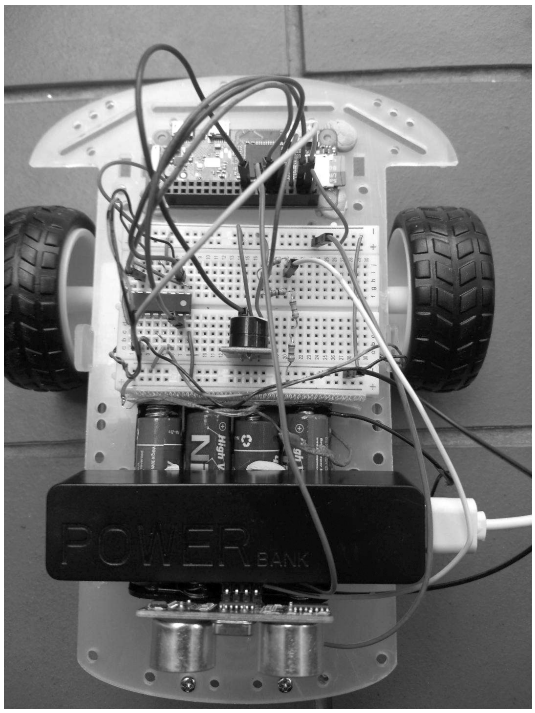

Figure 2.119 Robot assembly

2.19.3 Program Listing

Figure 2.120 shows the program listing (program robot2.py). The distance is measured by sending a 10 microsecond trigger pulse and then measuring the time taken to receive the echo pulse. Here, the **time.time()** function is used after sending the trigger pulse and the same function is used as soon as the echo pulse is received. The difference between the two times is the time taken to receive the echo pulse. This time is divided by 2 and the distance to the object is found in cm as follows:

Speed of sound = 340 m/s, or 34000cm/s

Distance to object (cm) = 34000 x time / 2

where, time is in seconds and it is the time taken to receive the echo pulse. We can re-write the above equation as:

Distance to object (cm) = 17000 x time

If the distance to the objects is less than or equal to the **Allowed_Distance** (which is set to 50cm) then the buzzer is sounded and the robot turns left until a clearance can be seen with no objects within the **Allowed_Distance**. The robot then moves forward. This process is repeated forever until stopped manually.

The program name robot2.py must be included in file **/etc/rc.local** in the following format so that the program starts as soon as the RPi ZW boots:

python /home/pi/robot2.py &

```
#--------------------------------------------------------------------
#                         OBSTACLE AVOIDING ROBOT
#                         =======================
#
# This is an obstacle avoding robot. An ultrasonic module is used
# to detect the disatnce in-front of the robot to the objects as
# the robot moves forward. If the distance to the objects is 40cm
# or less then the robot sounds a Buzzer and turns left until it
# finds a clearance with no obkects within 50cm and ut then goes
# forward.
#
#
# File  : robot2.py
# Date  : August 2017
# Author: Dogan Ibrahim
#--------------------------------------------------------------------
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
Allowed_Distance = 50

#
# L293 control pins
#
L1A = 2    # GPIO2
L2A = 3    # GPIO3
R3A = 17   # GPIO17
R4A = 27   # GPIO27

#
# Ultrasonic sensor pins
#
trig = 22    # GPIO22
echo = 4   # GPIO4
```

```
#
# BUZZER pin
#
Buzzer = 10    # GPIO10
#
# Configure L293 control pins,trig and Buzzer as outputs.
# Also, configure echo as input
#
GPIO.setup(L1A, GPIO.OUT)
GPIO.setup(L2A, GPIO.OUT)
GPIO.setup(R3A, GPIO.OUT)
GPIO.setup(R4A, GPIO.OUT)
GPIO.setup(trig, GPIO.OUT)
GPIO.setup(echo, GPIO.IN)
GPIO.setup(Buzzer, GPIO.OUT)

#
# Set forward movement
#
def FORWARD():
  GPIO.output(L1A, 0)
  GPIO.output(L2A, 1)
  GPIO.output(R3A, 0)
  GPIO.output(R4A, 1)
  return

#
# Turn left
#
def LEFT():
  GPIO.output(L1A, 1)
  GPIO.output(L2A, 0)
  GPIO.output(R3A, 0)
  GPIO.output(R4A, 1)
  return

#
# Turn ON the Buzzer
#
def BUZZERON():
  GPIO.output(Buzzer, 1)
  return

#
# Turn OFF the Buzzer
```

```
#
def BUZZEROFF():
  GPIO.output(Buzzer, 0)
  return

def GetDistance():
  GPIO.output(trig, 0)    # Wait to settle
  time.sleep(0.08)
  GPIO.output(trig,1)     # Send trig
  time.sleep(0.00001)     # Wait 10 microseconds
  GPIO.output(trig, 0)    # Remove trig
  while GPIO.input(echo) == 0:    # Wait until echo is received
    start_time = time.time()    # Start time

  while GPIO.input(echo) == 1:    # Echo is received
    end_time = time.time()    # End time

  pulse_width = end_time - start_time # Pulse duration
  distance = pulse_width * 17150  # Distance in cm
  return distance     # Return distance

#
# Start of the main program loop. Measure the distance to
# obstacles and turn the robot left if an obstacle is detected
# in-front of the robot in less than the Allowed_Distance.
# The buzzer is also sounded
#
BUZZEROFF()

while True:
  obstacle = GetDistance()    # Get distance to obstacle
  if obstacle <= Allowed_Distance:  # If less than Allowed_Distance
    BUZZERON()       # Turn Buzzer ON
    LEFT()       # Turn left
  else:
    BUZZEROFF()        # Turn Buzzer OFF
    FORWARD()        # Go forward
```

Figure 2.120 Program listing

After applying power to the robot, wait until the RPi ZW boots and the program should start automatically. You should remove your Python program name from file **/etc/rc.local** after testing and completing your project so that the program does not start every time you restart your RPi ZW.

2.20 Summary

In this Chapter we have developed many projects using a DC motor with the Arduino UNO and Raspberry Pi Zero W.

In the next Chapter we shall be looking at the design of stepper motor projects, again using Arduino Uno and Raspberry Pi Zero W processors.

CHAPTER 3 • SIMPLE STEPPER MOTOR PROJECTS

3.1 Overview

Stepper motors are DC motors that rotate in small steps. These motors have several coils that are energized in sequence, causing the motor to rotate one step at a time. Stepper motors have the advantages that very precise positioning or speed control of the motor shaft can be achieved. These motors are used in many precision motion control applications.

There are basically two types of stepper motors: unipolar and bipolar.

3.1.1 Unipolar Stepper Motors

Unipolar stepper motors have four windings with a common center tap on each pair of windings (see Figure 3.1). Therefore, there are normally 5 or 6 leads depending on whether the common leads are joined or not.

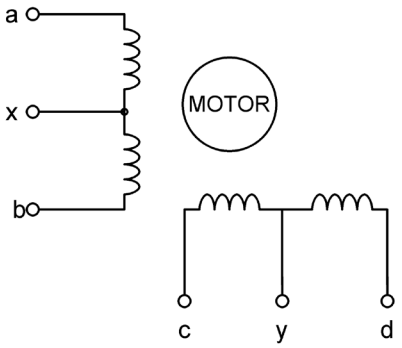

Figure 3.1 Unipolar stepper motor windings

Unipolar motors can be rotated in reverse by reversing the sequence of applied pulses. Unipolar stepper motors can be driven in full stepping mode or in half stepping mode. Most popular drive modes are 1 phase full-step, 2 phase full-step, and 2 phase half-step.
In 1 phase full-step mode, as shown in Table 3.1, each motor winding receives one pulse per step. This mode has the disadvantage that the available torque is low.

Step	a	c	b	d
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Table 3.1 1 Phase full-step mode

In 2 phase full-step mode, as shown in Table 3.2, two motor windings receive pulses per step. The advantage of this mode is that a higher torque is available from the motor.

Step	a	c	b	d
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1

Table 3.2 2 Phase full-step mode

In 2 phase half-step mode, as shown in Table 3.3, two motor windings sometimes receive pulses and sometimes only one winding receives a pulse. Because the motor is driven at half-step mode, 8 steps are required to complete a cycle instead of 4. This mode gives higher precision, but at the expense of lower torque.

Step	a	c	b	d
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

Table 3.3 2 Phase half-step mode

3.1.2 Bipolar Stepper Motors

Bipolar stepper motors have one winding per phase as shown in Figure 3.2. Bipolar motor driver circuits are more complicated since the current in the windings needs to be reversed in order to rotate them in the reverse direction.

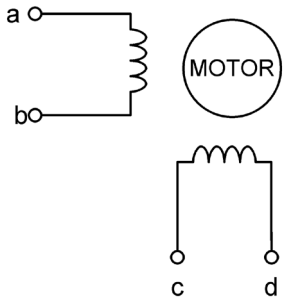

Figure 3.2 Bipolar stepper motor windings

Table 3.4 shows the steps required to drive a bipolar stepper motor. Here, + and – signs refer to the polarity of the voltages applied to the motor leads.

Step	a	c	b	d
1	+	-	-	-
2	-	+	-	-
3	-	-	+	-
4	-	-	-	+

Table 3.4 2 Bipolar stepper motor driving sequence

3.1.3 Speed of a Stepper Motor

The speed of a stepper motor depends on the time between the pulses given of its windings. For faster speeds, the pulses must be given with shorter delays between them. If T is the time between the pulses and β is the step constant of the motor, the moto rotates by β/T steps in one second. Since a complete revolution is 360º, the number of revolutions in a second is β/360T. The speed of a motor is normally quoted in RPM and therefore:

$$RPM = 60\beta/360T$$

or,

$$RPM = \beta/6T$$

where, RPM is the number of revolutions per minute, β is the step constant of the motor in degrees, and T is the time between the steps in seconds.

As an example, assume that the step constant of a stepper motor is 10 degrees (β = 10º). If we want to rotate this motor at a speed of 1000 RPM (assuming that the motor is capable of rotating this fast), the time between the pulses is calculated as:

$$T = \beta/6RPM = 10 / (6 \times 1000) = 1.66ms$$

Therefore, the pulses between each step must be 1.66ms.

3.1.4 Movement of the Motor Shaft

In some applications we may want the motor shaft to rotate a specified amount and we need to know how many pulses to send to the motor. If β is the step constant of the motor and we want the shaft to rotate by v degrees, the required number of pulses is given by:

$$n = v/\beta$$

For example, assuming that the step constant is 5º (β = 5), and that we want the motor to rotate by 200 degrees, the required number of steps is:

$$n = 200/5 = 40$$

3.1.5 Motor Rotation Time

Sometimes we may want the motor to rotate for a given time and we want to know how many pulses to apply to the motor. If T is the time between the pulses and we send n pulses

to the motor, the rotation time T0 can be calculated as follows:

$$T0 = nT$$

For example, assuming that the time between the pulses is 1ms, if we want the motor to rotate for 5 seconds, the required number of pulses is given by:

$$N = T0/T = 5/0.001 = 5000$$

Stepper motors can be driven by several ways, such as using bipolar transistors, using MOSFET transistors, or using integrated circuits such as L293, ULN2003 and so on. In this Chapter we shall be using a ULN2003 based driver circuit.

3.2 PROJECT 1 – Basic Stepper Motor Control

This is a very simple project where a small unipolar stepper motor is used. The motor is rotated in one direction for 2 complete revolutions. It is then stopped for 2 seconds, and then rotated in the reverse direction for 2 complete revolutions.

3.2.1 Block Diagram

In this project a small 28BYJ-48 type unipolar stepper motor (see Figure 3.3) is used. This motor has the following specifications:

Rated voltage:	5V
Number of phases:	4
Gear ratio:	64
Frequency:	100Hz
Stride angle:	5.625º/64

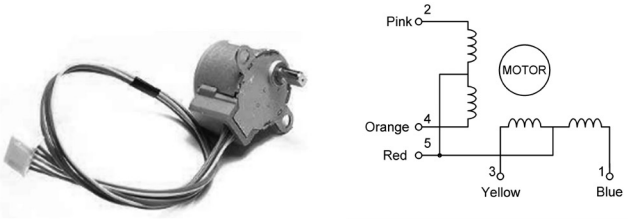

Figure 3.3 28BYJ-48 unipolar stepper motor

The block diagram of the project is shown in Figure 3.4.

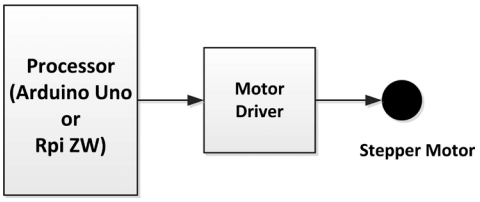

Figure 3.4 Block diagram of the project

3.2.2 Circuit Diagram (Arduino Uno)

The motor is driven using a ULN2003 IC based motor driver module shown in Figure 3.5 together with its circuit diagram. This module has four input connections labelled IN1, IN2, IN3 and IN4. The motor is plugged in to the socket in the middle of the module. Four LEDs, labelled A, B, C, D are provided to see the stats of the motor windings. Power to the module is applied through the bottom two header pins at the right hand side of the module. The LEDs can be enabled by shorting the two top header pins at the right hand side of the module. In this project the module was powered from an external +5V DC power supply for the motor (it is recommended not to use the Arduino Uno +5V power supply as this may not provide enough current to drive the motor).

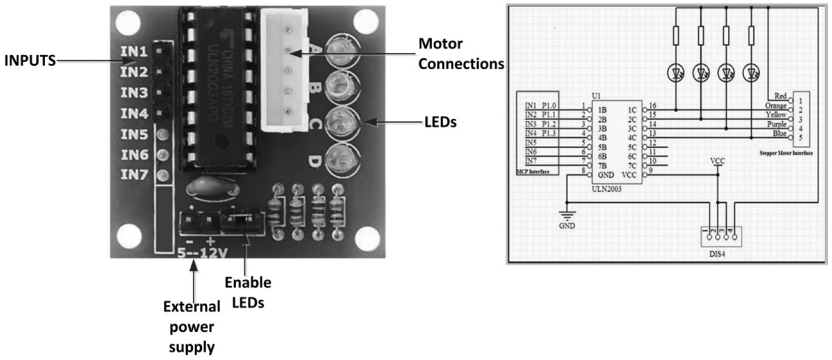

Figure 3.5 ULN2003 motor driver module

Figure 3.6 shows the Arduino Uno circuit diagram of the project. GPIO pins 0,1,2,3 of the Arduino Uno are connected to driver module inputs IN1,IN2,IN3 and IN4 respectively. The project with the stepper motor is shown in Figure 3.7

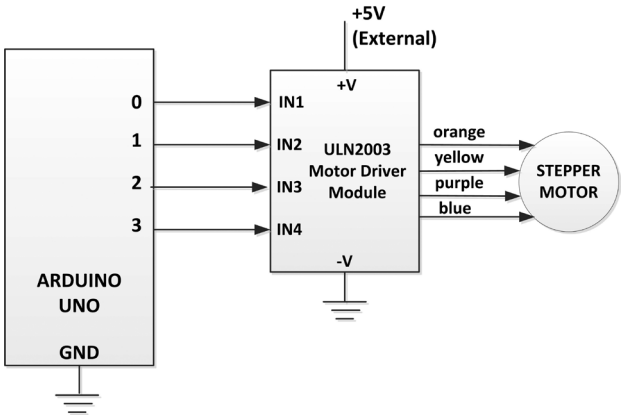

Figure 3.6 Circuit diagram of the project (Arduino Uno)

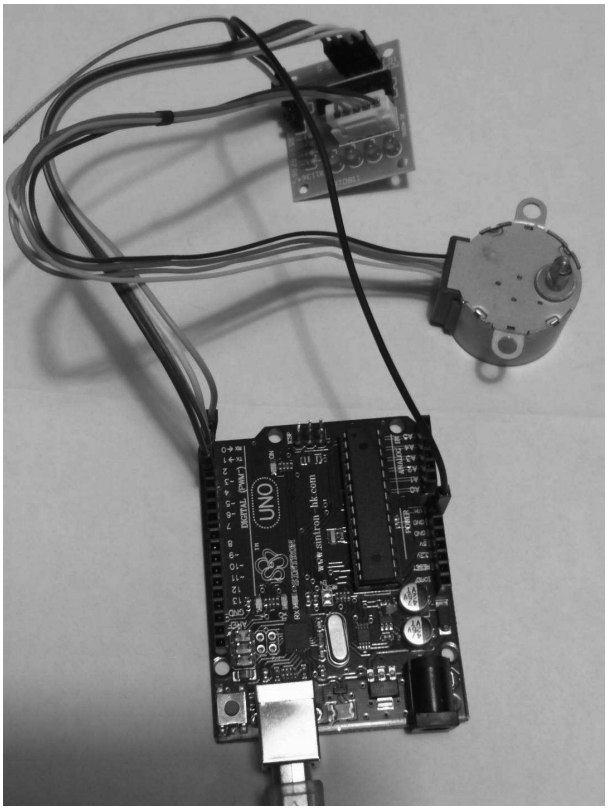

Figure 3.7 Project with the stepper motor

3.2.3 Circuit Diagram (Raspberry Pi Zero W)

Figure 3.8 shows the RPi ZW circuit diagram of the project.

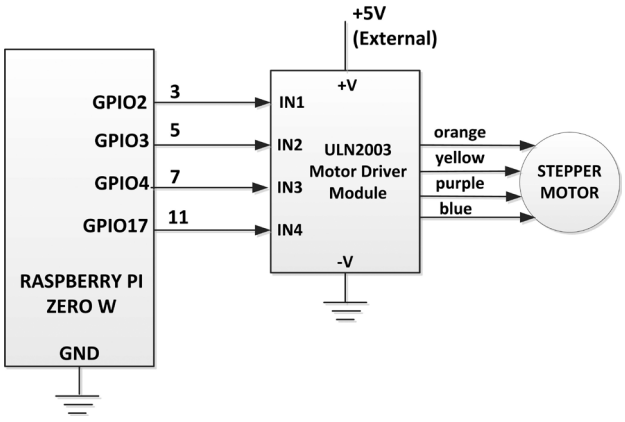

Figure 3.8 Circuit diagram of the project (RPi ZW)

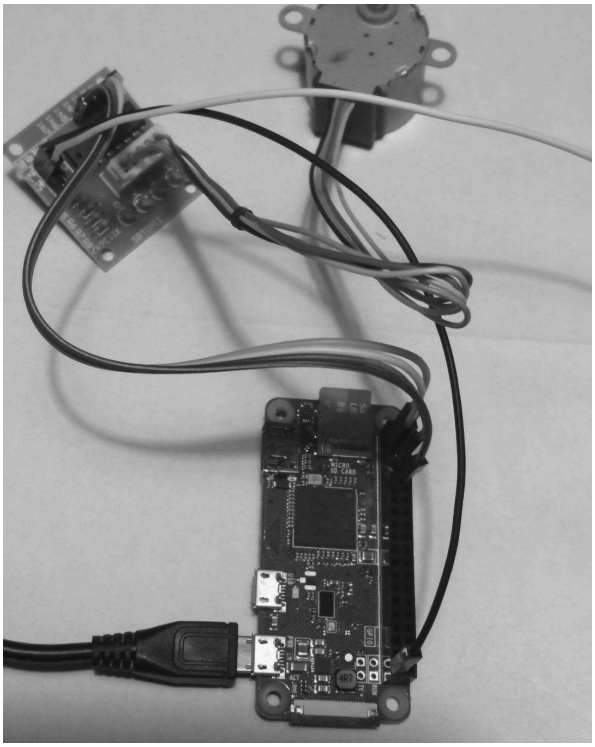

Figure 3.9 Project with the stepper motor

3.2.4 Driving the Stepper Motor

The 28BYJ-48 stepper motor can either be operated in full-step or in half-step modes.

Full-step Mode

In full-step mode there are 4 steps per cycle and 11.25 degrees/step, corresponding to 32 steps per one revolution of the internal motor shaft. Because the motor is geared with a gear ratio of 64 (in actual fact the gear ratio is 63.68395), the number steps for one external complete revolution is 2048 steps/revolution (512 cycles with 4 steps per cycle). Table 3.5 shows the motor winding sequence for the full-step mode (this sequence is repeated. Reversing the sequence reverses the direction of rotation).

Step	4 (orange) IN1	3 (yellow) IN2	2 (pink) IN3	1 (blue) IN4
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

Table 3.5 Full-step mode

Half-step Mode

The half-step mode with 8 steps per cycle is recommended by the manufacturer. In half-step mode we have 5.625 degrees/step, corresponding to 64 steps per one revolution of the internal motor shaft. Because the motor is geared with a gear ratio of 64, the number of steps for one external complete revolution is 4096 steps/revolution (512 cycles with 8 steps per cycle).

Table 3.6 shows the motor winding pulse sequence for the half-step mode (this sequence is repeated. Reversing the sequence reverses the direction of rotation).

Step	4 (orange) IN1	3 (yellow) IN2	2 (pink) IN3	1 (blue) IN4
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

Table 3.6 Half-step mode

3.2.5 Program Listing (Arduino Uno)

Arduino Uno has a built-in library to drive stepper motors. But unfortunately this library can be used to drive in full-step mode only. In this section we shall be seeing how to drive our stepper motor in both full-step mode with and without the library, and also in half-step mode without the library.

Full-Step Mode (Without Library)

Figure 3.10 shows the program listing driving the motor in full-step mode (program: stepper1) without the Arduino library. In this example the motor is rotated in one direction for 2 complete revolutions. It is then stopped for 2 seconds, and then rotated in the reverse direction for 2 complete revolutions. In this example the motor speed is set to 12 RPM. As described earlier, the speed depends on the delay inserted between each step. In Full-step mode there are 2048 steps in a complete revolution. The motor speed in RPM is given by the following equation:

$$RPM = 60 \times 10^3 / (2048 \times T)$$

or,

$$RPM = 29.3 / T$$

Where, RPM is the motor speed in revolutions per minute, and T is the delay between each step in milliseconds. We usually show the delay in microseconds and the above equation

becomes:

RPM = 29300 / T where T is in microseconds

At the beginning of the program motor driver pins IN1, IN2, IN3 and IN4 are assigned to GPIO pins 0, 1, 2, and 3 respectively. Then the motor speed is set to 12 RPM and the steps per revolution (variable **StepsPerRevolution**) is set to 512 for full-mode of operation. The pulses to be sent to the motor in Full-mode are stored in array **FullMode**. Inside the setup routine motor pins are configured as outputs and the time delay to be inserted between each step (**StepDelay**) is calculated. Inside the main program loop functions CLOCKWISE and ANTICLOCKWISE are created which rotate the motor clockwise and anticlockwise respectively. Required number of revolutions is sent as an argument to these functions.

Function CLOCKWISE sends 4 pulses to the motor with a delay of **StepDelay** between each pulse. This ensures that the motor speed is as required. This is repeated **StepsPerRevolution** times so that the motor makes a complete revolution. This whole process is then repeated **count** times which is the number of times we want the motor to rotate complete revolutions.

```
/*****************************************************************
*                 BASIC STEPPER MOTOR CONTROL
*                 ===========================
*
* In this project a unipolar stepper motor is controlled such that
* the motor is rotated clockwise for 2 revolutions, then after 2
* seconds delays the motor is rotated twice anticlockwise, and then
* it is stopped. The motor is controlled in FULL-STEP mode with
* the speed of 12 RPM.
*
* The motor has 2048 steps/revolutions in FULL-STEP mode, or 512
* cycles per revolution with 4 steps in every cyle
*
* In this project a ULN2003 type motor driver module is used
*
*
* File  : stepper1
* Date  : September 2017
* Author: Dogan Ibrahim
*****************************************************************/
//
// Stepper Motor pin definitions
//
#define IN1 0                                   // Motor orange
#define IN2 1                                   // Motor yellow
#define IN3 2                                   // Motor purple
#define IN4 3                                   // Motor blue
```

```
//
// Motor control parameters
//
int RPM = 12;                                          // Required speed
int StepsPerCycle = 512;                               // Steps per cycle
int StepDelay;                                         // Step Delay in
microseconds
int FullMode[4] = {B01100, B00110, B00011, B01001};

//
// Configure motor pins as outputs. Also calculate the time delay
// to be inserted between each step
//
void setup()
{
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  StepDelay = 29300 / RPM;
}

//
// This function rotates teh motor clockwise
//
void CLOCKWISE(int count)
{
  for(int j = 0; j < count; j++)
  {
     for(int m = 0; m < StepsPerCycle; m++)
     {
        for(int i = 4; i >= 0; i--)
        {
           SendPulse(i);
           delayMicroseconds(StepDelay);
        }
     }
  }
}

//
// This function rotate steh motor anticlockwise
//
void ANTICLOCKWISE(int count)
{
```

```
  for(int j = 0; j < count; j++)
  {
     for(int m = 0; m < StepsPerCycle; m++)
     {
        for(int i = 0; i < 4; i++)
        {
           SendPulse(i);
          delayMicroseconds(StepDelay);
        }
     }
  }
}

//
// Send pulss in FULL-MODE to the motor
//
void SendPulse(int k)
{
   digitalWrite(IN1, bitRead(FullMode[k], 0));
   digitalWrite(IN2, bitRead(FullMode[k], 1));
   digitalWrite(IN3, bitRead(FullMode[k], 2));
   digitalWrite(IN4, bitRead(FullMode[k], 3));
}

//
// Main program loop. Control the stepper motor as required:
// 2 rotations clockwise, 2 seconds delay, 2 rotations anticlockwise
//
void loop()
{
  CLOCKWISE(2);
  delay(2000);
  ANTICLOCKWISE(2);
  while(1);
}
```

Figure 3.10 Program listing (full-step mode without library)]

Half-Step Mode

Figure 3.11 shows the program listing driving the motor in half-step mode (program: stepper2) without the Arduino library (the Arduino library can only drive in full-step mode). In this example the motor is rotated in one direction for 2 complete revolutions. It is then stopped for 2 seconds, and then rotated in the reverse direction for 2 complete revolutions. In this example the motor speed is set to 12 RPM as before. In half-step mode there are 4096 steps in a complete revolution. The motor speed in RPM is given by the following equation:

$$RPM = 60 \times 10^3 / (4096 \times T)$$

or,

$$RPM = 14.648 / T$$

Where, RPM is the motor speed in revolutions per minute, and T is the delay between each step in milliseconds. We usually show the delay in microseconds and the above equation becomes:

$$RPM = 14648 / T \text{ where T is in microseconds}$$

The program in Figure 3.11 is very similar to the ne in Figure 3.10. Here, the pulse sequence to be sent to the motor (**HalfMode**) consists of 8 steps, and the delay between each pulse is different as shown in the above calculation. Also, the CLOCKWISE and ANTI-CLOSKWISE functions send 8 pulses per cycle instead of 4.

```
/*****************************************************************
*                 BASIC STEPPER MOTOR CONTROL
*                 ===========================
*
* In this project a unipolar stepper motor is controlled such that
* the motor is rotated clockwise for 2 revolutions, then after 2
* seconds delays the motor is rotated twice anticlockwise, and then
* it is stopped. The motor is controlled in HALF-STEP mode with
* the speed of 12 RPM
*
* The motor has 4096 steps/revolutions in HALF-STEP mode, or 512
* cycles per revolution with 8 steps in every cyle
*
* In this project a ULN2003 type motor driver module is used
*
*
* File  : stepper2
* Date  : September 2017
* Author: Dogan Ibrahim
*****************************************************************/
//
// Stepper Motor pin definitions
//
#define IN1 0                                   // Motor orange
#define IN2 1                                   // Motor yellow
#define IN3 2                                   // Motor purple
#define IN4 3                                   // Motor blue

//
// Motor control parameters
//
```

```
int RPM = 12;                                   // Required speed
int CyclesPerRevolution = 512;                  // Cycles per revolution
int StepDelay;                                  // Step Delay in microseconds
int HalfMode[8] = {B01000, B01100, B00100, B00110, B00010, B00011, B00001,
B01001};

//
// Configure motor pins as outputs. Also calculate the time delay
// to be inserted between each step
//
void setup()
{
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  StepDelay = 14648 / RPM;
}

//
// This function rotates teh motor clockwise
//
void CLOCKWISE(int count)
{
  for(int j = 0; j < count; j++)
  {
     for(int m = 0; m < CyclesPerRevolution; m++)
     {
        for(int i = 7; i >= 0; i--)
        {
           SendPulse(i);
           delayMicroseconds(StepDelay);
        }
     }
  }
}

//
// This function rotates the motor anticlockwise
//
void ANTICLOCKWISE(int count)
{
  for(int j = 0; j < count; j++)
  {
     for(int m = 0; m < CyclesPerRevolution; m++)
     {
```

```
        for(int i = 0; i < 8; i++)
        {
           SendPulse(i);
          delayMicroseconds(StepDelay);
        }
     }
  }
}

//
// Send pulses in HALF-MODE to the motor
//
void SendPulse(int k)
{
   digitalWrite(IN1, bitRead(HalfMode[k], 0));
   digitalWrite(IN2, bitRead(HalfMode[k], 1));
   digitalWrite(IN3, bitRead(HalfMode[k], 2));
   digitalWrite(IN4, bitRead(HalfMode[k], 3));
}

//
// Main program loop. Control the stepper motor as required:
// 2 rotations clockwise, 2 seconds delay, 2 rotations anticlockwise
//
void loop()
{
  CLOCKWISE(2);
  delay(2000);
  ANTICLOCKWISE(2);
  while(1);
}
```

Figure 3.11 Program listing (half-step mode)

Full-Step Mode (With the Library)

Figure 3.12 shows the program listing driving the motor in full-step mode (program: stepper3) using the Arduino stepper library. The Arduino stepper library supports the following functions:

Stepper(steps, pin1, pin2, pin3, pin4): This function creates a new instance of the Stepper class. This function must be declared at the top of the program before the setup routine. The number of parameters can be 2 or 4 depending on how the motor is wired. Parameter steps is the number of steps in one revolution.

setSpeed(rpm): This function sets the motor speed in RPM.

Step(steps): This function turns the motor a specific number of steps at the speed specified by the setSpeed function.

Function Stepper is called and the number of steps per revolution in full-mode (2048) and the motor pin connections are defined. Notice that the wiring sequence is slightly different when the library is used. Motor speed is set to 12 RPM in the setup routine and the number of required revolutions is set to 2. Inside the main program loop function **step** is called to rotate the motor required number of times in both directions.

```
/*****************************************************************
*                  BASIC STEPPER MOTOR CONTROL
*                  ===========================
*
* In this project a unipolar stepper motor is controlled such that
* the motor is rotated clockwise for 2 revolutions, then after 2
* seconds delays the motor is rotated twice anticlockwise, and then
* it is stopped. The motor is controlled in FULL-STEP mode with
* the speed of 12 RPM
*
* In this project a ULN2003 type motor driver module is used. This
* version of the program uses the Arduino stepper library
*
*
* File  : stepper3
* Date  : September 2017
* Author: Dogan Ibrahim
*****************************************************************/
#include <Stepper.h>
#define StepsPerRevolution 2048
Stepper MyMotor(StepsPerRevolution, 0, 2, 1, 3);      // Pulse sequence
// Motor definition
int RequiredRevolutions = 2;
int Revolutions;

void setup()
{
   MyMotor.setSpeed(12);                              // Set motor speed
   Revolutions = RequiredRevolutions * StepsPerRevolution;          //
Required number of revolutions
}

//
// Main program loop. Control the stepper motor as required:
// 2 rotations clockwise, 2 seconds delay, 2 rotations anticlockwise
//
void loop()
```

```
{
  MyMotor.step(Revolutions);
  delay(2000);
  MyMotor.step(-Revolutions);
  while(1);
}
```

Figure 3.12 Program listing (Full-step mode with the library)

3.2.6 Program Listing (Raspberry Pi Zero W)

In this section we will be controlling our stepper motor using the RPi ZW. The motor will be controlled in half-step mode as follows: 2 revolutions in the clockwise direction, 2 seconds delay, and 2 revolutions in the anticlockwise direction.

Figure 3.13 shows the program listing. At the beginning of the program the RPM is set to 12, steps per revolution to 512, and the delay is set so that the required RPM can be achieved. Notice that the delay is specified in seconds since the Python time delay function accepts its parameter in seconds. Then, the motor driver control pins are assigned to the GPIO ports and these ports are configured as outputs. The half-step sequence is stored in array **HalfStep**. Two functions are created: CLOCKWISE and ANTICLOCKWISE as in the Arduino Uno version of the project. The required number of revolutions is passed as an argument to these functions. Function **SendPulse** sends pulses to the motor in the correct sequence. This function has the following format. Array **HalfStep** is accessed using two indices, the first one being the **HalfStep** row, and the second one the bit inside the selected row.:

```
def SendPulse(k):
        GPIO.output(IN1, HalfStep[k][3])
        GPIO.output(IN2, HalfStep[k][2])
        GPIO.output(IN3, HalfStep[k][1])
        GPIO.output(IN4, HalfStep[k][0])
        return
```

Inside the main program loop the motor is rotated clockwise twice, then after 2 seconds delay it is rotated anticlockwise, and then it stops.

```
#-----------------------------------------------------------------
#                       BASIC CONTROL OF A STEPPER MOTOR
#                       ================================
#
# This is a simple stepper motor controller. In this project a
# small unipolar motor (28BYJ-48) is connected to the RPi ZW
# processor. The motor is controlled in half-step mode as follows:
#
# Turn 2 revolutions in clockwise direction
# Wait 2 seconds
# Turn 2 revolutions in anticlockwise direction
#
```

```
#
# File  : stepper1.py
# Date  : August 2017
# Author: Dogan Ibrahim
#-------------------------------------------------------------------------
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

RPM = 12
StepsPerRevolution = 512
StepDelay = 0.014648 / RPM

#
# Motor driver control pins
#
IN1 = 2    # GPIO2
IN2 = 3    # GPIO3
IN3 = 4    # GPIO4
IN4 = 17     # GPIO17

#
# Configure driver control pins as outputs
#
GPIO.setup(IN1, GPIO.OUT)
GPIO.setup(IN2, GPIO.OUT)
GPIO.setup(IN3, GPIO.OUT)
GPIO.setup(IN4, GPIO.OUT)

#
# Define Half-Step sequence
#
HalfStep = [[1,0,0,0],
            [1,1,0,0],
            [0,1,0,0],
            [0,1,1,0],
            [0,0,1,0],
            [0,0,1,1],
            [0,0,0,1],
            [1,0,0,1]]

#
# This function rotates the motor clockwise
#
def CLOCKWISE(count):
```

```
  global StepsPerRevolution
  for j in range(0,count):
    for m in range(0,StepsPerRevolution):
      for i in range(7, -1, -1):
        SendPulse(i)
        time.sleep(StepDelay)
  return

#
# This function rotates the motor anticlockwise
#
def ANTICLOCKWISE(count):
  global StepsPerRevolution
  for j in range (0,count):
    for m in range (0,StepsPerRevolution):
      for i in range (0, 8):
        SendPulse(i)
        time.sleep(StepDelay)
  return

#
# Send pulse to the motor
#
def SendPulse(k):
  GPIO.output(IN1, HalfStep[k][3])
  GPIO.output(IN2, HalfStep[k][2])
  GPIO.output(IN3, HalfStep[k][1])
  GPIO.output(IN4, HalfStep[k][0])
  return


#
# Start of the main program loop. Rotate the motor as:
# 2 revs clockwise, stop 2 seconds, 2 revs anticlockwise
#
while True:
  CLOCKWISE(2)
  time.sleep(2)
  ANTICLOCKWISE(2)
  while True:
    pass
```

Figure 3.13 Program listing (half-step mode)

3.3 PROJECT 2 – Thermometer With Dial

In this project we will be designing a thermometer using a temperature sensor IC and then show the temperature on a labelled card board with a moving dial hand wire controlled from a stepper motor. The temperature range is set to 0ºC to 40ºC. The temperature is updated every minute.

3.3.1 Block Diagram

Figure 3.14 shows the block diagram of the project. The temperature is displayed on a card board with a dial hand which is controlled by the stepper motor used in the previous project. The card board is fixed and the temperature values are marked on it. The dial hand is moved by the stepper motor and it points to the current measured ambient temperature. Figure 3.15 shows the card board display with the markings on it.

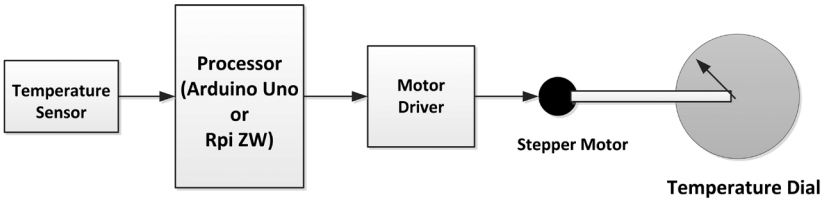

Figure 3.14 Block diagram of the project

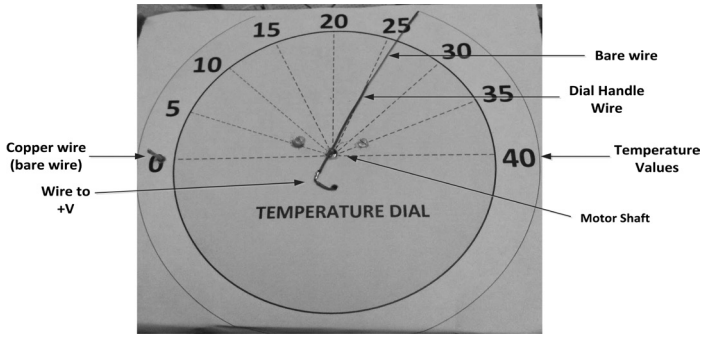

Figure 3.15 Card board display

Temperature scale is marked on a circular card board from 0ºC to 40ºC in steps of 5ºC. A dial hand, made of a wire is attached to the stepper motor shaft at the center of the board. A copper wire is mounted at 0ºC and this point is used as the starting point. The copper wire acts like a switch together with the dial hand wire. When power is applied to the processor the dial hand is moved left (stepper motor rotates anticlockwise) until it comes in contact with the copper wire (part of the dial handle wire is bare and has no sleeve) and it then stops at the starting point of 0ºC. From this point on, the dial hand moves right to the correct point on the scale depending upon the measured temperature value. The temperature is checked every minute and the dial hand moves left or right by the correct amount to show the new temperature if it has changed from the last reading.

There are 180 degrees of angle from 0ºC to 40ºC on the scale. Therefore, each degree of temperature corresponds to 4.5 degrees of angle. For example, if the temperature is 10ºC,

the angle is 45º, if the temperature is 20ºC the angle is 90º, if the temperature is 40ºC the angle is 180º and so on. Therefore, the required angle for a given temperature reading is given by the following formula:

Angle = 4.5 x T

Where, T is the measured temperature in degrees Centigrade, and Angle is the angle that the dial hand must move to show the correct temperature. If **D** is the angle in degrees that the external motor shaft moves for each step applied to the motor, the number of steps required (**S**) to move the dial hand to the correct temperature point on the scale is given by:

S = 4.5 x T / D

The above formula can be used to move the dial hand to the correct point on the scale at the beginning of the program. Notice that this is clockwise rotation of the stepper motor.
If the temperature is T1 degrees Centigrade and after the next reading it is T2 degrees Centigrade, then the number of steps required to move the dial hand is given by the following algorithm:

```
IF T1 = T2 THEN no change in dial position
ELSE IF T1 > T2 THEN
  Difference = T1 – T2
  Move motor clockwise by sending (4.5 x Difference) / D steps
ELSE
  Difference = T2 – T1
  Move motor anticlockwise by sending (4.5 x Difference) / D steps
ENDIF
```

3.3.2 Circuit Diagram (Arduino Uno)

Figure 3.16 shows the circuit diagram of the Arduino Uno based project. The stepper motor is connected to the ULN2003 motor driver module as in the previous project. A TMP36DZ type analog temperature sensor IC is used to measure the ambient temperature. The output of this sensor is connected to analog input A0 of the Arduino Uno. TMP36DZ outputs the temperature as an analog voltage where:

T = (Vo – 500) / 10

Where T is the measured temperature in degrees Centigrade, and Vo is the output voltage of the sensor in millivolts.

The dial hand is connected to +5V. The copper wire placed at 0ºC on the dial is connected to ground through a 10K resistor. The connection point of the copper wire with the resistor is connected to port pin 4 of the Arduino Uno. Normally this point is at logic 0 and goes to logic 1 when the dial hand touches it. When power is applied to the processor the dial hand is moved left by the stepper motor until it touches the copper wire and this point of the stepper motor is taken as the reference 0ºC point.

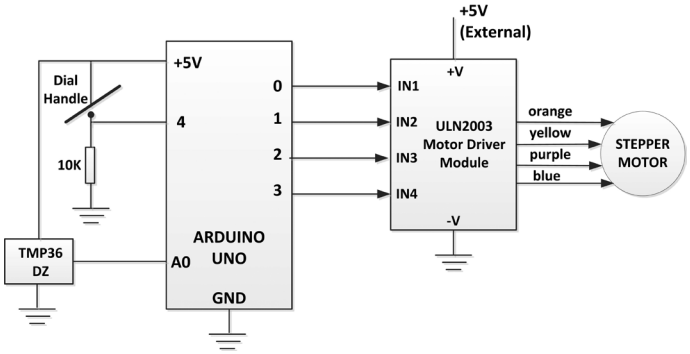

Figure 3.16 Circuit diagram of the project (Arduino Uno)

Power to the motor was supplied from an external +5V power supply through the motor driver module and the circuit was built on a breadboard.

3.3.3 Circuit Diagram (Raspberry Pi)

Figure 3.17 shows the circuit diagram of the Raspberry Pi project. In this version of the project a DHT11 type temperature/humidity sensor chip was used to read the ambient temperature since the RPi ZW does not have any ADC converters on board. The output of the DHT11 was connected to port GPIO22 (pin 15) of the RPi ZW. The stepper motor was connected to the ULN2003 motor driver module as in the previous project.

The connection point of the copper wire with the resistor was connected to port GPIO27 (pin 13) of the RPi ZW.

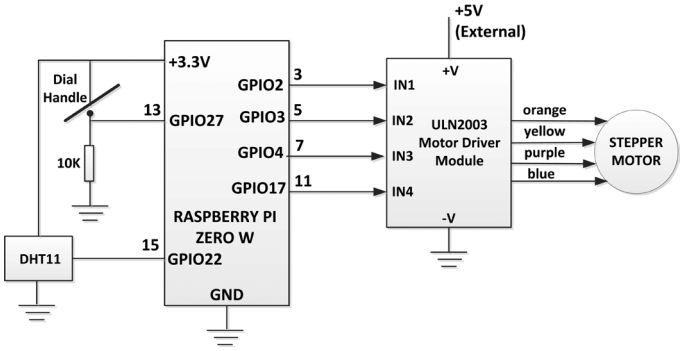

Figure 3.17 Circuit diagram of the project (RPi ZW)

3.3.4 Program Listing (Arduino Uno)

The program listing is shown in Figure 3.18 (program: stepper4). This program uses the Arduino stepper library for simplicity. At the beginning of the program the Arduino library header is included in the program, copper wire connection (**CopperWire**) is assigned to port pin 4, TMP36DZ sensor output (**TMP36**) is assigned to analog port A0, and the stepper motor connection, steps per revolution, and degrees per step are defined.

Inside the setup routine copper wire is configured as an input and the motor speed is set to 12 RPM. The stepper motor is then activated to turn anticlockwise and move the dial handle until the handle touches the copper wire. At this point a logic 1 signal is given to pin 4 of the Arduino Uno and the motor stops. This way we establish the reference point as the point on the dial with the mark 0ºC. The program then waits for a while (5 seconds) before starting to measure and show the temperature.

Inside the main program loop the temperature reading is read from TMP36DZ sensor chip and is converted into degrees Centigrade. This reading is stored in variable **mV**. Then the required number of steps to move the dial handle is calculated as discussed earlier and the motor turns clockwise to move the dial handle to show the temperature reading.

Temperature readings are taken every minute. After moving the dial the first time to show the temperature, next readings are compared with the previous readings and the dial handle is adjusted accordingly by rotating the stepper motor clockwise (present temperature greater than the previous one) or anticlockwise (present temperature less than the previous one) as described earlier. This process continues until stopped manually by the user.

```
/*****************************************************************
*                     TEMPERATURE DIAL
*                     ================
*
* In this project the ambient temperature is read using an analog
* TMP36DZ type sensor chip and it is then displayed on a dial type
* temperature scale. A circular temperature scale was made with the
* temperature values marked from 0C to 40C. A dial is moved by the
* stepper motor to point to the measured temperature.
*
*
* File  : stepper4
* Date  : September 2017
* Author: Dogan Ibrahim
*****************************************************************/
#include <Stepper.h>
#define StepsPerRevolution 2048
const float DegreesPerStep = 360.0 / StepsPerRevolution;
#define CopperWire 4                                  // Reference point
input
#define TMP36 0                                       // TMP36 at A0
Stepper MyMotor(StepsPerRevolution, 0, 2, 1, 3);      // Pulse sequence
int Temp;
float mV, Told, Tnew, Diff, RequiredSteps;
int Steps;
int First = 1;

//
```

```
// Configure CopperWire as input, set motor speed to 12 RPM
// and move the dial hand to the reference point (0C)
//
void setup()
{
   pinMode(CopperWire, INPUT);                          // Reference point
   MyMotor.setSpeed(12);                                // Set motor speed
   //
   // Move the dial to the reference point (0C)
   //
   while(digitalRead(CopperWire) == 0)                  // If not touching
Copper Wire
   {
     MyMotor.step(-1);
   }
   delay(5000);                                         // Wait a while
}

//
// Main program loop. Read the temperature every minute and move the
// dial to point to the correct temperature
//
void loop()
{
  Temp = analogRead(TMP36);                             // Read temperature
  mV = Temp * 5000.0 / 1024.0;                          // in millivolts
  mV = (mV - 500.0) / 10.0;                             // Temperature
  if(First == 1)                                        // If first time
  {
     First = 0;
     Told = mV;
     RequiredSteps = 4.5 * mV / DegreesPerStep;
     Steps = (int)RequiredSteps;
     MyMotor.step(Steps);
  }
  else                                                  // Not first time
  {
     Tnew = mV;
     if(Tnew > Told)
     {
        Diff = Tnew - Told;
        RequiredSteps = 4.5 * Diff / DegreesPerStep;
        Steps = (int)RequiredSteps;
        MyMotor.step(Steps);
     }
     else
```

```
      {
         Diff = Told - Tnew;
         RequiredSteps = 4.5 * Diff / DegreesPerStep;
         Steps = (int)RequiredSteps;
         MyMotor.step(-Steps);
      }
      Told = Tnew;
   }
   delay(60000);                                    // Wait 1 minute
}
```

Figure 3.18 Program listing (Arduino Uno)

3.3.5 Program Listing (Raspberry Pi Zero W)

DHT11 is a temperature/humidity sensor. A library is available from Adafruit for using the DHT11 type sensor in Python programming and this library was used in the RPi ZW part of the project. The DHT11 library must be installed before it can be used. The steps to install this library are given below. Go to RPi ZW command mode and enter the following commands:

- Download the library from Github:

  git clone https://github.com/adafruit/Adafruit_Python_DHT.git

- Change working directory to Adafruit_python_DHT

  cd Adafruit_python_DHT

- Enter the following commands:

  sudo apt-get update
  sudo apt-get install build-essential python-dev python-openssl

- Install the library:

  sudo python setup.py install

If you see an error that a package is already installed or that the package is at the latest version, you can ignore such messages and just continue.

The program listing is shown in Figure 3.19 (program: stepper2.py). At the beginning of the program, libraries RPi.GPIO, time, and Adafruit_DHT are imported to the program and DHT11 is assigned to port GPIO22. The program then specifies that we will be using the DHT11, and various stepper motor parameters are defined here. Copper wire is assigned to port GPIO27, motor driver control pins are defined, and these pins are configured as outputs. Copper wire connection is configured as an input. Then the half-step sequence is

defined in array **HalfStep**. Functions **CLOCKWISE** and **ANTICLOCKWISE** send pulses to the motor to turn it in clockwise and anticlockwise directions respectively. Notice that here we are sending the desired number of step pulses to the motor, and are not asking the motor to rotate a specified number of times as was the case with program **stepper1.py**. The number of steps is specified in the arguments of these two functions. For example, calling function CLOCKWISE(100) rotates the motor by 100 steps in clockwise direction, calling with CLOCKWISE(4096) rotates the motor one complete revolution in the clockwise direction and so on.

The program rotates the stepper motor anticlockwise until the dial handle touches the copper wire at the reference point (point where the 0ºC is marked). Then the program loop starts. Inside this loop the temperature (and humidity) is read from the DHT11 and the dial handled is moved to the show the correct temperature by sending the correct number of pulses to the stepper motor.

```
#-----------------------------------------------------------------
#                          TEMPERATURE DIAL
#                          ================
#
# In this project the ambient temperature is read using the digital
# DHT11 temperature/humidity sensor chip an dis then displayed on
# a dial type temperature scale. A circular scale was made with
# temperature values marked on it from 0C to 40C. The dial is moved
# by the stepper motor to point to the measured temperature.
#
#
# File  : stepper2.py
# Date  : September 2017
# Author: Dogan Ibrahim
#-----------------------------------------------------------------------
import RPi.GPIO as GPIO
import time
import Adafruit_DHT
sensor = Adafruit_DHT.DHT11
DHT11 = 22       # GPIO22
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
RPM = 12
StepsPerRevolution = 4096.0
DegreesPerStep = 360.0 / StepsPerRevolution
StepDelay = 0.014648 / RPM
CopperWire = 27    # GPIO27
First = 1
global cw, acw
cw = 7
acw = 0
```

```
#
# Motor driver control pins
#
IN1 = 2      # GPIO2
IN2 = 3      # GPIO3
IN3 = 4      # GPIO4
IN4 = 17       # GPIO17

#
# Configure driver control pins as outputs
#
GPIO.setup(IN1, GPIO.OUT)
GPIO.setup(IN2, GPIO.OUT)
GPIO.setup(IN3, GPIO.OUT)
GPIO.setup(IN4, GPIO.OUT)
GPIO.setup(CopperWire, GPIO.IN)

#
# Define Half-Step sequence
#
HalfStep = [[1,0,0,0],
            [1,1,0,0],
            [0,1,0,0],
            [0,1,1,0],
            [0,0,1,0],
            [0,0,1,1],
            [0,0,0,1],
            [1,0,0,1]]

#
# This function rotates the motor clockwise specified number of
# steps
#
def CLOCKWISE(StepCount):
  global cw
  j = 0
  while j < StepCount:
    SendPulse(cw)
    time.sleep(StepDelay)
    cw = cw - 1
    if cw == -1:
      cw = 7
    j = j + 1
  return
```

```
#
# This function rotates the motor anticlockwise specified number
# of steps
#
def ANTICLOCKWISE(StepCount):
  global acw
  j = 0
  while j < StepCount:
    SendPulse(acw)
    time.sleep(StepDelay)
    acw = acw + 1
    if acw == 8:
      acw = 0
    j = j + 1
  return

#
# Send pulse to the motor
#
def SendPulse(k):
  GPIO.output(IN1, HalfStep[k][3])
  GPIO.output(IN2, HalfStep[k][2])
  GPIO.output(IN3, HalfStep[k][1])
  GPIO.output(IN4, HalfStep[k][0])
  Return

#
# Move the dial hand to the reference point (0 degrees C) and wait
# for 5 seconds before starting
#
while(GPIO.input(CopperWire) == 0):
  ANTICLOCKWISE(1)

time.sleep(5)

#
# Start of the main program loop. Read the temperature every minute
# and move the dial hand to point to the correct temperature. Notice
# that DHT11 returns the humidity data as well but it is not used here
#
while True:
  humidity,temperature=Adafruit_DHT.read_retry(sensor, DHT11)
  if First == 1:
    First = 0
    Told = temperature
    RequiredSteps = 4.5 * temperature / DegreesPerStep
```

```
    CLOCKWISE(RequiredSteps)
  else:
    Tnew = temperature
    if Tnew > Told:
      Diff = Tnew - Told
      RequiredSteps = 4.5 * Diff / DegreesPerStep
      CLOCKWISE(RequiredSteps)
    else:
      Diff = Told - Tnew
      RequiredSteps = 4.5 * Diff / DegreesPerStep
      ANTICLOCKWISE(RequiredSteps)
  Told = temperature
  time.sleep(60)
```

Figure 3.19 Program listing (RPi ZW)

You can easily run the program in Python interactive mode.

3.4 Summary

In this section we have seen how to control a stepping motor from Arduino Uno and also from Raspberry Pi Zero W.

In the next Chapter we shall be looking at how to control a servo motor using an Arduino Uno and a Raspberry Pi Zero W.

CHAPTER 4 • SIMPLE SERVO MOTOR PROJECTS

4.1 Overview

Servo motors are DC motors, used in many position control applications where it is required to position the motor shaft at a desired position. These motors use built-in position sensing devices to determine the position of its shaft, and then rotate in the correct direction to position the motor shaft at the commanded position. Servo motors do not rotate freely round and round like a normal motor, but rather they can turn around 180 degrees or so back and forth.

Unlike other motors, servo motors have 3 wire connections: power, ground, and control signal. The control signal is positive going Pulse Width Modulated (PWM) voltage such that the frequency of this voltage (or the period) is fixed, but the duration varies in relation to the required position of the motor shaft. For example, the SG90 servo motor operates such that the period of the PWM signal is 20 ms (frequency of 50 Hz) so that the motor checks the duration of the pulse every 20 ms. The motor shaft is in the middle when the pulse duration is 1.5 ms, 90 degrees all the way to the right when the pulse duration is 2 ms, and -90 degrees all the way to the left when the pulse duration is 1 ms.
In this Chapter we shall be learning how to control a small servo motor using an Arduino Uno and a Raspberry Pi Zero W development boards. The two projects in this Chapter are simple and aimed to teach the basic control of a servo motor.

4.2 PROJECT 1 – Basic Servo Motor Control (Arduino Uno)

In this project we shall control a small servo motor as follows:

- Position the motor shaft in the middle (90 degrees)
- Wait 1 second
- Position the motor shaft all the way to the right (0 degrees)
- Wait 1 second
- Position the motor shaft all the way to the left (180 degrees)
- Wait 1 seconds
- Move the motor shaft all the way right to left and then left to right in increments of 1 degree
- Stop

The servo motor used in this project is the TowerPro SG90 servo motor shown in Figure 4.1 together with its wiring diagram.

Figure 4.1 SG90 servo motor

4.2.1 Block Diagram

The block diagram of the motor is as in Figure 4.2. The servo motor is simply connected to the processor. A handle supplied with the motor should be mounted on the motor shaft so that the rotation of the shaft can easily be seen.

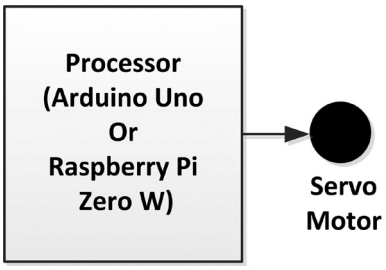

Figure 4.2 Block diagram of the project

4.2.2 Circuit Diagram

SG90 servo motor has the following specifications:

- Operating voltage: 3.3V to 6V
- Running current (at +5V): 220 ±50 mA
- Stall current (at +5V): 650 ±80 mA
- Idle current (at +5V): 6±10 mA
- Weight: 9 g
- Stall torque: 1.8 kgf.cm
- Speed: 0.1 s/60 degree

Figure 4.3 shows the circuit diagram of the project. Signal input of the motor (brown wire) is connected to port pin 3 of the Arduino Uno. The power supply input (red wire) is connected to +5V of the Arduino Uno. Notice that the +5V pin of the Arduino Uno has enough current capability to drive a small motor. Larger motors must be connected to an external power supply not to overload the Arduino Uno board.

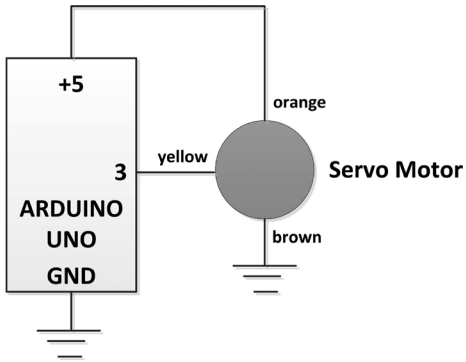

Figure 4.3 Circuit diagram of the project

4.2.3 Program Listing

In this project the Arduino servo library is used. This library has the following functions:

attach: attach the servo variable to a pin. Additionally, the pulse width corresponding to the minimum (0 degree) and the maximum (180 degree) angle can also be specified.

write: write an angle value to the servo. Valid values are 0 to 180.

writeMicroseconds: write a value in microseconds to the servo to control the motor shaft. Normally, 1500 is the middle point, 2000 is fully clockwise, and 1000 is fully anticlockwise.
read: read the current angle of the motor (value passed in the last write)

attached: check whether or not servo is attached to a pin

detach: detach the servo

Figure 4.4 shows the program listing of the project (program: servo1). At the beginning of the program the servo library header is included in the program and the servo pin is specified (3 in this project). Inside the setup routine servo pin is attached. The main program loop turns the servo shaft as specified in the project. At the end, the motor is stopped by detaching it.

Notice that as shown in Figure 4.5, 90 degrees is the middle point, 0 degrees is 90 degrees (all the way) clockwise, and 180 degrees is all the way anticlockwise.

```
/*****************************************************************
*                  BASIC SERVO MOTOR CONTROL
*                  =========================
*
* In this project a small servo motor (type TowerPro SG90) is
* connected to pin 3 of the Arduino Uno. The aim of this project
* is to show how the servo motor can be controlled. The motor is
* controlled as follows:
```

```
*
*   Position the motor shaft to the middle point
*   Wait 1 seconds
*   Position the motor shaft 90 degrees to the right
*   Wait 1 seconds
*   Position the motor shaft 90 degrees to the left
*   Wait 1 seconds
*   Move the motor shaft shaft left to right in increments of 1 degree
*   Stop the motor
*
* File  : servo1
* Date  : September 2017
* Author: Dogan Ibrahim
*****************************************************************/
#include <Servo.h>
Servo MyServo;

//
// Stepper Motor pin definition
//
#define ServoPin 3                              // Servo motor signal

//
// Attach the servo to ServoPin
//
void setup()
{
  MyServo.attach(ServoPin);
}


//
// Main program loop. Control the servo motor as required
//
void loop()
{
  MyServo.write(90);                            // Servo shaft in the middle
  delay(1000);                                  // 1 sec delay
  MyServo.write(0);                             // 90 degrees clockwise
  delay(1000);                                  // 1 sec delay
  MyServo.write(180);                           // 90 degrees anticlockwise
  delay(1000);                                  // 1 sec delay
  //
  // Move servo arm left to right (clockwise)
  //
  for(int Angle = 180; Angle > 0; Angle--)
```

```
  {
    MyServo.write(Angle);
    delay(100);
  }

  MyServo.detach();                                    // Detach the servo
  while(1);                                            // Stop

}
```

Figure 4.4 Program listing

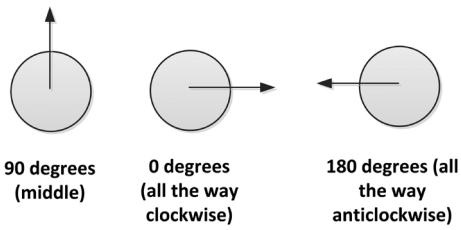

Figure 4.5 Motor shaft positions

Notice that the servo motors may need calibration. For example, the shaft of motor that the author used was not in the middle when it was set to 90 degrees. It was found by experiment that the shaft moved to the middle point when it was set to 105 degrees.
Figure 4.6 shows the servo motor connected to the Arduino Uno.

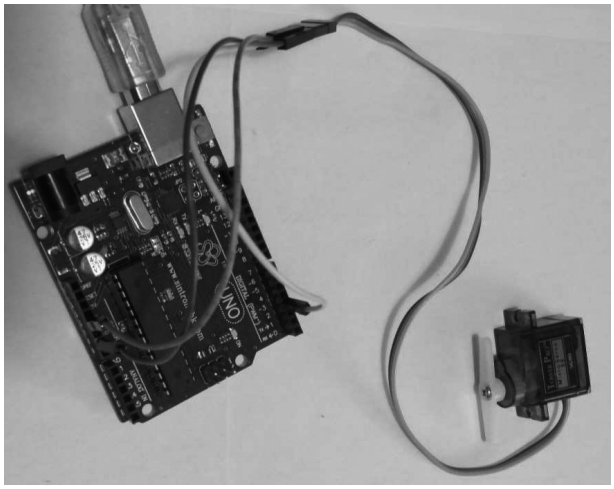

Figure 4.6 Connecting the servo motor to the Arduino Uno

4.3 PROJECT 2 – Controlling the Servo Motor with a Potentiometer (Arduino Uno)
In this project the servo motor shaft is controlled with a potentiometer such that rotating the potentiometer arm rotates the motor shaft accordingly. The same servo motor as in the previous project is used here.

4.3.1 Block Diagram

The block diagram of the motor is shown in Figure 4.7 The servo motor is simply connected to the processor and a potentiometer is used as the position controller.

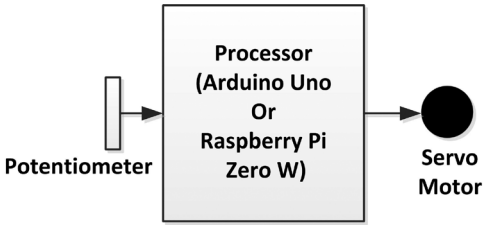

Figure 4.7 Block diagram of the project

4.3.2 Circuit Diagram

Figure 4.8 shows the circuit diagram of the project. The motor signal pin is connected to port pin 3 of the Arduino Uno as before. The arm of the potentiometer is connected to analog port A0.

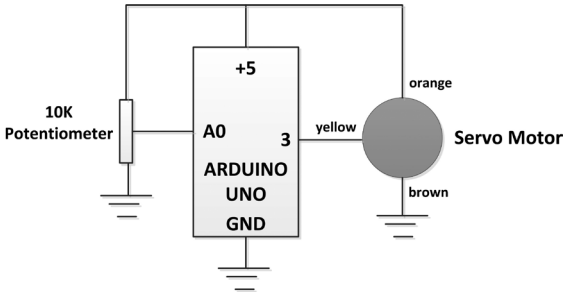

Figure 4.8 Circuit diagram of the project

4.3.3 Program Listing

The program listing is shown in Figure 4.9 (program: servo2). At the beginning of the program the servo library header is included, servo pin and the potentiometer arm pins are defined. Inside the main program loop value of the potentiometer arm is read from analog port A0 and stored in variable **PotValue**. Since we have a 10-bit ADC, the value read is between 0 and 1023. This value is mapped to the servo angles of 0 to 180 degrees using the Arduino map statement which maps a number from one range to another range. The mapped value is then sent to the servo motor using the servo **write** statement. As the potentiometer arm is rotated so does the servo shaft.

```
/*****************************************************************
*                 SERVO CONTROL WITH A POTENTIOMETER
*                 ==================================
*
* In this project a small servo motor (type TowerPro SG90) is
* connected to pin 3 of the Arduino Uno. In addition, a 10K
* potentiometer is connected to analog input A0. The servo motor
```

```
* shaft is controlled by manually rotating the potentiometer arm.
*
* File  : servo2
* Date  : September 2017
* Author: Dogan Ibrahim
*****************************************************************/
#include <Servo.h>
Servo MyServo;

//
// Stepper Motor and potentiometer pin definitions
//
#define ServoPin 3                                    // Servo motor signal
#define PotPin 0                                      // Potentiomter arm
int PotValue;

//
// Attach the servo to ServoPin
//
void setup()
{
  MyServo.attach(ServoPin);
}


//
// Main program loop. Control the servo shaft by the potentiometer
//
void loop()
{
  PotValue = analogRead(PotPin);                  // Read potentiometer
value
  PotValue = map(PotValue, 0, 1023, 0, 180);      // Map between 0 and 180
degrees
  MyServo.write(PotValue);                        // Send to servo
  delay(20);                                      // Wait until servo moves
}
```

Figure 4.9 Program listing

Figure 4.10 shows the circuit built on a breadboard.

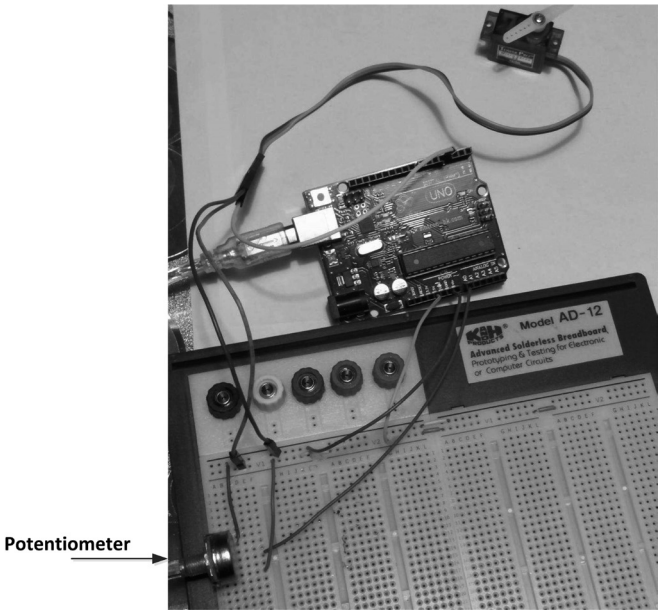

Figure 4.10 Circuit built on a breadboard

4.4 PROJECT 3 – Basic Servo Motor Control (Raspberry Pi Zero W)

In this project we shall see how a servo motor can be controlled from an RPi ZW. In this project the servo motor rotates as in Project 1:

- Position the motor shaft in the middle (90 degrees)
- Wait 2 seconds
- Position the motor shaft all the way to the right (0 degrees)
- Wait 2 seconds
- Position the motor shaft all the way to the left (180 degrees)
- Wait 2 seconds
- Move the motor shaft all the way from left to right
- Stop

The same servo motor as in Project 1 is used here.

4.4.1 Block Diagram

The block diagram of the motor is as shown in Figure 4.2.

4.4.2 Circuit Diagram

Figure 4.11 shows the circuit diagram of the project. Port pin GPIO 2 (pin 3) of the RPi ZW is connected to the signal input of the motor. Power to the motor is supplied from an external +5V power supply with the ground of this power supply and the RPi ZW ground pins are connected together. You should be aware that the RPi ZW power supply has limited capacity and running the motor from the RPi ZW power supply may damage your RPi ZW.

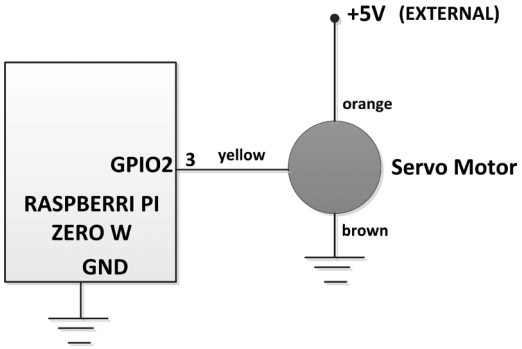

Figure 4.11 Circuit diagram of the project

4.2.3 Program Listing

The servo motor is controlled by sending PWM voltage waveforms to it with a period of 20 ms (frequency of 50 Hz). The duration of the PWM waveform determines the shaft position as follows:

- Shaft in the middle if the duration is 1.5ms
- Shaft is all the way to the right if the duration is 2 ms
- Shaft is all the way to the left if the duration is 1 ms

The Duty Cycle of a PWM waveform is expressed as a percentage and it is the ratio of the ON time to its period, multiplied by 100.

Considering that the period is 20 ms, the Duty Cycle for the 3 motor turns can be calculated as follows:

- Middle position = 100 x (1.5 / 20) = 7.5%
- Fully to the right = 100 x (2.0 / 20) = 10%
- Fully to the left = 100 x (1.0 / 20) = 5%

In Python programming we can use any pin to generate a PWM waveform. The required statements are:

```
P = GPIO.PWM(pin, freq)              // Define pin to be used for PWM
p.start(Initial Duty Cycle)          // Start PWM
p.ChangeDutyCycle(New Duty Cycle)    // Change Duty Cycle
p.stop()                             // Stop PWM
```

where **freq** is the frequency of the PWM waveform and **Initial Duty Cycle** is the starting Duty Cycle of the waveform.

Figure 4.12 shows the program listing (program: servo1.py). At the beginning of the program the servo pin is defined as port GPIO2 and the PWM channel is configured to operate at 50 Hz. Notice that all motors, even the same models from the same manufacturer, do

not respond in exactly the same way and it is usually necessary to calibrate a servo motor to get the best results. This can easily be done with experimentation by changing the Duty Cycle and observing the motor shaft positions. It was found by experiments that different duty cycles than the theoretical ones were required to move the motor shaft. The findings were: 5% (1.0 ms) moved the motor shaft to the middle position, 10% (2.0 ms) moved the motor all the way to the left, and 2.2% (0.44 ms) moved the motor shaft all the way to the right. It is therefore best to experiment until you get the correct duty cycle settings.

The motor shaft is put into middle position by setting the Duty Cycle to 5%, after 2 seconds the Duty Cycle is changed to 2.2% so that the motor rotates all the way to right. After 2 seconds the Duty Cycle is set to 10% so that the motor rotates all the way to left. Finally, the motor is rotated from left to right with a small delay between the steps.

```
#-----------------------------------------------------------------
#                       BASIC CONTROL OF A SERVO MOTOR
#                       ==============================
#
# This is a simple servo motor controller. In this project a
# small servo motor (TowerPro SG90) is connected to the GPIO2 of
# the RPi ZW processor. The motor is controlled in half-step mode as
follows:
#
# Move the shaft to middle position
# Wait 2 second
# Turn clockwise fully
# Wait 2 second
# Turn to anticlockwise fully
# Wait 2 second
# Move the shaft left to right (clockwise
#
# File  : servo1.py
# Date  : September 2017
# Author: Dogan Ibrahim
#-----------------------------------------------------------------------
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

ServoPin = 2     # GPIO2

#
# Configure Servo pin as output
#
GPIO.setup(ServoPin, GPIO.OUT)
```

```
#
# Configure the PWM on pin Servo
#
p = GPIO.PWM(ServoPin, 50)

#
# Start rotating the motor shaft as required
#
p.start(5)     # Middle position
time.sleep(2)      # Wait 2 second
p.ChangeDutyCycle(2.2)  # All the way to the right
time.sleep(2)      # Wait 2 second
p.ChangeDutyCycle(10)   # All the way to the left
time.sleep(2)      # Wait 2 second
#
# Move the motor from left to right with 1 degree steps
#
i = 10.0
while i > 2.1:
  p.ChangeDutyCycle(i)
  time.sleep(0.1)
  i = i - 0.1
p.stop()
#
# Wait here forever
#
while True:
  pass
```

Figure 4.12 Program listing

Figure 4.13 shows the Raspberry Pi Zero W connected to the servo motor.

You can run the program interactively from the command mode by entering the command:

```
pi@raspberrypi:~ $ sudo python servo1.py
```

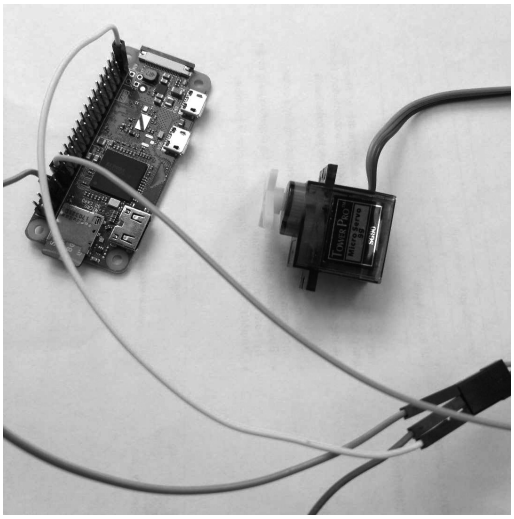

Figure 4.13 Connecting the servo motor to the RPi ZW

4.5 PROJECT 4 – Moving the Motor Shaft to a Given Angle (Raspberry Pi Zero W)
In many applications we may want to move the servo motor shaft to a desired angle position and we may need to know what the Duty Cycle should be to achieve this. In this project we will derive an equation that can be used to move the motor shaft to any angle between 0 and 180 degrees. As an experiment, we will move the motor shaft to 45 degrees (half way to the right). The same servo motor as in the previous projects is used here.

The block diagram and the circuit diagram of this project are as in Figure 4.2 and Figure 4.11.

4.5.1 Program Listing
Figure 4.14 shows the 3 main motor positions with the required Duty Cycle to move the motor shaft to these positions. The graph at the bottom of the figure shows the variation of the motor positions (shaft angle) with the Duty Cycles. Although this is not a straight-line graph, we can approximate it to a straight line and find the equation of this line as:

$$Duty\ Cycle = \frac{7.8 \times Angle}{180} + 2.2$$

Thus, for example, if we want to move the motor to an angle of 45 degrees, the Duty Cycle should be:

$$Duty\ Cycle = \frac{7.8 \times Angle}{180} + 2.2$$

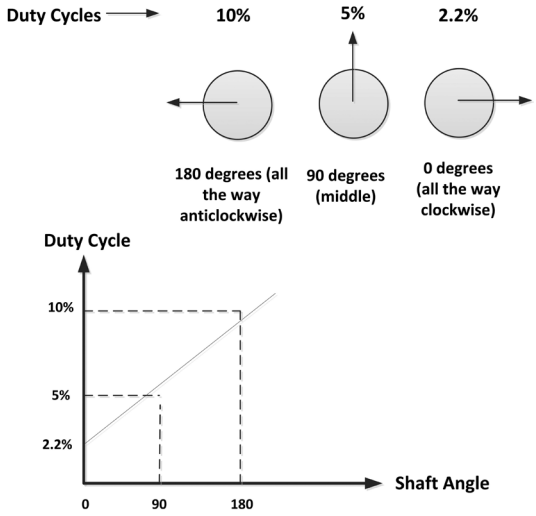

Figure 4.14 Motor positions and Duty Cycles

Figure 4.15 shows the program listing to move the motor shaft to 45 degrees (program: servo2.py). Notice that it will be required to calibrate your servo motor and draw a new graph similar to the one in Figure 4.14 in order to get accurate results.

```
#-------------------------------------------------------------------
#                       MOVE SERVO MOTOR to 45 DEGREES
#                       ==============================
#
# In this project the servo motor shaft is moved to the angle of
# 45 degrees by setting the correct PWM Duty Cycle.
#
# File  : servo2.py
# Date  : September 2017
# Author: Dogan Ibrahim
#-------------------------------------------------------------------
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
Angle = 45
DutyCycle = 2.2 + Angle * 7.8 / 180.0

ServoPin = 2     # GPIO2

#
# Configure Servo pin as output
#
GPIO.setup(2, GPIO.OUT)
```

```
#
# Configure the PWM on pin Servo and set the PWM to 50 Hz
#
p = GPIO.PWM(ServoPin, 50)

#
# Start rotating the motor shaft as required
#
p.start(DutyCycle)    # Move to required angle

#
# Wait here forever
#
while True:
  pass
```

Figure 4.15 Program listing

You can run the program interactively from the command mode by entering the command:

pi@raspberrypi:~ $ **sudo python servo2.py**

4.6 Summary

In this Chapter we have seen how to control a servo motor.

In the next Chapter we shall be looking at how to control our robot from an Android mobile phone using Bluetooth communications with an Arduino Uno.

CHAPTER 5 • ARDUINO BLUETOOTH ROBOT CONTROL

5.1 Overview

Bluetooth is a wireless technology operating from 2.4 to 2.485 GHz and is used to exchange data over short distances. It is available on all smart phones and on most modern laptops and desktop computers.

In this Chapter we shall see how to control our robot remotely using Bluetooth on our mobile phone to give commands and Arduino Uno as the robot controller. The robot used in this Chapter is the one used in Chapter 3. By giving commands on our mobile phone we shall see how to move the robot forwards, backwards, turning left or right, and so on. In this project an Android type mobile phone is used to send commands to the robot.

5.2 PROJECT 1 – Bluetooth Based Remote Robot Control

In this project we shall be using the robot we used in Chapter 3. Additionally, we will mount a front LED, a back LED and a buzzer on our robot chassis. We shall be controlling the robot movements, the LEDs and the buzzer separately from our mobile phone using Bluetooth by giving simple commands.

The commands given in Table 5.1 will be used to control the LEDs and the buzzer. The commands to control the robot movements are given in Table 5.2. In this table n is the duration in seconds. For example, the command F2 means **move robot forwards for 2 seconds and then stop**.

Controlled part	ON command	OFF command
Front LED (head LED)	h1	h0
Rear LED (tail LED)	t1	t0
Buzzer	b1	b0

Table 5.1 Commands to control the LEDs and the buzzer

Move robot forwards	Fnn
Move robot backwards	Bnn
Turn robot left	Lnn
Turn robot right	Rnn

Table 5.2 Robot movement commands (nn must be 2 digit integer numbers)

5.2.1 Block Diagram

Figure 5.1 shows the block diagram of the project. Arduino Uno has no on-board Bluetooth capability. For this reason a Bluetooth module is connected to the Arduino Uno so that communication with a mobile phone can be achieved.

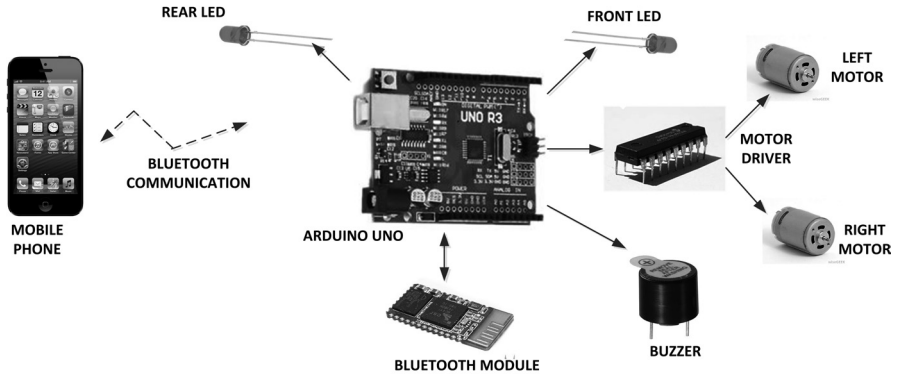

Figure 5.1 Block diagram of the project

5.2.2 Circuit Diagram

In this project an HC-06 type Bluetooth module is connected to the Arduino Uno so that it can communicate over the Bluetooth link. Figure 5.2 shows the HC-06 module. This is a small module that communicates with the host processor over its serial link. The module has 4 pins: RXD, TXD, GND and VCC. As shown in the circuit diagram in Figure 5.3, VCC and GND are connected to the +5V power pin and GND pin of the Arduino Uno respectively. Notice that although HC-06 is a +3.3V module, its power supply can be +3.3V to +6V. The input pins of the HC-06 are not +5V compatible and they must not be connected to any of the Arduino output pins. Pin TXD of the HC-06 is serial output pin and it must be connected to the RXD pin of the Arduino (pin 0). Pin RXD of the HC-06 is the serial input pin which is not +5V tolerant and must not be connected directly to the Arduino TX pin (pin 1). As shown in Figure 5.3, a resistive potential divider circuit is used to lower the output voltage of the Arduino to +3.3V before it is connected to the RXD input of the HC-06.

The front and back (tail) LEDs are connected to Arduino Uno port pins 7 and 6 respectively through 330 Ohm current limiting resistors. The positive input of the active buzzer is connected to pin 8 of the Arduino, and the other pin is connected to GND.

1A, 2A, 3A and 4A pins of the L293 motor controller chip are connected to port pins 2,3,4 and 5 of the Arduino Uno. As in the previous robot control projects, left and right motors of the robot are connected to pins 3,6 and 11,14 of the L293 motor driver chip.

Power to the Arduino is supplied using 4 x Ni-Zn batteries, connected to Vin pin of the Arduino Uno. The two wheel motors also receive their power from these batteries through pin 8 of the L293 chip. Notice that you must not power the motors from +5V pin of the Arduino Uno as it may not be able to supply the required current for the two motors. Always use an external power supply for the motors.

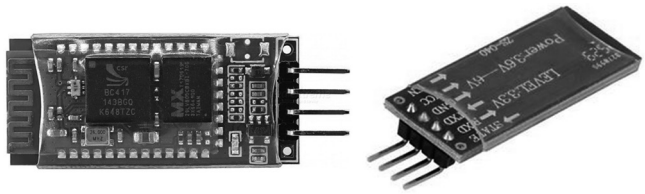

Figure 5.2 HC-06 Bluetooth module

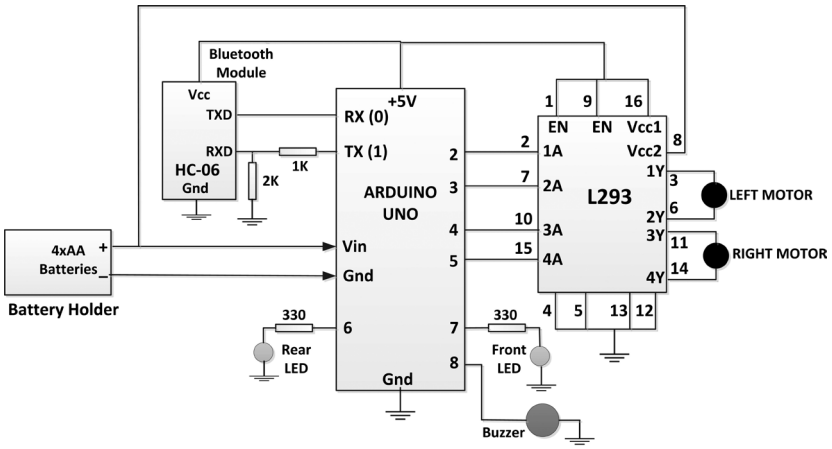

Figure 5.3 Circuit diagram of the project

5.2.3 Android Mobile Phone Apps

In this project we will be sending commands to the robot from an Android based mobile phone. We therefore need an application on our mobile phone where we can send data through Bluetooth. There are many such applications free of charge in **Play Store**. The one chose by the author is called **Arduino Bluetooth Controller** by **Khondokar Production** (see Figure 5.4). You should install this application (or a similar one) on your Android phone.

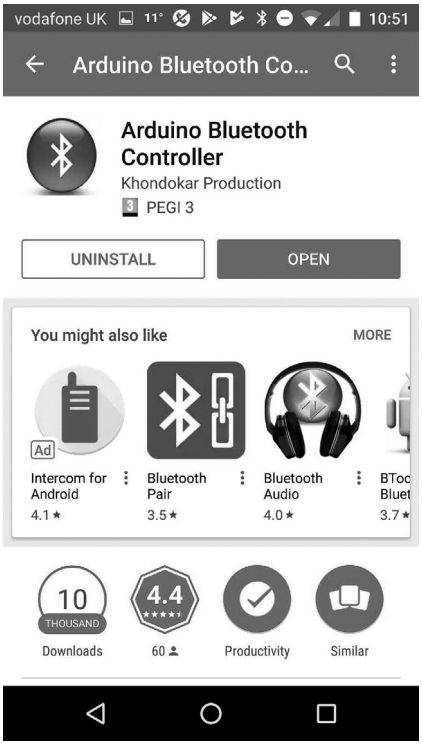

Figure 5.4 Android Bluetooth apps

5.2.4 Pairing the HC-06

Before using the HC-06 first time we have to pair it with our mobile phone. Enable Bluetooth on your mobile phone from the **Settings** menu, apply power to your robot, and wait until it detects the HC-06. Click to pair and enter the passkey as 1234 (or 0000) when asked. You should now be able to communicate with your robot over the Bluetooth connection (see Figure 5.5).

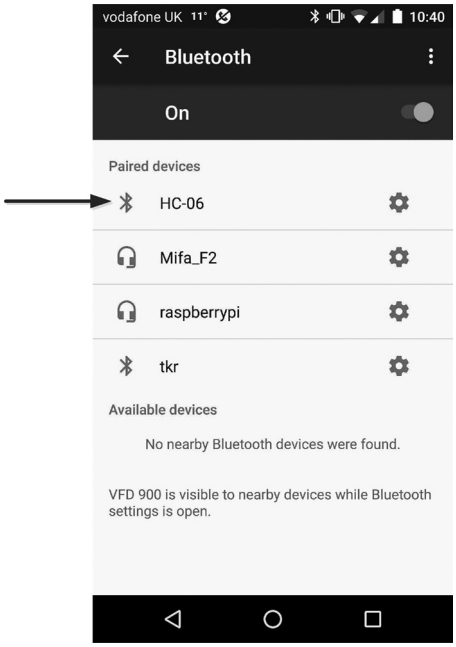

Figure 5.5 HC-06 connected to our mobile phone

5.2.5 Controlling the Robot

You should be able now to control your robot by sending commands from your mobile phone. The steps to send commands to your robot are as follows:

- Make sure that the program is uploaded into your Arduino Uno and that power is applied to the robot
- Enable Bluetooth on your mobile phone and make sure that it is paired with the HC-06 module
- Start the apps **Arduino Bluetooth Controller** on your mobile phone and select HC-06 as shown in Figure 5.6 (second one from the top).

Figure 5.6 Select HC-06 module

- If you are connected to HC-06 successfully you should see a screen where you can enter characters at the top. Enter the following command and click **Send Data** (at the top right hand corner of the screen) to turn ON the front LED (see Figure 5.7). Notice that all commands must be terminated with the # character. Also, the time delay in robot movement commands cannot be greater than 99 seconds:

  h1#

  The front LED should turn ON. To turn it OFF enter command **h0#**
  Similarly, to turn the buzzer on enter **b1#** and to turn it OFF enter **b0#**

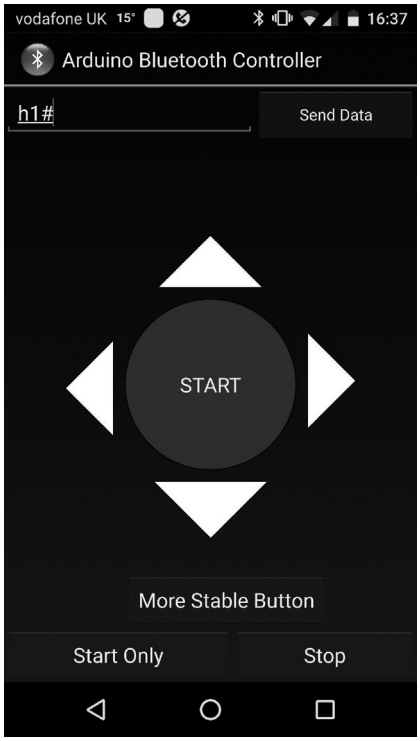

Figure 5.7 Command to turn ON the front LED

- To move the robot forward for 3 seconds enter the command:

  F03#

- To move the robot reverse for 12 seconds enter the command:

  F12#

5.2.6 Program Listing

Figure 5.8 shows the program listing (program: mobile1). At the beginning of the program motor driver connections, LED connections, and the buzzer connection are defined. Inside the setup routine motor pins, LEDs and the buzzer are configured as outputs. Also, the LEDs and the two motors are turned OFF to start with. In addition, the Bluetooth interface speed is configured as 9600 which is the default value. Functions FORWARD, REVERSE, LEFT and RIGHT move the robot in the specified directions. These functions have arguments which specify for how long the action should take place in seconds.

Inside the main program loop the program checks if a command is received from the Bluetooth module. If a command is available the it is read using function **readBytesUntil** which in this program reads until the terminator character **#** is detected or until 3 characters are received from the Bluetooth module.

Commands are stored in array **cmd** and are handled using a **switch** statement. cmd[0] is the first character of the command (e.g. h, t, b, F etc). cmd[1] is the action (0 or 1) for the LED and buzzer commands. cmd[1] and cmd[2] are the delay values for the motor movement commands (F, B, L, R). For example, the front LED command **h** is handled as follows:

```
      case 'h':
              if(cmd[1] == '1')
                digitalWrite(FrontLED, HIGH);
              else if(cmd[1] == '0')
                digitalWrite(FrontLED, LOW);
              break;
```

```
/*****************************************************************
*                 REMOTE ROBOT CONTROL USING BLUETOOTH
*                 ====================================
*
* In this project a mobile robot is connected to the Arduino Uno.
* In addition, a front LED, a back LED, and a buzzer are connected
* to the Arduino Uno.
*
* A Bluetooth module is attached to the Arduino Uno so that it
* communicates with a mobile phone using Bluetooth. The commands
* sent from the mobile phone can turn ON/OFF the LEDs and the buzzer,
* and it can also control the movements of the robot.
*
* The Bluetooth module used is the HC-06 which has the following
* default settings in data mode: 9600 Baud, 8 bits, 1 stop bit, no
* parity, no handshaking
*
* Valid commands are (The ommands must be terminated with the
* "#" character):
*
* h1 Turn ON front LED, h0 Turn OFF front LED
* t1 turn ON tail LED,  t0 Turn OFF tail LED
* b1 turn ON buzzer,    b0 turn OFF buzzer
* Fn Move robot forward n seconds
* Bn Move robot reverse n seconds
* Ln Turn robot left n seconds
* Rn Turn robot right n seconds
*
*
* File  : mobile1
* Date  : September 2017
* Author: Dogan Ibrahim
*****************************************************************/
//
```

```
// Define motor driver connections
//
#define L1A 2                                   // LEFT motor 1A
#define L2A 3                                   // LEFT motor 2A
#define R3A 4                                   // RIGHT motor 3A
#define R4A 5                                   // RIGHT motor 4A

//
// Define LED and buzzer connections
//
#define FrontLED 7                              // Front LED
#define TailLED 6                               // Tail LED
#define Buzzer 8                                // Buzzer

char cmd[10];                                   // Array to store commands
int DelayTime;

//
// Configure 1A, 2A, 3A, 4A, LEDs and Buzzer as outputs. Turn OFF
// LEDs and Buzzer at the begining. Also stop the motors at the
// beginning just in case, and the Baud rae is set to 9600
//
void setup()
{
  pinMode(L1A, OUTPUT);
  pinMode(L2A, OUTPUT);
  pinMode(R3A, OUTPUT);
  pinMode(R4A, OUTPUT);
  pinMode(FrontLED, OUTPUT);
  pinMode(TailLED, OUTPUT);
  pinMode(Buzzer, OUTPUT);
  digitalWrite(FrontLED, LOW);
  digitalWrite(TailLED, LOW);
  digitalWrite(Buzzer, LOW);
  StopMotors();
  Serial.begin(9600);
}

//
// This function stops the motor
//
void StopMotors()
{
  digitalWrite(L1A, LOW);
  digitalWrite(L2A, LOW);
  digitalWrite(R3A, LOW);
```

```
  digitalWrite(R4A, LOW);
}

//
// This function moves the robot forward for tim seconds. Both
// motors rotate in the same direction
//
void FORWARD(char tim)
{
  digitalWrite(L1A, LOW);
  digitalWrite(L2A, HIGH);
  digitalWrite(R3A, LOW);
  digitalWrite(R4A, HIGH);
  delay(tim*1000);
  StopMotors();
}

//
// This function reverses the motor for tim seconds. Both motors
// rotate in the same direction
//
void REVERSE(char tim)
{
  digitalWrite(L1A, HIGH);
  digitalWrite(L2A, LOW);
  digitalWrite(R3A, HIGH);
  digitalWrite(R4A, LOW);
  delay(tim*1000);
  StopMotors();
}

//
// This function turns the motor LEFT for tim seconds. Left motor
// is stopped and the right motor is activated
//
void LEFT(char tim)
{
  digitalWrite(L1A, HIGH);
  digitalWrite(L2A, LOW);
  digitalWrite(R3A, LOW);
  digitalWrite(R4A, HIGH);
  delay(tim*1000);
  StopMotors();
}

// This function turns the motor RIGHT for tim seconds. Right motor
```

```
// is stopped and the left motor is activated
//
void RIGHT(char tim)
{
  digitalWrite(L1A, LOW);
  digitalWrite(L2A, HIGH);
  digitalWrite(R3A, HIGH);
  digitalWrite(R4A, LOW);
  delay(tim*1000);
  StopMotors();
}

//
// This function extracts the required delay in a motor movement
// command. The delay is in seconds
//
int GetDelay()
{
  int dly = 10*(cmd[1]-'0') + cmd[2] - '0';
  return dly;
}

//
// Main program loop. Read the command from the serial Bluetooth
// device, decode teh command and implement the control action
// as required
//
void loop()
{
  cmd[0]=0;                                          // Clear command array
  cmd[1]=0;
  cmd[2]=0;

  if(Serial.available() > 0)                         // Any incoming data?
  {
    Serial.readBytesUntil('#',cmd, 3);               // Read incoming data
    switch(cmd[0])
    {
      case 'h':
              if(cmd[1] == '1')
                digitalWrite(FrontLED, HIGH);
              else if(cmd[1] == '0')
                digitalWrite(FrontLED, LOW);
              break;
      case 't':
              if(cmd[1] == '1')
```

```
                digitalWrite(TailLED, HIGH);
              else if(cmd[1] == '0')
                digitalWrite(TailLED, LOW);
              break;
        case 'b':
              if(cmd[1] == '1')
                digitalWrite(Buzzer, HIGH);
              else if(cmd[1] == '0')
                digitalWrite(Buzzer, 0);
              break;
        case 'F':
               DelayTime = GetDelay();
               FORWARD(DelayTime);
               break;
        case 'B':
               DelayTime = GetDelay();
               REVERSE(DelayTime);
               break;
        case 'L':
               DelayTime = GetDelay();
               LEFT(DelayTime);
               break;
        case 'R':
               DelayTime = GetDelay();
               RIGHT(DelayTime);
               break;
    }
  }
}
```

Figure 5.8 Program listing

Figure 5.9 shows the robot assembly.

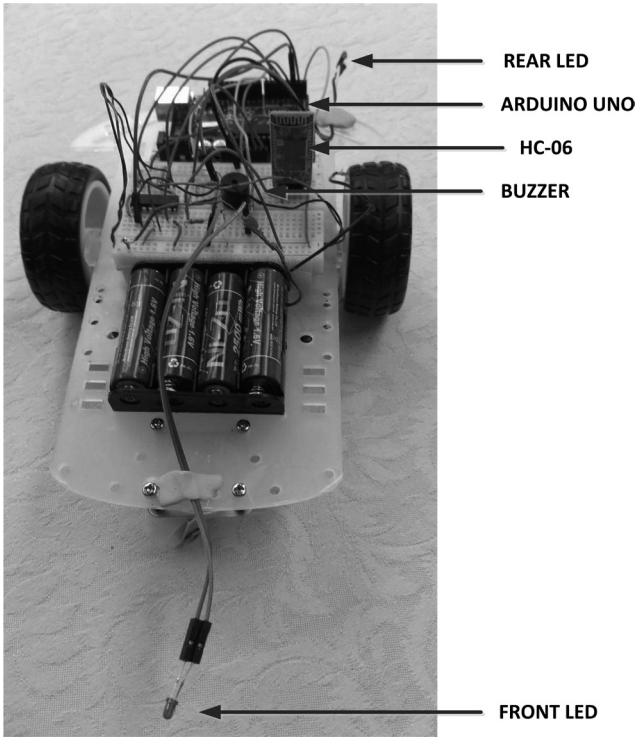

Figure 5.9 Robot assembly

5.3 Summary

In this Chapter we have seen how to control a robot remotely from a mobile phone using the Bluetooth interface.

In the next Chapter we shall be exploring how to control the same robot again from a mobile phone, but this time using Wi-Fi for communication and by sending UDP messages.

CHAPTER 6 • ARDUINO WI-FI ROBOT CONTROL

6.1 Overview

Wi-Fi is one of the most commonly used Internet communications device, found in most homes and offices in developed countries. It is a technology for wireless local area networking, based on the IEEE 802.11 standards. Devices that can use Wi-Fi technology include desktop computers, laptop computers, smart phones, tablets, digital cameras, modern printers, smart TVs and many more Internet compatible devices. These devices access the Internet via an Access Point (AP). An AP has a typical range of 20 metres indoors, although extender devices are available to extend this range to several metres or more. Wi-Fi commonly uses the 2.4 GHz and 5 GHz UHF frequencies.

In this Chapter we shall look at how to control our robot remotely using Wi-Fi via our mobile phone to give commands to the Arduino Uno mounted on the robot chassis. The robot used in this Chapter is the one used in Chapter 3. By giving commands on our mobile phone we shall see how to move the robot forwards, backwards, turning left or right, and so on. In this project an Android type mobile phone is used to send commands to the robot. In this project, UDP data packets are used for communication between the mobile phone and the Arduino Uno.

6.2 PROJECT 1 – Wi-Fi Based Remote Robot Control

In this project we shall be using the robot we used in Chapter 3. Additionally, we will mount a front LED, a back LED and a buzzer on our robot chassis as in the project in Chapter 5. We shall be controlling the robot movements, the LEDs, and the buzzer separately from our mobile phone by sending UDP packets.

Same commands as the ones given in Table 5.1 and Table 5.2 will be used in this project.

6.2.1 Block Diagram

Figure 6.1 shows the block diagram of the project. Arduino Uno has no on-board Wi-Fi capability. For this reason a Wi-Fi module is connected to the Arduino Uno so that communication with a mobile phone can be achieved.

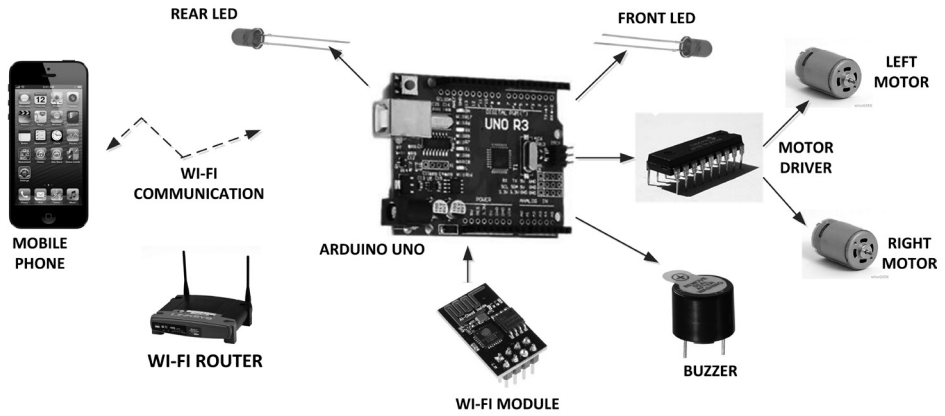

Figure 6.1 Block diagram of the project

6.2.2 Circuit Diagram

In this project an ESP-01 type Wi-Fi module with the ESP8266 processor on-board and is connected to the Arduino Uno so that the Arduino can communicate over the Wi-Fi link. Figure 6.2 shows the ESP-01 module. This is a very low-cost small Wi-Fi enabled module that has an on-board processor and can either be used as a standalone processor with limited I/O capability, or it can be connected to a host processor to give Wi-Fi capability to the host processor. In this project ESP-01 is used to give Wi-Fi capability to the Arduino Uno. This is accomplished by sending AT commands to the ESP-01 to configure it as a Wi-Fi device and then to communicate via the Wi-Fi link.

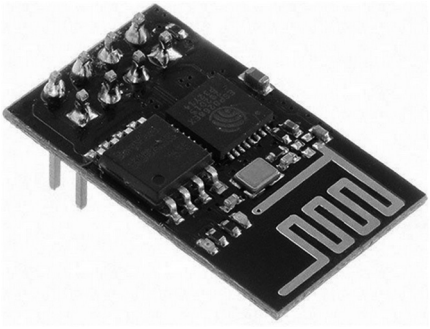

Figure 6.2 ESP-01 module

ESP-01 is a 3.3V compatible device and its I/O pins must not be raised to greater than +3.3V. The device has 8 pins:

VCC: Power supply pin. Must be connected to a +3.3V external power supply
GND: Power supply ground
GPIO0: I/O pin. This pin must be connected to +3.3V for normal operation, and to 0V for uploading firmware to the chip
GPIO2: I/O pin
RST: Reset pin. Must be connected to +3.3V for normal operation

CH_PD: Enable pin. Must be connected to +3.3V for normal operation
TX: Serial output pin
RX: Serial input pin

Figure 6.3 shows the circuit diagram of the project. It is important to use an external +3.3V power supply for the ESP-01 board since the Arduino +3.3V power supply cannot provide enough current for the ESP8266. In this project the LM1086-3.3 power regulator chip is used to provide power to the ESP-01 board. The regulator receives power from the battery and its output is connected to the VCC pin of the ESP-01. Additionally, the GPIO0, RST, and CHD_PD inputs of the ESP-01 are all connected to +3.3V (you may consider connecting the RST input through a switch so that this pin is connected to GND when the switch is pressed, thus enabling the device to be reset if required).

The output voltage of an Arduino pin is +5V and this is too high for the inputs of the ESP-01. In this project a resistor potential divider circuit is used to lower the serial output voltage of the Arduino Uno to +3.3V before it is connected to the RX input of ESP-01. The TX output of ESP-01 is connected to an input pin of the Arduino which is configured as a serial input by the software. Pins 9 and 10 of the Arduino Uno are used as the serial RX and TX pins respectively.

The front and back (tail) LEDs are connected to Arduino Uno port pins 7 and 6 respectively through 330 Ohm current limiting resistors. The positive input of the active buzzer is connected to pin 8 of the Arduino, and the other pin is connected to GND.

1A, 2A, 3A and 4A pins of the L293 motor controller chip are connected to port pins 2,3,4 and 5 of the Arduino Uno. As in the previous robot control projects, left and right motors of the robot are connected to pins 3,6 and 11,14 of the L293 motor driver chip.

Power to the Arduino is supplied using 4 x Ni-Zn batteries, connected to Vin pin of the Arduino Uno. The two wheel motors also receive their power from these batteries through pin 8 of the L293 chip. Notice that you must not power the motors from +5V pin of the Arduino Uno as it may not be able to supply the required current for the two motors. Always use an external power supply for the motors.

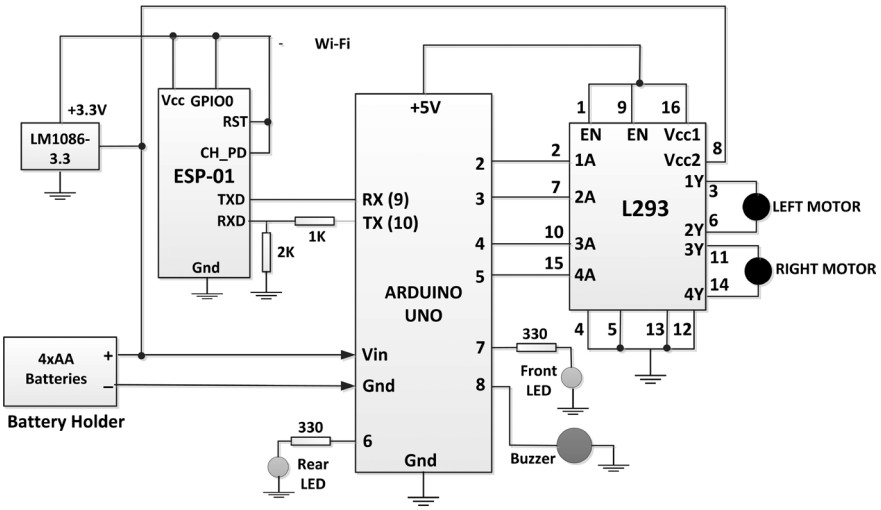

Figure 6.3 Circuit diagram of the project

Notice that the circuit is built on the breadboard mounted on top of the robot chassis. ESP-01 board is not breadboard compatible and you will need to purchase a suitable adapter so that you can mount it on the breadboard.

6.2.3 UDP or TCP ?

There are basically two protocols used in network applications (e.g. Wi-Fi): TCP/IP and UDP. TCP/IP is a connection based protocol used to send data reliably from one point to another one. UDP on the other hand is a connectionless protocol to send data fast from one point to another one. Although TCP/IP is slow, it is reliable and guarantees the delivery of packets. In addition, the sender receives an acknowledgement when a packet is delivered successfully. UDP does not guarantee the delivery of packets but it is much easier to setup and it is much faster than the TCP/IP with smaller overheads. In this project we shall be using the UDP protocol for simple and fast packet delivery from the mobile phone to the Arduino Uno.

6.2.4 Android Mobile Phone Apps

In this project we will be sending commands in the form of UDP packets from an Android based mobile phone. We therefore need an application on our mobile phone where we can send UDP data via Wi-Fi. There are many such applications free of charge in **Play Store**. The one used in this project is called **UDP RECEIVE and SEND** by Wezzi Studios (version 4.0). You should install this apps (or a similar one) on your Android phone so that you can send UDP data to the Arduino Uno. This application can send and receive UDP packets from a mobile phone to any other device running the UDP protocol. The screen is in two parts: the upper Sender part and the lower Receiver part (see Figure 6.4). The user must specify the destination IP address, port number, and the message to be sent. Clicking the **SEND UDP** MESSAGE button will send the packet to its destination.

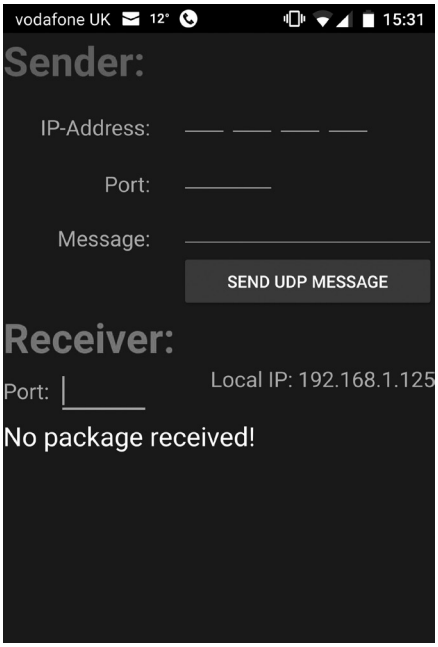

Figure 6-4 Android UDP send/receive apps

6.2.5 Controlling the Robot

You should be able now to control your robot by sending commands from your mobile phone. The steps to send commands to your robot are as follows (you should wait until the ESP-01 is ready to accept a command. This can be up to 30 seconds):

- Make sure that the program is uploaded into your Arduino Uno and that power is applied to the robot

- Start the apps **UDP RECEIVE and SEND** on your mobile phone

- Enter the following command and click **SEND UDP** (see Figure 6.5). Notice that all commands must be terminated with the # character. Also, the time delay in robot movement commands should be an integer not be greater than 99 seconds. Notice that in this project the IP address of the ESP-01 was **192.168.1.160** and the port used for UDP communication was **5000**:

  h1#

  The front LED should turn ON. To turn it OFF enter command **h0#**
  Similarly, to turn the buzzer on enter **b1#** and to turn it OFF enter **b0#**

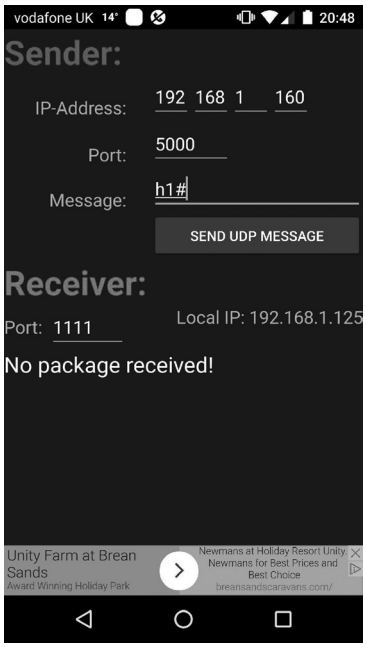

Figure 6.5 Command to turn ON the front LED]

- To move the robot forward for 3 seconds enter the command:

  F03#

- To move the robot reverse for 15 seconds enter the command:

  F15#

6.2.6 Program Listing

Figure 6.6 shows the program listing (program: mobile2). In this program the software serial library is used instead of the hardware serial library (Pins 0 and 1, i.e. TX and RX) for communicating with the ESP-01. The reason for this is because the Serial Monitor uses the hardware serial pins TX and RX and we may need to use the Serial Monitor to display various messages during the development of the program. By using the software serial library we have freed the hardware serial pins and we can therefore use the Serial Monitor. Software serial library is configured to use pins 9 and 10 for RX and TX respectively. The serial link is named **esp82666**.

At the beginning of the program motor driver connections, LED connections, and the buzzer connection are defined. Inside the setup routine motor pins, LEDs and the buzzer are configured as outputs. Also, the LEDs and the two motors are turned OFF to start with. In addition, the speed of the software serial link esp8266 is set to 115200 (ESP-01 board by default communicates at 115200 baud). The Serial Monitor is also started since we may

want to display various messages during the development of the program. ESP-01 is reset by sending it the AT command **AT+RST**.

As in the previous project in Chapter 5, functions FORWARD, REVERSE, LEFT and RIGHT move the robot in the specified directions. These functions have arguments which specify for how long the action should take place in seconds.

Function **SentToEsp8266** receives the command and the required timeout in seconds as its arguments and sends this command to the ESP-01 board. The function waits a specified number of milliseconds before returning so that the ESP-01 is ready to receive the next command. The response of the ESP8266 to the send AT command is displayed in this function by the Serial Monitor. Therefore, you should open the Serial Monitor if you wish to see the response such as the IP address allocated to the ESP-01 board.

Inside the main program, AT commands are sent to ESP-01 to connect it to the local Wi-Fi and then waits to receive UDP packets. Notice that each command is terminated with a carriage-return and line-feed pair. Command **AT+CWJAP** connects ESP-01 to the local Wi-Fi where the SSID and the Password of the Wi-Fi must be specified in this command. Command **AT+CIFSR** displays the IP address allocated to the ESP-01 by the Wi-Fi router. Notice that the responses to all these commands are displayed by the Serial Monitor so that you know what the allocated IP address to the ESP-01 is since this will be required when sending UDP data from the mobile phone. Command **AT+CIPSTART** configures ESP-01 to wait for UDP packets on port number 5000 and from any source.

The remainder of the program is the same as the program in Chapter 5 where the data is received into array **cmd** and then the commands are decoded using a **switch** statement.

```
/*****************************************************************
*                 REMOTE ROBOT CONTROL USING Wi-Fi
*                 ================================
*
* In this project a mobile robot is connected to the Arduino Uno.
* In addition, a front LED, a back LED, and a buzzer are connected
* to the Arduino Uno.
*
* An ESP-01 (ESP8266 procesor) board is attached to the Arduino Uno
* so that it communicates with a mobile phone using Wi-Fi. The
* commands sent from the mobile phone can turn ON/OFF the LEDs
* and the buzzer,  and it can also control the movements of the robot.
*
* The ESP-01 is connected to pins 9 and 10 of the Arduino and uses
* software serial communications library of Arduino.
*
* Valid commands are (The ommands must be terminated with the
* "#" character):
*
```

```
* h1 Turn ON front LED, h0 Turn OFF front LED
* t1 turn ON tail LED,  t0 Turn OFF tail LED
* b1 turn ON buzzer,    b0 turn OFF buzzer
* Fn Move robot forward n seconds
* Bn Move robot reverse n seconds
* Ln Turn robot left n seconds
* Rn Turn robot right n seconds
*
*
* File  : mobile2
* Date  : September 2017
* Author: Dogan Ibrahim
*****************************************************************/
#include <SoftwareSerial.h>                       // Software serial library
SoftwareSerial esp8266(9, 10);                    // 9=RX, 10=TX
//
// Define motor driver connections
//
#define L1A 2                                     // LEFT motor 1A
#define L2A 3                                     // LEFT motor 2A
#define R3A 4                                     // RIGHT motor 3A
#define R4A 5                                     // RIGHT motor 4A
String str;
//
// Define LED and buzzer connections
//
#define FrontLED 7                                // Front LED
#define TailLED 6                                 // Tail LED
#define Buzzer 8                                  // Buzzer

char cmd[10];                                     // Array to store commands
int DelayTime;
int First = 1;

//
// Configure 1A, 2A, 3A, 4A, LEDs and Buzzer as outputs. Turn OFF
// LEDs and Buzzer at the begining. Also stop the motors at the
// beginning just in case. Start the software serial link named
// esp8266 and reset the ESP-01 by sending an AT command.
//
void setup()
{
  pinMode(L1A, OUTPUT);
  pinMode(L2A, OUTPUT);
  pinMode(R3A, OUTPUT);
  pinMode(R4A, OUTPUT);
```

```
  pinMode(FrontLED, OUTPUT);
  pinMode(TailLED, OUTPUT);
  pinMode(Buzzer, OUTPUT);
  digitalWrite(FrontLED, LOW);
  digitalWrite(TailLED, LOW);
  digitalWrite(Buzzer, LOW);
  StopMotors();
  esp8266.begin(115200);
  Serial.begin(9600);
  SendToEsp8266("AT+RST\r\n", 2000);
}

//
// This function stops the motor
//
void StopMotors()
{
  digitalWrite(L1A, LOW);
  digitalWrite(L2A, LOW);
  digitalWrite(R3A, LOW);
  digitalWrite(R4A, LOW);
}

//
// This function moves the robot forward for tim seconds. Both
// motors rotate in the same direction
//
void FORWARD(char tim)
{
  digitalWrite(L1A, LOW);
  digitalWrite(L2A, HIGH);
  digitalWrite(R3A, LOW);
  digitalWrite(R4A, HIGH);
  delay(tim*1000);
  StopMotors();
}

//
// This function reverses the motor for tim seconds. Both motors
// rotate in the same direction
//
void REVERSE(char tim)
{
  digitalWrite(L1A, HIGH);
  digitalWrite(L2A, LOW);
  digitalWrite(R3A, HIGH);
```

```
  digitalWrite(R4A, LOW);
  delay(tim*1000);
  StopMotors();
}

//
// This function turns the motor LEFT for tim seconds. Left motor
// is stopped and the right motor is activated
//
void LEFT(char tim)
{
  digitalWrite(L1A, HIGH);
  digitalWrite(L2A, LOW);
  digitalWrite(R3A, LOW);
  digitalWrite(R4A, HIGH);
  delay(tim*1000);
  StopMotors();
}

// This function turns the motor RIGHT for tim seconds. Right motor
// is stopped and the left motor is activated
//
void RIGHT(char tim)
{
  digitalWrite(L1A, LOW);
  digitalWrite(L2A, HIGH);
  digitalWrite(R3A, HIGH);
  digitalWrite(R4A, LOW);
  delay(tim*1000);
  StopMotors();
}

//
// This function extracts the required delay in a motor movement
// command. The delay is in seconds
//
int GetDelay()
{
  int dly = 10*(cmd[1]-'0') + cmd[2] - '0';
  return dly;
}

void SendToEsp8266(String cmd, const int tim)
{
  esp8266.print(cmd);                              // Send cmd to ESp-01
  long int time = millis();
```

```
  while((time+tim) > millis())                    // Wait required time
  {
      while(esp8266.available())                  // If response available
      {
        char ch= esp8266.read();                  // Read the response
        Serial.print(ch);                         // Display the response
      }
  }
}

//
// Main program loop. Read the command from the ESP-01 device
// through Wi-Fi, decode the command and implement the control action
// as required. AT commands are sent to the ESP-01 to connect the
// Arduino Uno to Wi-Fi and then wait to receive UDP packets.
//
void loop()
{
  if(First == 1)
  {
    First = 0;
    SendToEsp8266("AT+CWMODE=1\r\n", 5000);
    SendToEsp8266("AT+CWJAP=\"BTHomeSpot-XNH\",\"49345abaeb\"\r\n", 5000);
    delay(10000);
    SendToEsp8266("AT+CPIMUX=1\r\n", 2000);
    SendToEsp8266("AT+CIFSR\r\n",3000);
    SendToEsp8266("AT+CIPSTART=\"UDP\",\"0.0.0.0\",0,5000,2\r\n", 5000);
    delay(100);
  }
  cmd[0]=0;                                       // Clear command array
  cmd[1]=0;
  cmd[2]=0;

  if(esp8266.available())                         // Command available ?
  {
    esp8266.readBytesUntil('#', cmd, 3);          // Read a command

    switch(cmd[0])                                // Decode and implement
    {
      case 'h':                                   // Head LED
              if(cmd[1] == '1')
                digitalWrite(FrontLED, HIGH);
              else if(cmd[1] == '0')
                digitalWrite(FrontLED, LOW);
              break;
      case 't':                                   // Tail LED
```

```
                if(cmd[1] == '1')
                  digitalWrite(TailLED, HIGH);
                else if(cmd[1] == '0')
                  digitalWrite(TailLED, LOW);
                break;
        case 'b':                                       // Buzzer
                if(cmd[1] == '1')
                  digitalWrite(Buzzer, HIGH);
                else if(cmd[1] == '0')
                  digitalWrite(Buzzer, 0);
                break;
        case 'F':                                       // Forward
                 DelayTime = GetDelay();
                 FORWARD(DelayTime);
                 break;
        case 'B':                                       // BAckwards
                 DelayTime = GetDelay();
                 REVERSE(DelayTime);
                 break;
        case 'L':                                       // Left
                 DelayTime = GetDelay();
                 LEFT(DelayTime);
                 break;
        case 'R':                                       // Right
                 DelayTime = GetDelay();
                 RIGHT(DelayTime);
                 break;
    }
  }
}
```

Figure 6.6 Program listing

Figure 6.7 shows the robot assembly.

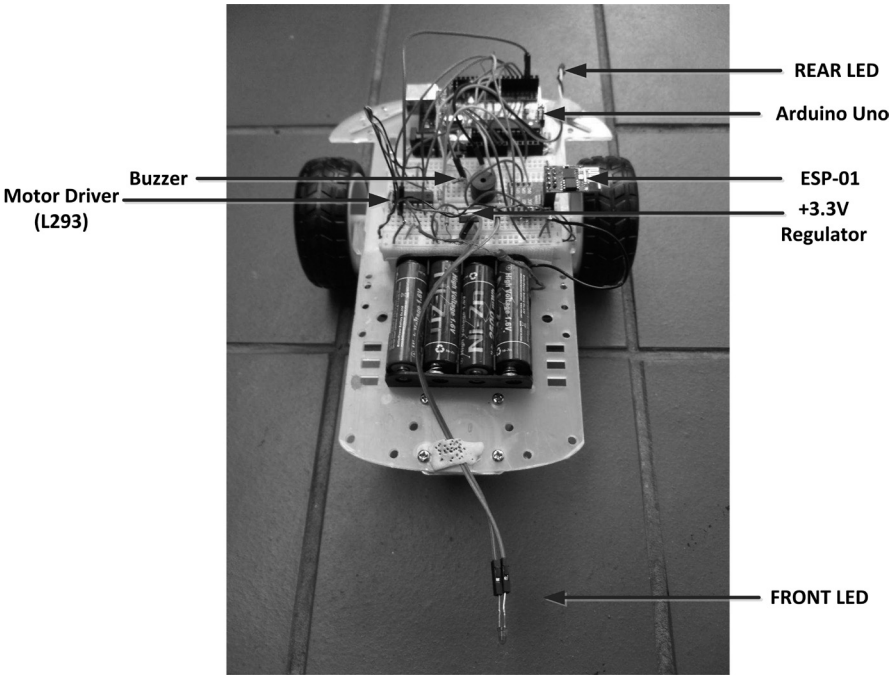

Figure 6.7 Robot assembly

6.3 Summary

In this Chapter we have seen how to control a robot remotely from a mobile phone using Wi-Fi interface. An Arduino Uno is used in this Chapter as the controller.

In the next Chapter we shall be looking at how to control the same robot again from a mobile phone, but this time using Bluetooth on a Raspberry Pi.

CHAPTER 7 • RASPBERRY PI ZERO W WI-FI ROBOT CONTROL

7.1 Overview

In this Chapter we shall see how to control our robot remotely using Wi-Fi on our mobile phone to give commands to the RPi ZW mounted on the robot chassis. The robot used in this Chapter is the one used in Chapter 3. By giving commands on our mobile phone we shall see how to move the robot forwards, backwards, turning left or right, and so on. In this project an Android type mobile phone is used to send commands to the robot. In this project, UDP data packets are used for communication between the mobile phone and the RPi ZW.

7.2 PROJECT 1 – Wi-Fi Based Remote Robot Control

In this project we shall be using the robot we used in Chapter 3, but additionally we have a front LED, a back LED, and a buzzer mounted on the robot chassis.

Same commands as the ones given in Table 5.1 and Table 5.2 will be used in this project.

7.2.1 Block Diagram

The block diagram of the project is shown in Figure 7.1. Because the RPi ZW has Wi-Fi capability, there is no need to use an external Wi-Fi module.

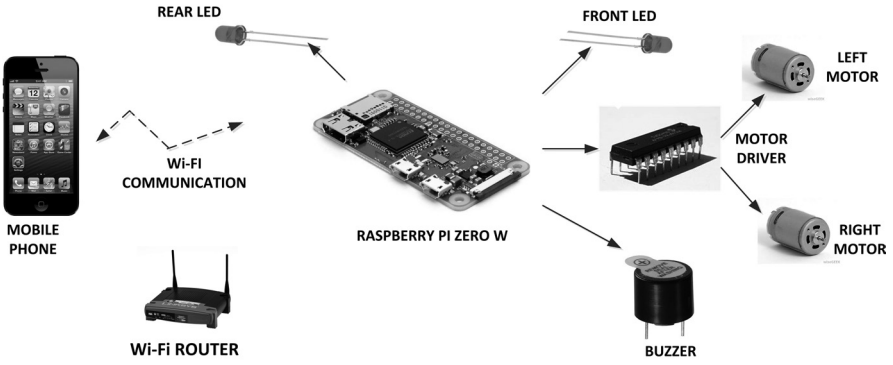

Figure 7.1 Block diagram of the project

7.2.2 Circuit Diagram

The circuit diagram of the project is shown in Figure 7.2. GPIO2, GPIO3, GPIO17 and GPIO27 pins are connected to pins 1A, 2A, 3A, and 4A of the motor driver IC respectively. Front and rear LEDs are connected to ports GPIO9 and GPIO22 respectively. The buzzer is connected to GPIO10. Power to the motors is supplied by the batteries since the RPi ZW cannot provide power for the motors. A +5V portable charger unit is used to supply power to the raspberry Pi Zero W.

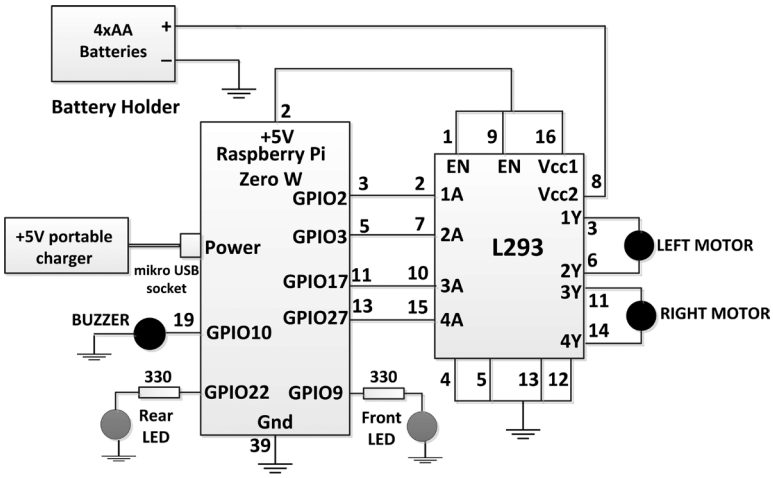

Figure 7.2 Circuit diagram of the project

7.2.3 Program Listing

The communication between the Android mobile phone and the RPi ZW is based on Wi-Fi. As in Chapter 6, **UDP RECEIVE and SEND** by Wezzi Studios (version 4.0) apps is used on the mobile phone to send UDP packets to the RPi ZW.

The program listing is shown in Figure 7.3 (program: mobile1.py). The Python program on the RPi ZW uses the socket library. At the beginning of the program the socket library is imported into the program, motor driver pins, LED pins, and the buzzer pin are assigned to GPIO ports. These pins are then configured as outputs.

Functions FORWARD, REVERSE, LEFT and RIGHT move the robot by the specified number of seconds in the selected direction.

The program uses port 5000 to communicate with the mobile phone using UDP packets. The IP address of the RPi ZW Wi-Fi connection must be specified in the code. This can be found by entering the following code at the RPi ZW command mode:

```
pi@raspberrypi:~ $ sudo ifconfig
```

Figure 7.4 shows a typical display when the above command is entered. In this example, the IP address of the RPi ZW was **192.168.1.161** and this was assigned to variable **IP** in the program.

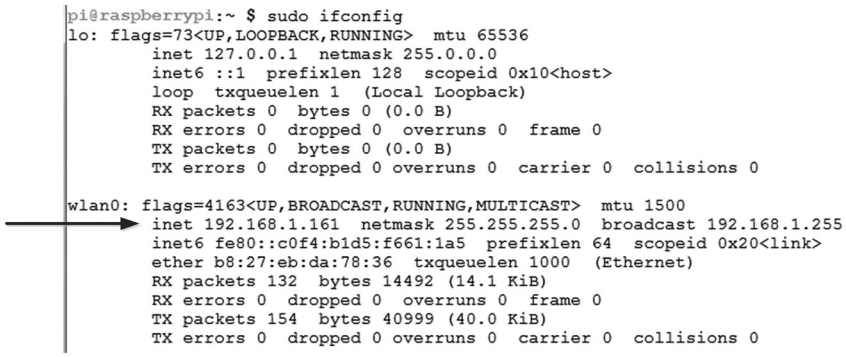

Figure 7.4 Finding the IP address of the RPi ZW

An endless loop is then formed where inside this loop UDP packets are received in the form of commands using the following statement:

```
data, addr = sock.recvfrom(10)
```

The received commands are decoded and the required actions are implemented. For example, if the command is **h1** to turn ON the front LED, then the following code is executed:

```
if data[0] == 'h':
    if data[1] == '1':
          GPIO.output(FrontLED, 1)
    else:
          GPIO.output(FrontLED, 0)
```

As before, you should enter the program name in the following format into file **/etc/rc.local** so that the program starts automatically after the RPI ZW re-starts:

python /home/pi/mobile1.py &

```
#-----------------------------------------------------------------------
#                          REMOTE ROBOT CONTROL VIA WI-FI
#                          ==============================
#
# In this project the RPi ZW is mounted on the robot chassis together
# with the L293 motor driver IC. In addition, two LEDs are mounted on
# the robot chasses: Front LED and Rear LED. Additionally, a buzzer is
# mounted on the breadboard. The Raspberry Pi Zero W has on board
# Wi-Fi and Bluetooth capability. In this example only the Wi-Fi is
# used.
#
# In this project the LEDs, buzzer and the robot movements are all
# controlled from a mobile phone using the Wi-Fi link to establish
# communciation between the mobile phone and the RPi ZW.
```

```
#
# The commands are as follows (In the robot movement commands the
# time must be an integer number):
#
# h1 Turn ON front LED, h0 Turn OFF front LED
# t1 Turn ON rear LED,  t0 Turn OFF rear LED
# b1 Turn ON buzzer,    b0 Turn OFF buzzer
# Fn Move robot forward n seconds
# Bn Move robot reverse n seconds
# Ln Turn robot left n seconds
# Rn Turn robot right n seconds
#
#
# File  : mobile1.py
# Date  : September 2017
# Author: Dogan Ibrahim
#-----------------------------------------------------------------------
import RPi.GPIO as GPIO
import time
import socket
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

#
# L293 control pins
#
L1A = 2    # GPIO2
L2A = 3    # GPIO3
R3A = 17     # GPIO17
R4A = 27     # GPIO2

#
# LED and Buzzer pins
#
FrontLED = 9
RearLED = 22
Buzzer = 10
global data

#
# Configure L293 control pins, LED, and buzzer as outputs
#
GPIO.setup(L1A, GPIO.OUT)
GPIO.setup(L2A, GPIO.OUT)
GPIO.setup(R3A, GPIO.OUT)
GPIO.setup(R4A, GPIO.OUT)
```

```
GPIO.setup(FrontLED, GPIO.OUT)
GPIO.setup(RearLED, GPIO.OUT)
GPIO.setup(Buzzer, GPIO.OUT)

#
# Stop the motor
#
def StopMotor():
  GPIO.output(L1A, 0)
  GPIO.output(L2A, 0)
  GPIO.output(R3A, 0)
  GPIO.output(R4A, 0)
  return

#
# Set forward movement
#
def FORWARD(tim):
  GPIO.output(L1A, 0)
  GPIO.output(L2A, 1)
  GPIO.output(R3A, 0)
  GPIO.output(R4A, 1)
  time.sleep(tim)
  StopMotor()
  return

#
# Set reverse movement
#
def REVERSE(tim):
  GPIO.output(L1A, 1)
  GPIO.output(L2A, 0)
  GPIO.output(R3A, 1)
  GPIO.output(R4A, 0)
  StopMotor()
  return

#
# Turn left
#
def LEFT(tim):
  GPIO.output(L1A, 1)
  GPIO.output(L2A, 0)
  GPIO.output(R3A, 0)
  GPIO.output(R4A, 1)
  time.sleep(tim)
```

```
  StopMotor()
  return

#
# Turn right
#
def RIGHT(tim):
  GPIO.output(L1A, 0)
  GPIO.output(L2A, 1)
  GPIO.output(R3A, 1)
  GPIO.output(R4A, 0)
  time.sleep(tim)
  StopMotor()
  return

#
# This function extracts the delay in seconds
#
def GetDelay():
  global data
  No = int(data[1:])
  return No

#
# Start of the main program loop. Get a command from the mobile
# phone, decode the command, and implement the required action
##
# Stop the motors at the beginning of the program. Define the UDP
# port number and the IP address of our RPi ZW (can be found using
# command sudo ifconfig in command mode).
#
StopMotor()
PORT = 5000
IP = “192.168.1.161”
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP, PORT))
#
# Start of the program loop where a command is received and implemented
#
while True:
  data, addr = sock.recvfrom(10)
#
# Decode and implement the command. First index of data[0] is the comamnd
# itself (h,t,b,F,B,R,L). In ythe case of LED and Buzzer controls, the
# second index cmd[1] is 1 or 0 depending whether or not we wish to turn
# ON or OFF respectively. In teh case of motor movement cmd[1],cmd[2]...
```

```
# are the times in seconds that the motor should be ON
#
  if data[0] == 'h':
    if data[1] == '1':
      GPIO.output(FrontLED, 1)
    else:
      GPIO.output(FrontLED, 0)
  elif data[0] == 't':
    if data[1] == '1':
      GPIO.output(RearLED, 1)
    else:
      GPIO.output(RearLED, 0)
  elif data[0] == 'b':
    if data[1] == '1':
      GPIO.output(Buzzer, 1)
    else:
      GPIO.output(Buzzer, 0)
  elif data[0] == 'F':
    DelayTime = GetDelay()
    FORWARD(DelayTime)
  elif data[0] == 'B':
    DelayTime = GetDelay()
    REVERSE(DelayTime)
  elif data[0] == 'L':
    DelayTime = GetDelay()
    LEFT(DelayTime)
  elif data[0] == 'R':
    DelayTime = GetDelay()
    RIGHT(DelayTime)
```

Figure 7.3 Program listing

When you finish your project don't forget to remove the above line from file **/etc/rc.local,** otherwise the program will run every time your RPi ZW is re-started. You should also shutdown your RPi ZW orderly instead of just removing the power cable. The command to shutdown orderly is:

```
pi@raspberrypi:~ $  sudo shutdown –h now
```

Figure 7.5 shows the command entered to turn the front LED ON. The robot assembly is shown in Figure 7.6.

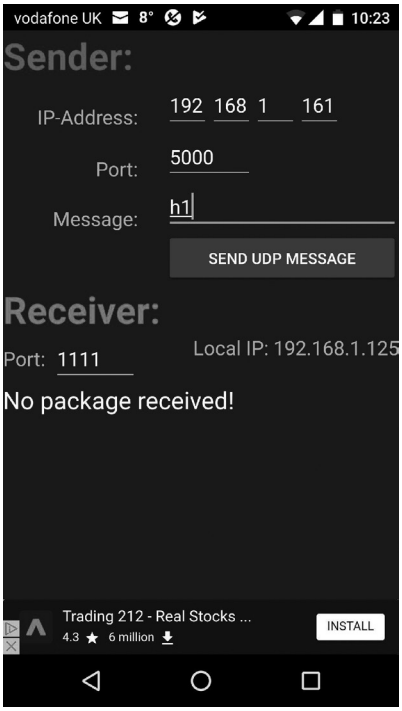

Figure 7.5 Command to turn ON the front LED

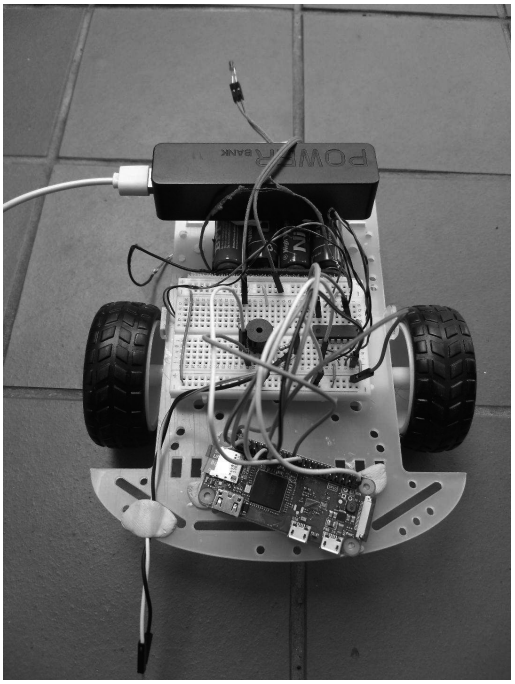

Figure 7.6 The robot assembly

7.3 Summary

In this Chapter we have seen how to control a robot remotely from a mobile phone using Wi-Fi interface. A Raspberry Pi Zero W is used in this Chapter as the controller.

In the next Chapter we shall be looking at how to control the same robot again from a mobile phone, but this time using Bluetooth on a Raspberry Pi Zero W.

CHAPTER 8 • RASPBERRY PI ZERO W BLUETOOTH ROBOT CONTROL

8.1 Overview

Raspberry Pi Zero W is both Wi-Fi and Bluetooth enabled. In this Chapter we shall see how to control our robot remotely using Bluetooth on our mobile phone to give commands to the RPi ZW mounted on the robot chassis. The robot used in this Chapter is the one used in Chapter 7. By giving commands on our mobile phone we shall see how to move the robot forwards, backwards, turning left or right, and so on. In this project an Android type mobile phone is used to send commands to the robot.

8.2 PROJECT 1 – Bluetooth Based Remote Robot Control

In this project we shall be using the robot we used in Chapter 7. Same commands as the ones given in Table 5.1 and Table 5.2 will be used in this project.

8.2.1 Block Diagram

The block diagram of this project is same as the one given in Figure 7.1, but instead of Wi-Fi, in this project Bluetooth is used for communication between the mobile phone and the RPi ZW.

8.2.2 Circuit Diagram

The circuit diagram of this project is as shown in Figure 7.2. The LEDs, buzzer, and the motor driver IC are all connected to the RPi ZW. Power to the RPi ZW is supplied from a +5V portable charger unit mounted on the robot chassis. Power to the wheel motors is supplied from 4 x NiZn AA type batteries as in the previous robot control projects.

8.2.3 Enabling Bluetooth on Raspberry Pi Zero W

In this project we shall be sending commands from an Android mobile phone to our RPi ZW. We must therefore first of all enable Bluetooth from the Settings menu on our Android device. In this example the Android Bluetooth device name is **VFD900**.

There are two ways you can enable Bluetooth on the RPi ZW: using graphical desktop (GUI mode), or using the command mode.

Using the Graphical Desktop

The GUI mode is described in Appendix A.8 and is repeated here:

- Start the VNC server on your RPi ZW and login to the RPi ZW using the VNC Viewer.
- Click on the Bluetooth icon on your RPi ZW at the top right hand side, and turn Bluetooth ON. Then, select **Make Discoverable.** You should see the Bluetooth icon flashing (Figure 8.1)

Figure 8.1 Enable Bluetooth on your RPi ZW

- Select **raspberrypi** in the Bluetooth menu on your mobile device
- Accept the pairing request on your RPi ZW as shown in Figure 8.2

Figure 8.2 Bluetooth pairing request

- You should now see the message **Connected Successfully** on your RPi ZW

Using Command Mode

You can enable Bluetooth on your RPi ZW using the command mode. Additionally you can make Bluetooth discoverable, scan for nearby Bluetooth devices and then connect to a Bluetooth device. The steps are given below (characters types by the user are in bold):

- Make your Bluetooth discoverable with the following command:

  pi@raspberrypi: ~ $ **sudo hciconfig hci0 piscan**

- Start the Bluetooth tool on your Rpi ZW from the command mode:

  pi@raspberrypi:~ $ **bluetoothctl**

- Turn Bluetooth ON:

  [bluetooth]# **power on**

- Configure Bluetooth to run:

  [bluetooth]# **agent on**
  [Bluetooth]# **default-agent**

- Make device discoverable:

  [Bluetooth]# **discoverable on**

- Scan for nearby Bluetooth devices. You should see the nearby Bluetooth devices

listed with their MAC addresses. Make a note of the MAC address of the device you wish to connect to (Android mobile phone in this project) as we will be using this address to connect to the device. An example is shown in Figure 8.3:

[bluetooth]# **scan on**

```
pi@raspberrypi:~ $ bluetoothctl
[NEW] Controller B8:27:EB:25:87:C9 raspberrypi [default]
[bluetooth]# power on
Changing power on succeeded
[bluetooth]# agent on
Agent registered
[bluetooth]# scan on
Discovery started
[CHG] Controller B8:27:EB:25:87:C9 Discovering: yes
[NEW] Device A4:77:33:BD:1A:7F A4-77-33-BD-1A-7F
[CHG] Device A4:77:33:BD:1A:7F Name: Chromecast7872
[CHG] Device A4:77:33:BD:1A:7F Alias: Chromecast7872
[bluetooth]#
```

Figure 8.3 Scanning nearby devices

In this example our mobile phone is VFD900 and the Bluetooth MAC address is: 28:BE:03:7A:53:F5

- Pair the device:

  [bluetooth]# **pair 28:BE:03:7A:53:F5**

- Connect to our mobile phone:

  [bluetooth]# **connect 28:BE:03:7A:53:F5**

- Exit from the Bluetooth tool by entering Cntrl+Z

You can find the Bluetooth MAC address of your RPi ZW by entering the following command:

pi@raspberrypi:~ $ **hciconfig | grep "BD Address"**

You can change the Bluetooth broadcast name by the following command:

pi@raspberrypi:~ $ **sudo hcoconfig hci0 name "new name"**

To see your Bluetooth broadcast name, enter:

pi@raspberrypi:~ $ **sudo hcoconfig hci0 name**

Some other useful RPi ZW Bluetooth commands are:

- To reset Bluetooth adapter: **sudo hciconfig hci0 reset**
- To restart Bluetooth: **sudo invoke-rc.d bluetooth restart**

- To list Bluetooth adapters: **hciconfig**

You can find the Bluetooth MAC address of your Android phone as follows:

- Go to **Settings** menu
- Tap **About Phone**
- Tap **Status**
- Scroll down to see your **Bluetooth address**

8.2.4 Python Bluetooth Library

You will need to install the Python Bluetooth library before developing your program. This is done by entering the following command in the command mode:

```
pi@raspberrypi:~ $ sudo get-apt install python-bluez
```

8.2.5 Accessing From the Mobile Phone

In this project we shall be using the Android mobile phone apps **Arduino Bluetooth Controller** as in Chapter 5. In order to be able to access the RPi ZW from a mobile phone apps make the following changes to your RPi ZW from the command line:

- Start nano to edit the following file:

  ```
  pi@raspberrypi:~ $ sudo nano /etc/systemd/system/dbus-org.bluez.service
  ```

- Add –C at the end of the ExecStart= line. Also add another line after the ExecStart line. The final two lines should look like:

  ```
  ExecStart=/usr/lib/bluetooth/bluetoothd -C
  ExecStartPost=/usr/bin/sdptool add SP
  ```

- Exit and save the file by entering Ctrl+X and Y
- Reboot RPi ZW:

  ```
  pi@raspberrypi:~ $ sudo reboot
  ```

8.2.5 Program Listing

RPi ZW supports two types of Bluetooth sockets: RFCOMM and L2CAP. In this project RFCOMM is used. The two socket types are similar with the difference that they use different port numbers.

Figure 8.4 shows the program listing (program: mobile2.py). At the beginning of the pro-

gram RPi.GPIO, time, and Bluetooth libraries are imported into the program. The motor driver connections, LED connections, and the buzzer connections are as before. Functions FORWARD, REVERSE, LEFT and RIGHT move the robot in the specified directions with the specified duration. The Bluetooth socket is configured so that it accepts connection from any paired and connected device.

The remainder of the program is executed in an endless loop where the program waits to receive data through the Bluetooth socket, and then decodes and implements this code as before.

```
#-----------------------------------------------------------------
#                  REMOTE ROBOT CONTROL VIA BLUETOOTH
#                  ==================================
#
# In this project the RPi ZW is mounted on the robot chassis together
# with the L293 motor driver IC. In addition, two LEDs are mounted on
# the robot chasses: Front LED and Rear LED. Additionally, a buzzer is
# mounted on the breadboard. The Raspberry Pi Zero W has on board
# Wi-Fi and Bluetooth capability. In this example only the Bluetooth
# is used.
#
# In this project the LEDs, buzzer and the robot movements are all
# controlled from a mobile phone using the Bluetooth link to establish
# communciation between the mobile phone and the RPi ZW.
#
# The commands are as follows (In the robot movement commands the
# time must be an integer number):
#
# h1 Turn ON front LED, h0 Turn OFF front LED
# t1 Turn ON rear LED,  t0 Turn OFF rear LED
# b1 Turn ON buzzer,    b0 Turn OFF buzzer
# Fn Move robot forward n seconds
# Bn Move robot reverse n seconds
# Ln Turn robot left n seconds
# Rn Turn robot right n seconds
#
#
# File  : mobile2.py
# Date  : October 2017
# Author: Dogan Ibrahim
#-----------------------------------------------------------------------
import RPi.GPIO as GPIO
import time
import bluetooth
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
```

```
#
# L293 control pins
#
L1A = 2    # GPIO2
L2A = 3    # GPIO3
R3A = 17   # GPIO17
R4A = 27   # GPIO2

#
# LED and Buzzer pins
#
FrontLED = 9
RearLED = 22
Buzzer = 10
global data

#
# Configure L293 control pins, LED, and buzzer as outputs
#
GPIO.setup(L1A, GPIO.OUT)
GPIO.setup(L2A, GPIO.OUT)
GPIO.setup(R3A, GPIO.OUT)
GPIO.setup(R4A, GPIO.OUT)
GPIO.setup(FrontLED, GPIO.OUT)
GPIO.setup(RearLED, GPIO.OUT)
GPIO.setup(Buzzer, GPIO.OUT)

#
# Stop the motor
#
def StopMotor():
  GPIO.output(L1A, 0)
  GPIO.output(L2A, 0)
  GPIO.output(R3A, 0)
  GPIO.output(R4A, 0)
  return

#
# Set forward movement
#
def FORWARD(tim):
  GPIO.output(L1A, 0)
  GPIO.output(L2A, 1)
  GPIO.output(R3A, 0)
  GPIO.output(R4A, 1)
```

```
  time.sleep(tim)
  StopMotor()
  return

#
# Set reverse movement
#
def REVERSE(tim):
  GPIO.output(L1A, 1)
  GPIO.output(L2A, 0)
  GPIO.output(R3A, 1)
  GPIO.output(R4A, 0)
  StopMotor()
  return

#
# Turn left
#
def LEFT(tim):
  GPIO.output(L1A, 1)
  GPIO.output(L2A, 0)
  GPIO.output(R3A, 0)
  GPIO.output(R4A, 1)
  time.sleep(tim)
  StopMotor()
  return

#
# Turn right
#
def RIGHT(tim):
  GPIO.output(L1A, 0)
  GPIO.output(L2A, 1)
  GPIO.output(R3A, 1)
  GPIO.output(R4A, 0)
  time.sleep(tim)
  StopMotor()
  return

#
# This function extracts the delay in seconds
#
def GetDelay():
  global data
  No = int(data[1:])
  return No
```

```
#
# Start of the main program loop. Get a command from the mobile
# phone, decode the command, and implement the required action
##
# Stop the motors at the beginning of the program. Initialize the
# Bluetooth code
#
StopMotor()

ServerSock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
Port = 1
ServerSock.bind((“”, Port))
ServerSock.listen(1)
ClientSock,addr = ServerSock.accept()

#
# Start of the program loop where a command is received and implemented
#
while True:
  data = ClientSock.recv(1024)

#
# Decode and implement the command. First index of data[0] is the comamnd
# itself (h,t,b,F,B,R,L). In ythe case of LED and Buzzer controls, the
# second index cmd[1] is 1 or 0 depending whether or not we wish to turn
# ON or OFF respectively. In teh case of motor movement cmd[1],cmd[2]...
# are the times in seconds that the motor should be ON
#
  if data[0] == 'h':
    if data[1] == '1':
      GPIO.output(FrontLED, 1)
    else:
      GPIO.output(FrontLED, 0)
  elif data[0] == 't':
    if data[1] == '1':
      GPIO.output(RearLED, 1)
    else:
      GPIO.output(RearLED, 0)
  elif data[0] == 'b':
    if data[1] == '1':
      GPIO.output(Buzzer, 1)
    else:
      GPIO.output(Buzzer, 0)
  elif data[0] == 'F':
    DelayTime = GetDelay()
```

```
    FORWARD(DelayTime)
elif data[0] == 'B':
    DelayTime = GetDelay()
    REVERSE(DelayTime)
elif data[0] == 'L':
    DelayTime = GetDelay()
    LEFT(DelayTime)
elif data[0] == 'R':
    DelayTime = GetDelay()
    RIGHT(DelayTime)
```

Figure 8.4 Program listing

Figure 8.5 shows the mobile application ready to send command **h1** to the RPi ZW to turn ON the front LED. Notice that you must start the RPi ZW program before starting the apps on your mobile phone.

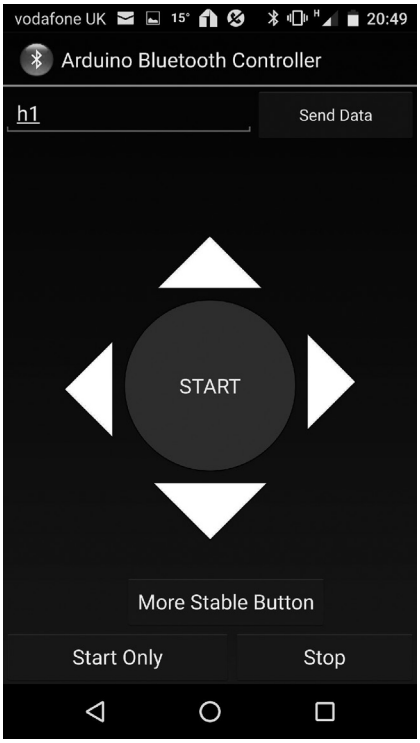

Figure 8.5 Command h1

As before, you should enter the program name in the following format into file **/etc/rc.local** so that the program starts automatically after the RPI ZW re-starts:

python /home/pi/mobile2.py &

When you finish your project don't forget to remove the above line from file **/etc/rc.local,** otherwise the program will run every time your RPi ZW is re-started. You should also shutdown your RPi ZW orderly instead of just removing the power cable. The command to shutdown orderly is:

```
pi@raspberrypi:~ $  sudo shutdown –h now
```

The robot assembly is same as in Figure 7.6.

8.3 Summary

In this Chapter we have seen how to control a robot remotely using the Bluetooth communications between an Android mobile phone and the Raspberry Pi Zero W

APPENDIX A • RASPBERRY PI ZERO W

Raspberry Pi and Raspberry Pi Zero have been around for several years. The Zero W (W for "wireless") is a new addition to the family and it incorporates WLAN and Bluetooth capabilities. In this Appendix we shall briefly look at the specifications of the Raspberry Pi Zero W (called RPi WZ in future references) and also see how the operating system can be installed on an SD card. With its low cost of around $20, the RPi WZ is an ideal choice for many wireless, robotics, control, monitoring, and automation projects.

A.1 The Hardware

- RPi WZ has the following specifications:
- Single core BCM2835 processor
- 1 GHz clock
- 512 MB RAM
- Mini HDMI port
- micro-B USB data ports
- micro-B USB power port
- CSI camera connector
- 802.11n Wireless LAN
- Bluetooth 4.0
- Micro SD card socket (for the operating system)
- 2 x 20 pin GPIO interface
- Dimensions 65mm x 30mm x 5mm

The GPIO interface is accessed through a 40-way header which should be soldered to the board.

Figure A.1 shows the RPi WZ. At the top of the board is the GPIO ports. At the bottom of the board are the HDMI socket, USB socket, and the power socket. On the left hand side is the micro SD card socket. On the right hand side is the camera interface.

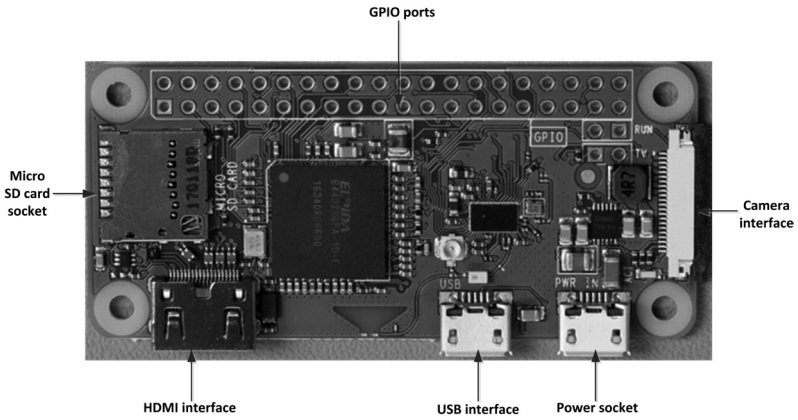

Figure A.1 Raspberry Pi Zero W

As with all the Raspberry Pi boards, a power supply, a keyboard, mouse (or another input device), and screen (monitor) must be provided to use the RPi WZ.

A.2 Setting Up the Raspberry Pi Zero W

RPi ZW is sold in several different formats. Some distributors supply all the necessary adapters, power supply, and even the operating system on a micro SD card. For example, Pimoroni (https://shop.pimoroni.com) supply the RPi ZW as either on its own or together with the necessary adapters. PiHut supply a complete RPi ZW kit with all the adapters, power supply and the pre-installed operating system on a16 GB SD card for around £15 (see Figure A.2).

Figure A.2 PiHut Raspberry Pi Zero W

The RPi ZW should be connected to a monitor through its HDMI socket. A keyboard, preferably with a mouse should be connected to the micro-USB socket, and the power supply should be connected to the micro-USB power socket.

A.3 Installing the Operating System on SD Card

If you have purchased your RPi ZW with the operating system pre-installed on an SD card then you can skip this section.

The instructions to install the operating system on a blank SD card are given below. You will need a micro SD card with a capacity of at least 8 GB of memory, although16 GB is recommended for future expansion and installing new applications and programs. You might need a SD card adapter to insert the micro SD card into your computer's card slot. The recommended operating system is called **Raspbian**.

- Download (ZIP) the current file Image of Raspbian from the following site (see Figure A.3) to a folder on your PC (e.g. to C:\RPI). It is recommended that first time users download and install the software called NOOBS (**N**ew **O**ut **O**f **B**ox **S**oftware) onto a new 8GB or larger micro SD card. The version at the time of writing this book is 2.4.3:

  www.raspberrypi.org/downloads/raspbian

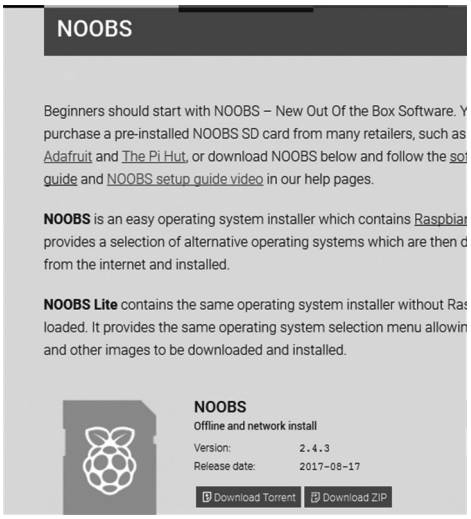

Figure A.3 Download the NOOBS software

- Install the SD Formatter software from the SD Association's web site (see Figure A.4). At the time of writing this book the software was called SD_CardFormatter0500SetupEN.exe:

  https://www.sdcard.org/downloads/formatter_4/eula_windows/index.html

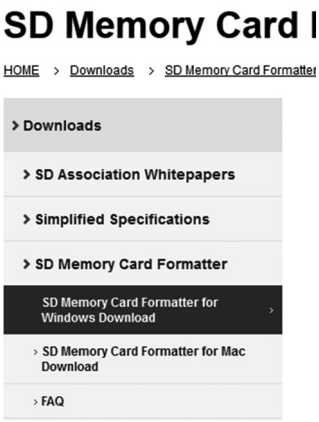

Figure A.4 Install the SD Formatter software

Insert your SD card into your computer's card socket and make note of the drive letter allocated to it. e.g. **F:** as shown in Figure A.5

Figure A.5 Make a note of the drive letter allocated to your SD card

Start the SD card formatter you have installed and enter the drive letter to format the SD card, select **Quick format** as shown in Figure A.6 (**be careful to enter the correct drive letter here otherwise you might delete all the files on your computer!**).

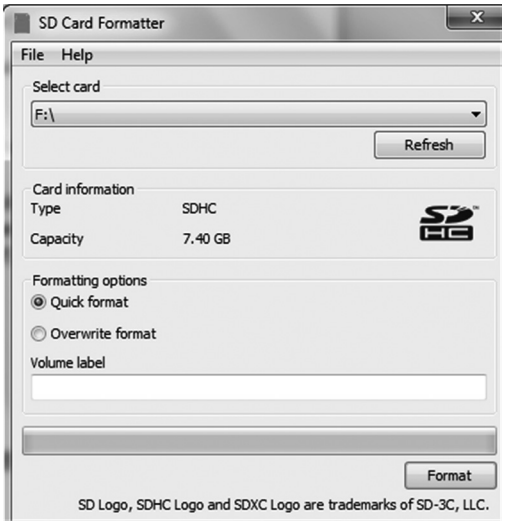

Figure A.6 Enter the SD card drive letter

- Extract all the files from the folder where you have stored the NOOBS operating system ZIP file.

- Drag all the extracted files and drop them to the newly formatted SD card. All the necessary files will be transferred to the SD card. After copying, part of the contents of the SD card are shown in Figure A.7

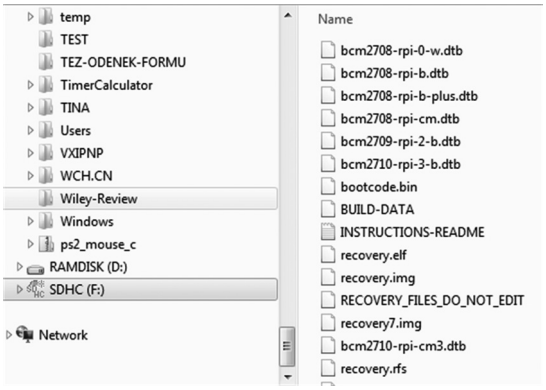

Figure A.7 Part of the contents of SD card

- Remove the SD card from your computer and install into the Raspberry Pi Zero W micro SD card slot.

Notice that further information about the installation process can be obtained from the following link:

https://www.raspberrypi.org/learning/software-guide/quickstart/

A.4 Applying Power to the Raspberry Pi Zero W

Connect the monitor to the HDMI port, and the keyboard/mouse to the USB data port and the power supply to the USB power port. After a short while you will see the startup menu on the monitor as shown in Figure A.8. Select the **Raspbian** operating system (recommended) and click the **Install** button (at the top left of the menu). Click **Yes** to confirm to install the **Raspbian** operating system. You should see the message **Raspbian: Extracting filesystem** at the bottom of the monitor. Wait for 5 to 10 few minutes until the operating system has been installed on the SD card (see Figure A.10). You should see a progress bar at the bottom of the monitor as the installation process continues.

Note that the author used the **Ultra Mini Keyboard** with built-in mouse as shown in Figure A.9. This keyboard is connected to the RPi WZ via wireless USB port.

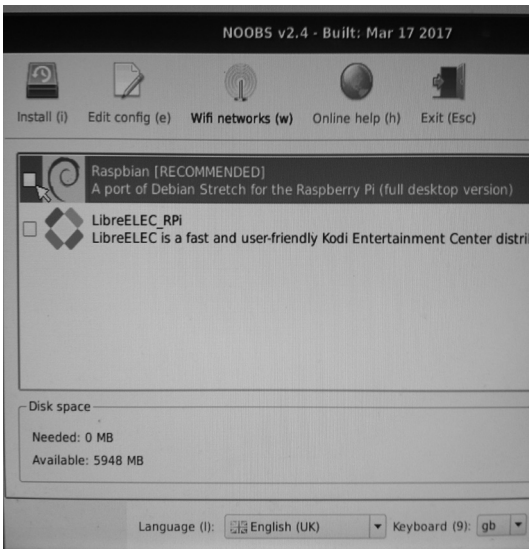

Figure A.8 The startup menu

Figure A.9 Mini keyboard/mouse

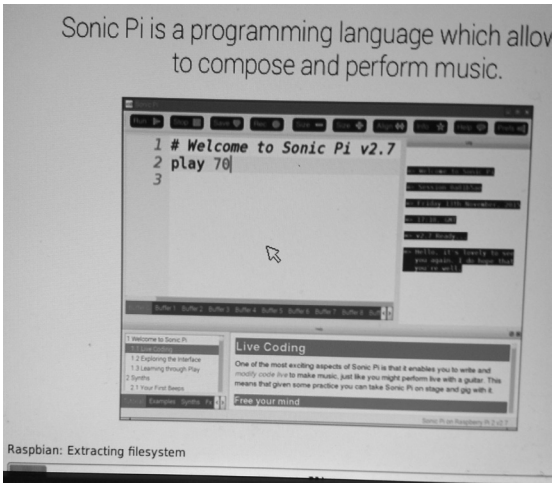

Figure A.10 Wait until the operating system is installed

At the end of the installation you should restart the RPi WZ which will display the GUI screen shown in Figure A.11 after the restart.

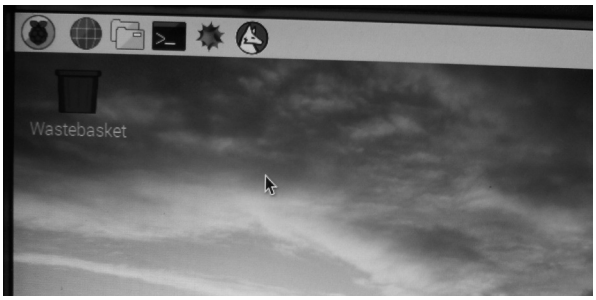

Figure A.11 Screen after the restart

A.5 Setting Up the WiFi and Remote Access

It is very likely that you will want to access your RPi WZ remotely from your desktop or laptop computer. The easiest option here is to enable WiFi on your RPi WZ and then access it from your computer using the SSH client protocol. This protocol requires a server and a client. The server is your RPi WZ and the client is your desktop or laptop computer. In this section we will see how to enable the WiFi on your RPi WZ and how to access it remotely.

Setting Up WiFi

To enable the WiFi on your RPi WZ, the steps are as follows:

- Click on the WiFi icon which is a pair of red crosses at the top right hand side of the screen
- Select your WiFi router from the displayed list (see Figure A.12)

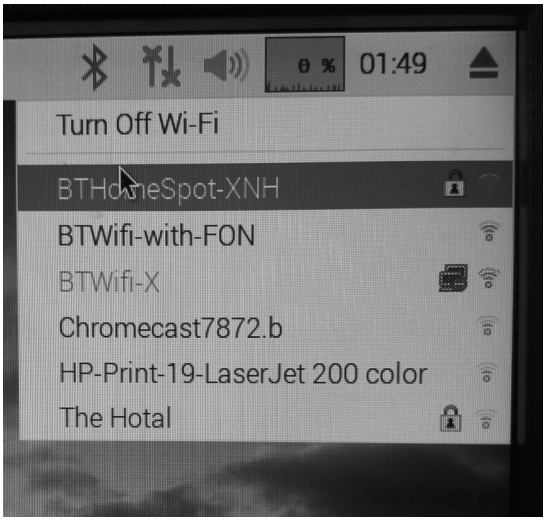

Figure A.12 Select your WiFi from the list

- Enter the password for your WiFi router

- The WiFi icon should become a typical WiFi image. If you click on the icon now you should see a green tick next to the selected router as shown in Figure A.13.

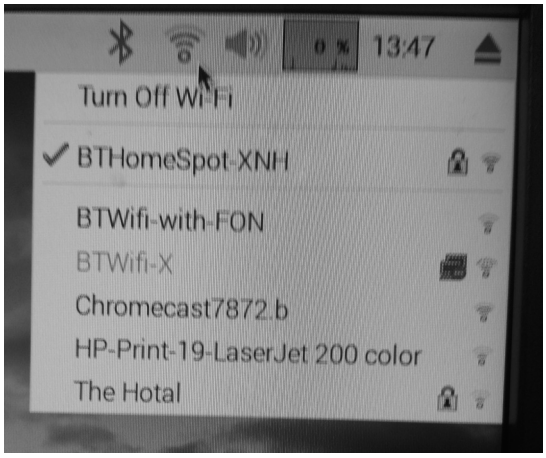

Figure A.13 Connected to the WiFi successfully

- To see the IP address of your WiFi connection, place the mouse over the WiFi icon as shown in Figure A.14. In this example the IP address was 192.168.1.84

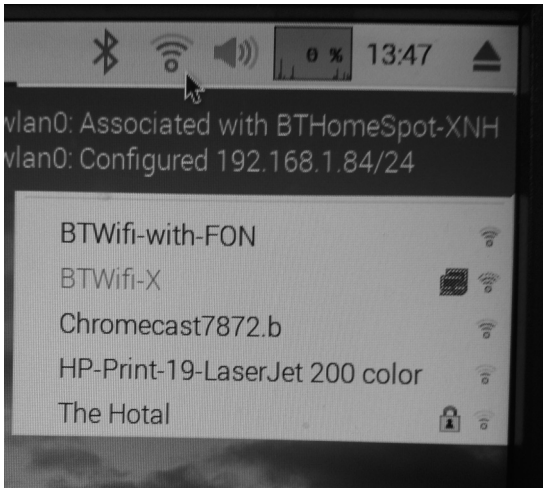

Figure A.14 IP address of our connection

Remote Access

The program we will be using to access our RPi WZ is called **Putty** with the SSH protocol. The steps to download and use Putty are as follows:

- Download Putty from the following link (or search Google for “Download Putty”)

http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

- For security reasons the SSH protocol is disabled by default on a new operating system. To enable it, click on the **Applications** menu at the top left of the screen, click **Accessories**, and then click **Terminal** (see Figure A.15)

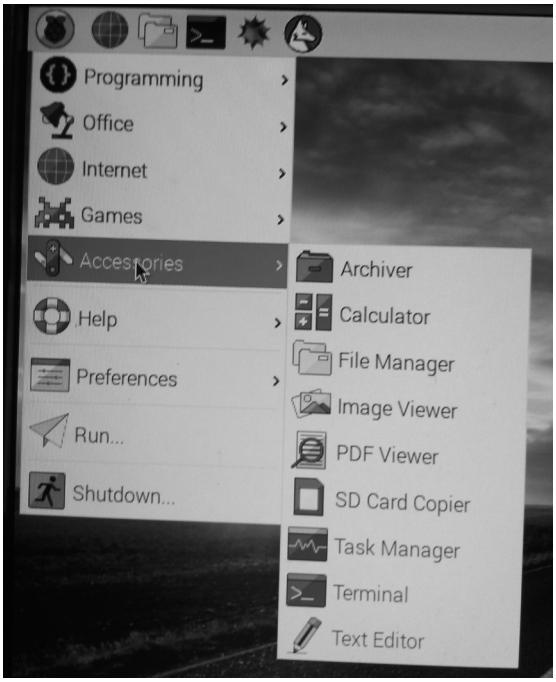

Figure A.15 Access the Terminal menu

- You should now be in the RPi WZ command prompt. Type:

  sudo raspi-config

to go into the configuration menu and select **Interface Options**. Go down to **P2 SSH** and enable SSH as shown in Figure A.16

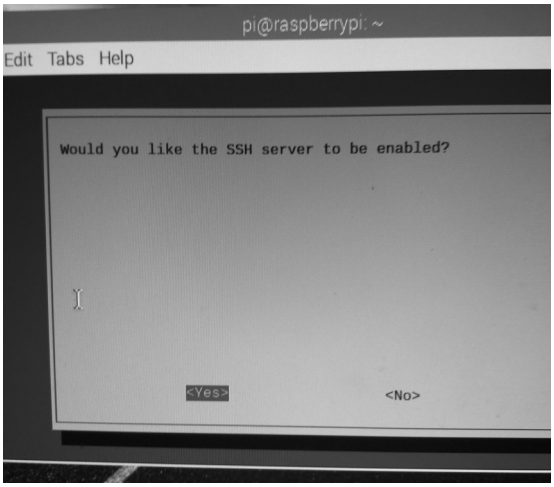

Figure A.16 Enable the SSH server

- Click <Finish> to exit the configuration menu. You should now be back in the command mode, identified by the prompt:

  pi@raspberrypi:~ $

- Putty is a standalone program and there is no need to install it. Simply double click to run it. You should see the Putty startup screen as in Figure A.17

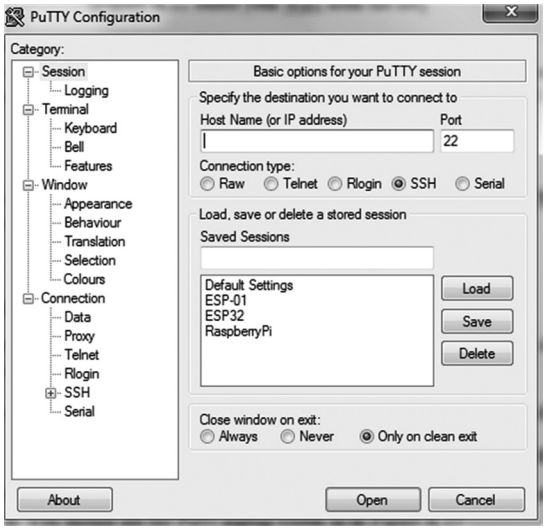

Figure A.17 Putty startup screen

- Make sure that the Connection type is SSH and enter the IP address of your RPi WZ. Click Open as shown in Figure A.18)

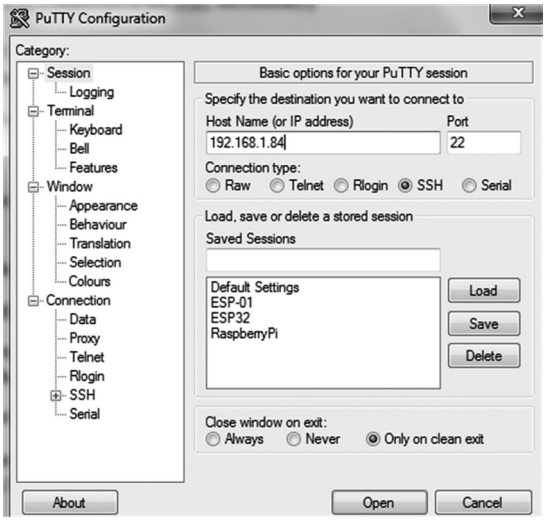

Figure A.18 Enter the IP address

- The message shown in Figure A.19 will be displayed on the PC screen the first time you access the RPi WZ. Click Yes to accept this security alert.

Figure A.19 Click Yes to accept

- You will then be prompted for the username and password. The default values are:

  Username: **pi**
  Password: **raspberry**

- After a successful login you should see the RPi ZW command prompt a sin Figure A.20.

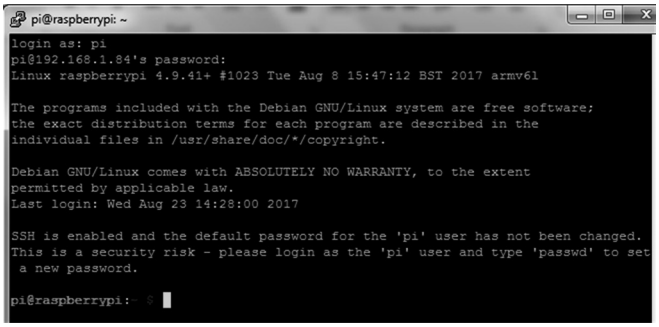

Figure A.20 Successful login

- To change your password, enter the following command:

  passwd

- To restart the RPi ZW enter the following command:

  sudo reboot

- To shutdown the RPi ZW enter the following command. Never shutdown by pulling the power cable as this may result in the corruption or loss of files:

  sudo shutdown –h now

A.6 Shutting Down or Rebooting in GUI Mode

You must always shutdown your RPi ZW properly. To shut down while in the GUI mode, follow the steps below:

- Click Applications menu (top left corner)
- Click Shutdown (see Figure A.21)
- Click Shutdown (or Reboot as required)

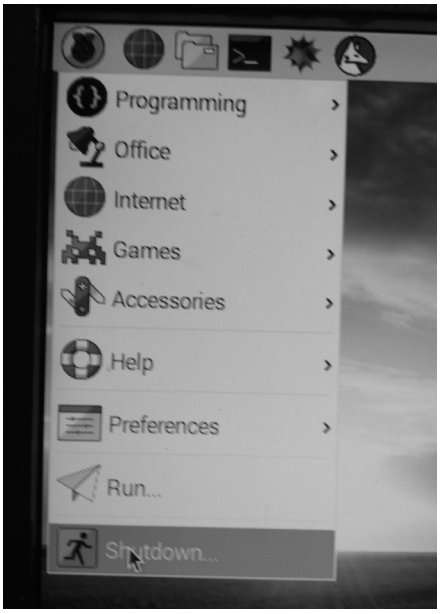

Figure A.21 Shutdown or reboot in GUI mode

A.7 Remote Access of the Desktop

If you will be using your RPi ZW with local keyboard, mouse, and monitor then you can skip this section. If on the other hand you want to access your Desktop remotely over the network, you will find that SSH services cannot be used. The easiest and simplest way to access your Desktop remotely from a computer is by installing the VNC (Virtual Network Connection) client and server. The VNC server runs on your Pi and the VNC client runs on your computer. The steps to install and use the VNC are given below:

- Connect to your RPi ZW using SSH as explained earlier. Then enter the following command to install a program called TightVNC server on your RPi ZW. You will see many lines of messages. Make sure there are no error messages:

  sudo apt-get update
  sudo apt-get install tightvncserver

- Run the VNC server on your RPi ZW by entering the following command:

  vncserver :1

- You will be prompted to enter and verify a password. This will be the password you will be using to access the Desktop remotely (see Figure A.22).

```
update-alternatives: using /usr/bin/tightvncpasswd to provid
 (vncpasswd) in auto mode
Processing triggers for man-db (2.7.6.1-2) ...

Processing triggers for shared-mime-info (1.8-1) ...
Processing triggers for gnome-menus (3.13.3-9) ...
Processing triggers for hicolor-icon-theme (0.15-1) ...
Setting up xfonts-base (1:1.0.4+nmu1) ...
Processing triggers for fontconfig (2.11.0-6.7) ...
pi@raspberrypi:~ $ vncserver :1

You will require a password to access your desktops.

Password:
Verify:
Would you like to enter a view-only password (y/n)? n

New 'X' desktop is raspberrypi:1

Creating default startup script /home/pi/.vnc/xstartup
Starting applications specified in /home/pi/.vnc/xstartup
Log file is /home/pi/.vnc/raspberrypi:1.log

pi@raspberrypi:~ $
```

Figure A.22 Enter a password for the VNC server

- The VNC server is now running on your RPi ZW. The only command you need to enter on your RPi ZW to start the VNC server is:

  vncserver :1

- We must now setup a VNC client on our laptop (or desktop). There are many VNC clients available, but the recommended one which is compatible with TightVNC is the VNCViewer, which can be downloaded from the following link. Note that this program is not free of charge, but a 30 day free trial version is available. You should register to get a trial license and then apply this license to the software to use free of charge for 30 days:

  http://www.realvnc.com

  - Download the VNCviewer program into a suitable directory on your computer.
  - Double click to install it and enter the required license. Start the **VNCViewer** program by double clicking its icon in your desktop. Enter the IP address of your RPi ZW followed by :1 as shown in Figure A.23 and click Connect.

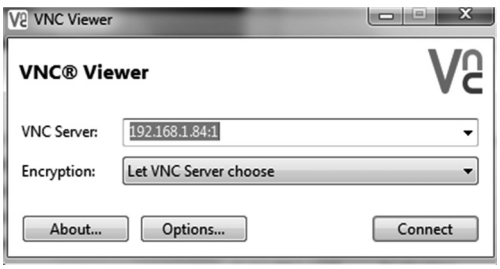

Figure A.23 Enter the IP address

- Enter the password selected previously. You should now see the RPi ZW Desktop displayed on your laptop (or desktop) computer as in Figure A.24 and you can access all of the Desktop applications remotely.

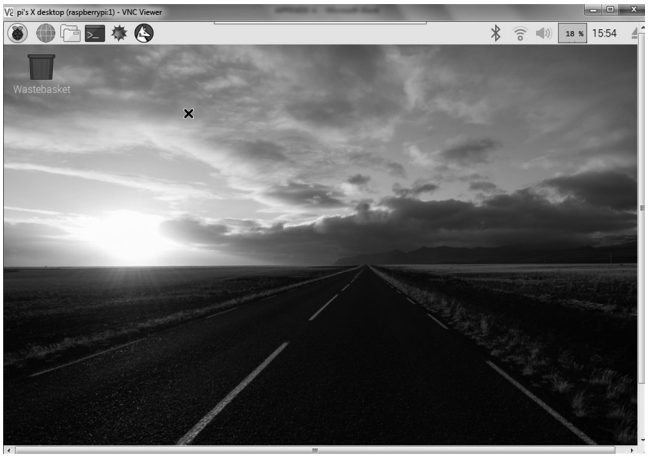

Figure A.24 RPi ZW Desktop displayed on the laptop

A.8 Enabling Bluetooth

In this section we will see how to enable the Bluetooth on your RPi ZW so that it can communicate with your mobile phone. The steps are given below:

- Enable the Bluetooth on your mobile device
- Click on the Bluetooth icon on your RPi ZW at the top right hand side, and select **Make Discoverable.** You should see the Bluetooth icon flashing
- Select raspberrypi in the Bluetooth menu on your mobile device
- Accept the pairing request on your RPi ZW as shown in Figure A.25

Figure A.25 Bluetooth pairing request

- You should now see the message **Connected Successfully** on your RPi ZW and you can exchange files between your mobile device and the RPi ZW.

A.9 Creating and Running a Python Program

We will be programming our RPi ZW using the Python programming language. It is worthwhile to look at the creation and running of a simple Python program on our RPi ZW. In this section we will display the message **Hello From Raspberry Pi Zero W** on our PC screen. As described below, there are 3 methods to create and run programs on our RPi ZW

Method 1 – Interactively from Command Prompt

In this method, we will login to our RPi ZW using the SSH and then create and run our program interactively. This method is excellent for small programs. The steps are as follows:

- Login to the RPi ZW using SSH
- At the command prompt enter **python**. You should see the Python command mode which is identified by three characters **>>>**
- Type the program:

  Print ("Hello From Raspberry Pi Zero W")

- The required text will be displayed interactively on the screen as shown in Figure A.26

```
pi@raspberrypi:~ $ python
Python 2.7.13 (default, Jan 19 2017, 14:48:08)
[GCC 6.3.0 20170124] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("Hello From Raspberry Pi Zero W")
Hello From Raspberry Pi Zero W
>>>
```

Figure A.26 Running a program interactively

Method 2 – Create a Python File in Command Mode

In this method, we will login to our RPi ZW using the SSH as before and then create a Python file. A Python file is simply a text file with the extension **.py**. We can use a text editor, e.g. the **nano** text editor to create our file. In this example a file called **hello.py** is created using the **nano** text editor. Figure A.27 shows the contents of file hello.py. This figure also shows how to run the file under Python. Notice that the program is run by entering the command:

>>> python hello.py

```
pi@raspberrypi:~ $ ls hello.py
hello.py
pi@raspberrypi:~ $ cat hello.py
print ("Hello From  Raspberry Pi Zero W")

pi@raspberrypi:~ $ python hello.py
Hello From  Raspberry Pi Zero W
pi@raspberrypi:~ $
```

Figure A.27 Creating and running a Python file

Method 3 – Create a Python File in GUI mode

In this method, we will login to our RPi ZW using the VNC and create and run our program in GUI mode. The steps are given below:

- Click Applications menu
- Click Programming and select Python 2 or Python 3 (see Figure A.28)

Figure A.28 Select Python 2 programming

- You should see the Python command mode, identified by characters >>>
- Click **File** and then click **New File** and write your program
- Save the file by giving it a name (e.g. hello2)
- Run the program by clicking **Run** and then **Run Module** as shown in Figure A.29

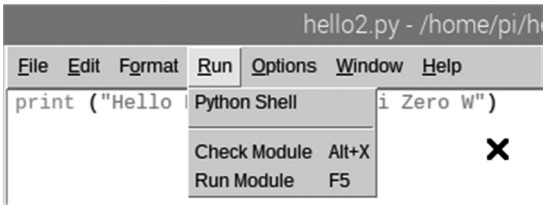

Figure A.29 Run the program

- A new screen will be shown with the output of the program displayed as in Figure A.30

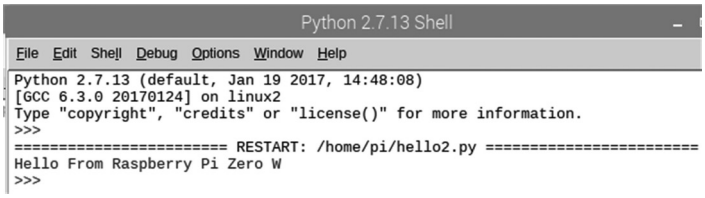

Figure A.30 Output of the program

Which Method ?

The choice of a method depends upon the size and complexity of a program. Small programs can be run interactively without creating a program file. Larger programs can be created as Python files and then they can run either in the command mode or in the GUI mode. In this book, program files are created for all the Python programs.

A.10 GPIO Library

The GPIO library is called RPi.GPIO and it should already be installed on your RPi ZW. This library must be included at the beginning of your Python programs if you will be using the GPIO functions. The statement to include this library is:

```
import RPi.GPIO as GPIO
```

If you get an error while trying to import the GPIO library then it is possible that the library is not installed. Enter the following commands while in the command mode (identified by the prompt **pi@raspberrypi:~ $**) to install the GPIO library (characters that should be entered by you are in bold):

pi@raspberrypi: ~ $ **sudo apt-get update**
pi@raspberrypi: ~$ **sudo apt-get install python-dev**
pi@raspberrypi: ~$ **sudo apt-get install python-rpi.gpio**

The GPIO provides a number of useful functions. The available functions are given in the next sections

A.10.1 Pin Numbering

There are two ways that we can refer to the GPIO pins. The first is using the BOARD numbering, where the pin numbers on the GPIO connector of the RPi ZW are used. Enter the following statement to use the BOARD method:

```
GPIO.setmode(GPIO.BOARD)
```

The second numbering system, also known as the BCM method is the preferred method and it uses the channel numbers allocated to the pins. This method requires you to know which channel number refers to which pin on the board. In this book we shall be using this second method. Enter the following statement to use the BCM method:

```
GPIO.setmode(GPIO.BCM)
```

A.10.2 Channel (I/O port pin) Configuration

Input Configuration
You need to configure the channels (or port pins) you are using whether they are input or output channels. The following statement is used to configure a channel as an input. Here, channel refers to the channel number based on the **setmode** statement above:

```
GPIO.setup(channel, GPIO.IN)
```

When there is nothing connected to an input pin, the data at this input is not defined. We can specify additional parameters with the input configuration statement to connect pull-up or pull-down resistors by software to an input pin. The required statements are:

For pull-down:

```
GPIO.setup(channel, GPIO.IN,  pull_up_down=GPIO.PUD_DOWN)
```

For pull-up:

```
GPIO.setup(channel, GPIO.IN,  pull_up_down=GPIO.PUD_UP)
```

We can detect an edge change of an input signal at an input pin. Edge change is when the signal changes from LOW to HIGH (rising edge), or from HIGH to LOW (falling edge). For example, pressing a push-button switch can cause an edge change at the input of a pin. The following statements can be used to wait for an edge of the input signal. These are blocking functions. i.e. the program will wait until the specified edge is detected at the input signal. For example, if this is a push-button, the program will wait until the button is pressed:

To wait for a rising edge:

```
GPIO.wait_for_edge(channel, GPIO.RISING)
```

To wait for a falling edge:

```
GPIO.wait_for_edge(channel, GPIO.FALLING)
```

We can also wait until either a rising or a falling edge is detected by using the following statement:

```
GPIO.wait_for_edge(channel, GPIO.BOTH)
```

We can use event detection function with an input pin. This way, we can execute the event detection code whenever an event is detected. Events can be rising edge, falling edge, or change in either edge of the signal. Event detection is usually used in loops where we can check for the event while executing other code.

For example, to add rising event detection to an input pin:

```
GPIO.add_event_detect(channel, GPIO.RISING)
```

We can check whether or not the event occurred by the following statement:

```
If GPIO.event_detected(channel):
 ……
 ……
```

Event detection can be removed by the following statement:

```
GPIO.remove_event_detect(channel)
```

We can also use interrupt facilities (or callbacks) to detect events. Here, the event is handled inside a user function. The main program carries on its usual duties and as soon as the event occurs the program stops whatever it is doing and jumps to the event handling function. For example, the following statement can be used to add interrupt based event handling to our programs on rising edge of an input signal. In this example, the event handling code is the function named **MyHandler**:

```
GPIO.add_event_detect(channel, GPIO.RISING, callback=MyHandler)
……
……

def MyHandler(channel):
    ……
    ……
```

We can add more than one interrupt by using the add_event_callback function. Here the callback functions are executed sequentially:

```
GPIO.add_event_detect(channel, GPIO.RISING)
GPIO.add_event_callback(channel, MyHandler1)
GPIO.add_event_callback(channel, MyHandler2)
……
……

def MyHandler1(channel):
    ……
    ……

def MyHandler2(channel):
    ……
    ……
```

When we use mechanical switches in our projects we get what is known as the switch bouncing problem. This occurs as the contacts of the switch bounce many times until they settle to their final state. Switch bouncing could generate several pulses before it settles down. We can avoid switch bouncing problems in hardware or software. GPIO library provides a parameter called bouncetime that can be used to eliminate the switch bouncing problem. An example use of this parameter is shown below where the switch bounce time is assumed to be 10ms:

GPIO.add_event_detect(channel,GPIO=RISING,callback=MyHandler, bouncetime=10)

We can also use the callback statement to specify the switch bouncing time as@

GPIO.add_event_callback(channel, MyHandler, bouncetime=10)

To read the state of an input pin we can use the following statement:

GPIO.input(channel)

Output Configuration
The following statement is used to configure a channel as an output. Here, channel refers to the port number based on the **setmode** statement described earlier:

GPIO.setup(channel, GPIO.OUT)

We can specify a value for an output pin during its setup. For example, we can configure a channel as output and at the same time set its value to logic HIGH (+3.3V):

GPIO.setup(channel, GPIO.OUT, initial=GPIO.HIGH)

To send data to an output port pin we can use the following statement:

GPIO.output(channel, value)

Where value can be 0 (or GPIO.LOW, or False), or 1 (or GPIO.HIGH, or True)

At the end of the program we should return all the used resources to the operating system. This is done by including the following statement at the end of our program:

GPIO.cleanup()

APPENDIX B ● LIST OF COMPONENTS

A list of the components used in the projects is given below:

- 1 x Arduino Uno (including USB cable)
- 1 x Raspberry Pi Zero W (including SD card with OS and power supply)
- 1 x Small breadboard
- 1 x Small DC motor
- 1 x BC108 transistor
- 4 x 1N4001 diode
- 4 x IRL540 MOSFET transistor
- 1 x Small relay
- 1 x MCP3002 IC
- 1 x Button
- 1 x Small buzzer
- 1 x L293 IC
- 1 x LMD18200 IC
- 1 x Small geared DC motor (Pololu) with encoder
- 1 x I2C LCD
- 1 x Ultrasonic sensor module
- 1 x Small Robot assembly (with chassis, wheels, and motors)
- 2 x LDR
- 1 x Bright white LED
- 1 x Small stepper motor (including driver)
- 1 x TMP36DZ
- 1 x Small servo motor (SG90)
- 1 x HC-06 Bluetooth module
- 1 x ESP-01 (including breadboard adapter)
- 1 x 3.3K resistor
- 2 x 15K resistor
- 2 x 1K resistor
- 2 x 2K resistor
- 1 x 10K resistor
- Male-male jumper wires
- Male – female jumper wires

APPENDIX C • ARDUINO UNO PIN DIAGRAM

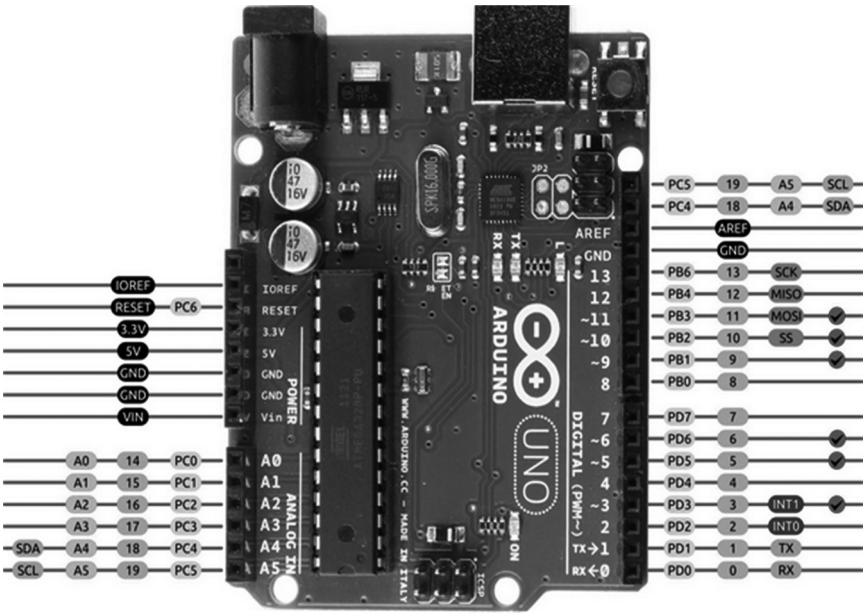

Figure C-1 Arduino Uno pin diagram

APPENDIX D ● RASPBERRY PI ZERO W PIN DIAGRAM

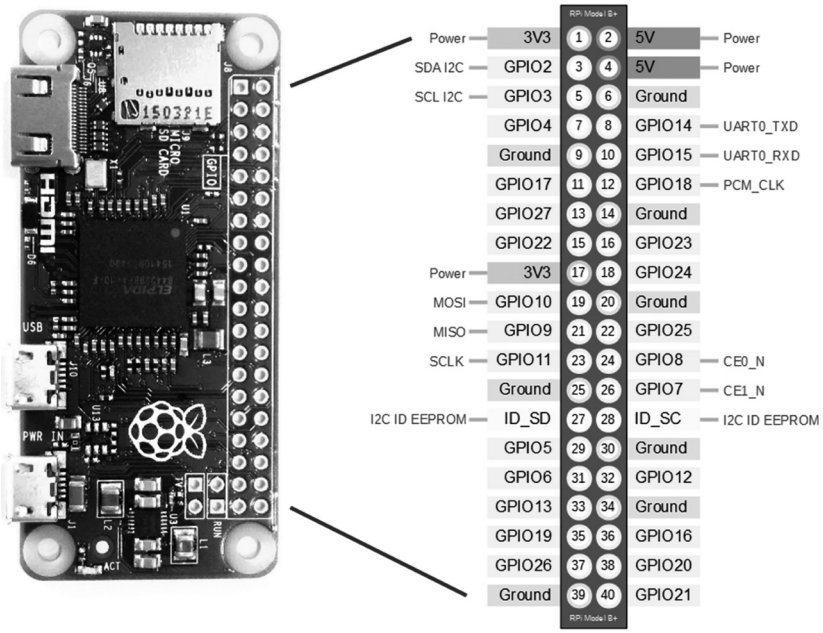

Figure D-1 Raspberry Pi Zero W pin diagram

APPENDIX E • USING THE gpiozero LIBRARY

gpiozero is a Python library that includes many functions that can be used to interface to various peripheral devices such as LEDs, RGB LEDs, LED bar graphs, buttons, line sensors, motion sensors, light sensors, distance sensors, motors, buzzers, ADC, robot control, digital input/output, SPI devices, and many more.

gpiozero functions are very detailed and make the programming a much easier task. Further information about the use of gpiozero and all of its library functions is available at the following web site:

https://gpiozero.readthedocs.io/en/stable/index.html

gpiozero library uses Broadcom (BCM) pin numbering for the GPIO pins as was the case in all the Python programs in this book.

Some examples are given in this Appendix to show how easy it is to develop projects using the gpiozero library functions.

Example A.1
Write a Python program using the gpiozero library to flash an LED connected to port pin GPIO2.

Solution A.1
The program is given below:

```
from gpiozero import LED
import time
led = LED(2)
while true:
     led.on()
     sleep(1)
   led.off()
     sleep(1)
```

We could instead use the **blink** function as follows. The default flashing time is 1 second, but can be changed if required:

```
from gpiozero import LED
from signal import pause
MyLed = LED(2)
MyLed.blink()
while True:
       pass
```

The general format of the **blink** function is:

blink(on_time=1, off_time=1, n=None, background=True)

where,

on_time: number of seconds ON (default is 1 second)

off_time: number of seconds OFF (default is 1 second)

n: number of times to blink (default is forever)

background: If True (default) starts a background thread to continue blinking and return immediately. If False, returns when blink is finished

In addition, we can turn the device ON or OFF, or toggle its state.

Example A.2
A push-button switch is connected to port pin GPIO3 with one of the legs of the button connected to ground. Write a program using the gpiozero library to display the message **Button pressed** when the button is pressed, and display **Button is not pressed** when the button is not pressed.

Solution A.2
The program is given below:

```
from gpiozero import Button
MyButton = Button(2)
while True:
      if MyButton.is_pressed:
            print(“Button pressed”)
      else:
            print(“Button is not pressed”)
```

In addition, the button function can have the following parameters:

pull_up: used to pull-up (default) or pull-down the pin

bounce_time: used to specify number of seconds of debounce time (default is none)

hold_time: time to wait in seconds (default is 1) after the button is pressed until executing the when_held handler

hold_repeat: if True, the when_held handler will be repeatedly executed as long as the device remains active, every hold_time seconds.

Other button options are: wait_for_press, wait_for_release, held_time, hold_time, hold_repeat, and many more.

The reader should refer to the web site given at the beginning of this Appendix for further details on using the gpiozero library.

Index

PROJECTS WITH ARDUINO & RASPBERRY PI

MOTOR CONTROL

Dogan Ibrahim

Prof Dr Dogan Ibrahim is a Fellow of the Institution of Electrical Engineers.

He is the author of over 60 technical books, published by international famous publishers, such as Wiley, Butterworth, and Newnes.

In addition, he is the author of over 250 technical papers, published in journals, and presented in seminars and conferences.

ISBN 978-19-0792-066-0

9 781907 920660

Elektor International Media BV

www.elektor.com

This book is about DC electric motors and their use in Arduino and Raspberry Pi Zero W based projects. The book includes many tested and working projects where each project has the following sub-headings:

- Title of the project
- Description of the project
- Block diagram
- Circuit diagram
- Project assembly
- Complete program listing of the project
- Full description of the program

The projects in the book cover the standard DC motors, stepper motors, servo motors, and mobile robots. The book is aimed at students, hobbyists, and anyone else interested in developing microcontroller based projects using the Arduino Uno or the Raspberry Pi Zero W.

One of the nice features of this book is that it gives complete projects for remote control of a mobile robot from a mobile phone, using the Arduino Uno as well as the Raspberry Pi Zero W development boards. These projects are developed using Wi-Fi as well as the Bluetooth connectivity with the mobile phone. Readers should be able to move a robot forward, reverse, turn left, or turn right by sending simple commands from a mobile phone. Full program listings of all the projects as well as the detailed program descriptions are given in the book. Users should be able to use the projects as they are presented, or modify them to suit their own needs.