Simon Fraser University ENGINEERING COMPETITION 2021

User Manual

Educational Robot Kit

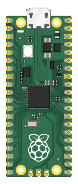
Table of Contents

Introduction

Modern civilization has greatly benefited from the advancements made through technological innovation. A lot of the time, we leave these advancements to the professionals, as even knowing where to start can seem daunting. With this kit as well as a Raspberry Pi, we can introduce and familiarize students from grades K-12 with STEM concepts in a way that is fun and exciting.

This kit will help introduce engineering concepts such as programming and electronics, as well as being modular enough to create your own parts using recyclable materials.

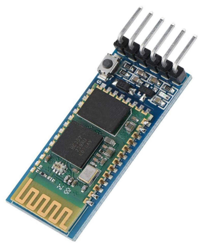
Parts



Raspberry Pi Pico - This microcontroller will be the heart and soul of all our projects. It acts as a computer and can be programmed to do all sorts of things. It can also communicate with electronic parts to do stuff in real time.

DC motor - Converts electrical energy to mechanical energy in the form of rotation

Servo - A motor that can rotate at high precision and can be programmed to rotate to certain positions(not included in the base kit)



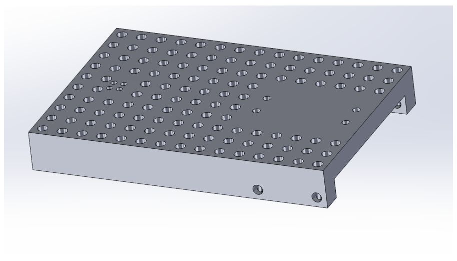
Bluetooth Module - Allows the Raspberry Pi to communicate with a controller wirelessly via bluetooth

LED - Light emitting diode. A type of diode that emits light when electricity passes through it (not included)

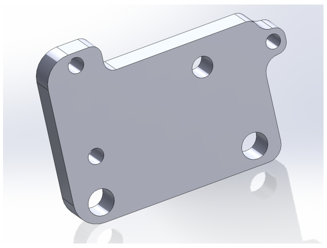
Buzzer - Can be used to generate noise or detect vibrations

Ultrasonic sensor - Uses sound waves to measure the distance to an object (not included in the base kit)

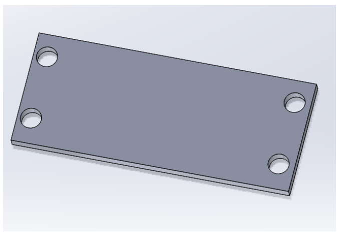
Joystick - Used to control Car (not included in kit, we encourage sourcing third party used Bluetooth controllers)



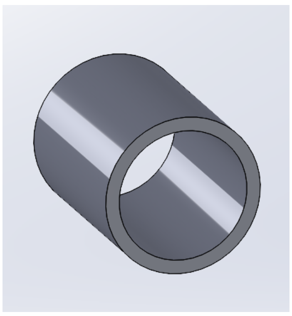
Case



Motor mount



I/O shield



Spacer

Setup

To start using your kit, you must first install the required drivers and IDE in order to program the Raspberry Pi. Visit the following link to get started:
https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico

Sample code to drive forward:

```
digitalWrite(motorPin1, HIGH); // drive left motor forward
digitalWrite(motorPin2, LOW);
digitalWrite(motorPin3, HIGH); // drive right motor forward
digitalWrite(motorPin4, LOW);
delay(2000); // time to run the command (e.g. 2 seconds)
```

Sample code to drive backward:

```
digitalWrite(motorPin1, LOW);
digitalWrite(motorPin2, HIGH); // drive left motor backward
digitalWrite(motorPin3, LOW);
digitalWrite(motorPin4, HIGH); // drive right motor backward
delay(2000); // time to run the command (e.g. 2 seconds)
```

Wide left turn - to achieve a wide left turn, we can keep the left wheel off and the right wheel on:

```
digitalWrite(motorPin1, LOW);
digitalWrite(motorPin2, LOW);
digitalWrite(motorPin3, HIGH); // drive right motor forward
digitalWrite(motorPin4, LOW);
delay(2000); // time to run the command (e.g. 2 seconds)
```

Narrow right turn - to achieve a narrow right turn, we can turn the left wheel forward and the right wheel backward

```
digitalWrite(motorPin1, HIGH); // drive left motor forward
digitalWrite(motorPin2, LOW);
digitalWrite(motorPin3, LOW);
digitalWrite(motorPin4, HIGH); // drive right motor backward
delay(2000); // time to run the command (e.g. 2 seconds)
```

Discover more code, libraries and resources online.

Projects

Project A: Building and driving robot

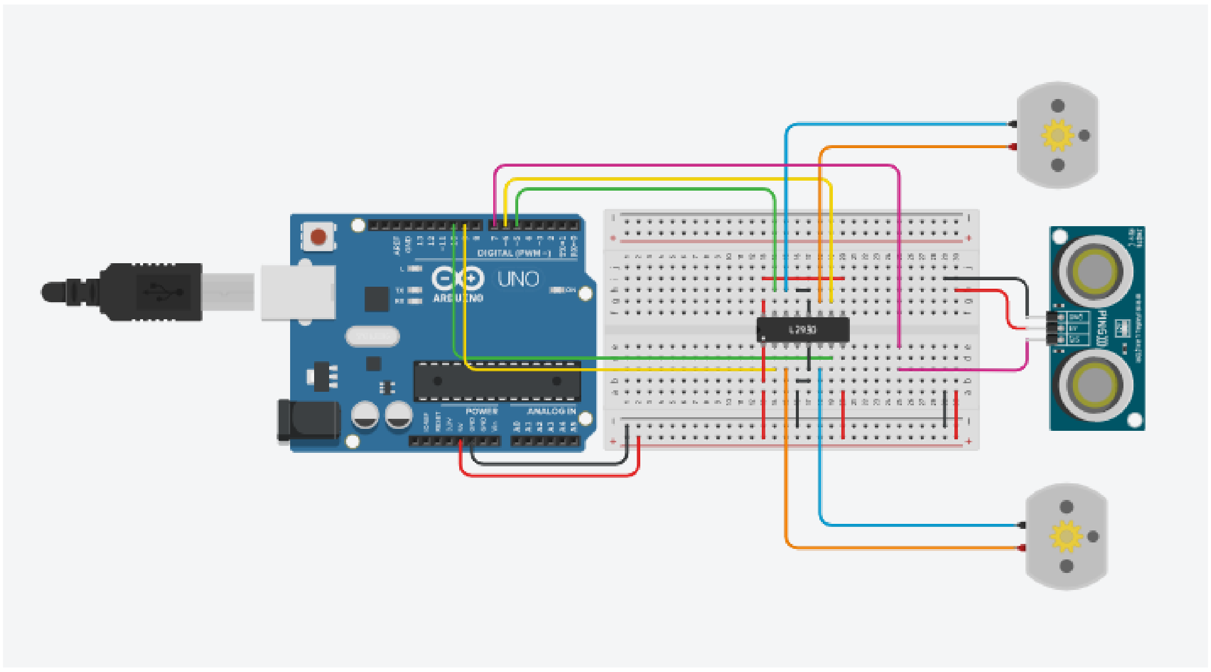
In our very first project, we will build and program a remote control car! Parts we need include:

1. Raspberry Pi Pico
2. Dc motors
3. Case (included in kit or can be 3D printed)
4. Swivelling wheel
5. joystick
6. Bluetooth module (optional)

To assemble our car, we will first attach the Raspberry pi to the case. Then we mount our motors to their designated location, and finally the swivelling wheel to the bottom of the case.

This car can be controlled in three ways:

1. Via bluetooth (fully autonomous)
2. By using uploaded code (Semi-autonomous)
3. Via USB



Project B: Drawing bot

In this project, we will be building on top of our previous project by attaching a pen attachment in order to draw shapes on paper. The type of shape drawn will be dependent on the code you use. Sample code to draw a square is provided below:

```
//L293D
//Motor A
const int motorPin1  = 5;   // Pin 14 of L293
const int motorPin2  = 6;   // Pin 10 of L293
//Motor B
const int motorPin3  = 10; // Pin  7 of L293
const int motorPin4  = 9;   // Pin  2 of L293

const int pingPin = 7;

//This will run only one time.
void setup(){
       Serial.begin(9600);
    //Set pins as outputs
    pinMode(motorPin1, OUTPUT);
    pinMode(motorPin2, OUTPUT);
    pinMode(motorPin3, OUTPUT);
    pinMode(motorPin4, OUTPUT);

}


void loop() {
  //Motor Control - Motor A: motorPin1,motorpin2 & Motor B: motorpin3,motorpin4

    //Forward
    digitalWrite(motorPin1, HIGH);
    digitalWrite(motorPin2, LOW);
    digitalWrite(motorPin3, HIGH);
    digitalWrite(motorPin4, LOW);
    delay(2000);

    //Turn Right
    digitalWrite(motorPin1, LOW);
    digitalWrite(motorPin2, HIGH);
    digitalWrite(motorPin3, HIGH);
    digitalWrite(motorPin4, LOW);
    delay(2000);

    //Stop
    digitalWrite(motorPin1, LOW);
    digitalWrite(motorPin2, LOW);
    digitalWrite(motorPin3, LOW);
    digitalWrite(motorPin4, LOW);
}
```

Project C: Generate PWM

Below is an example of the template and attached documentation:

```
if(digitalRead(ch1_ds)==0&&digitalRead(SC1_B)==1){
            //DC out...
      }else if(digitalRead(ch1_ds)==1&&digitalRead(SC1_B)==1){
            //square wave...
       }

      if(digitalRead(ch1_st)==1&&digitalRead(SC1_B)==0){
            //triangular wave...
      }else if(digitalRead(ch1_st)==0&&digitalRead(SC1_B)==0){
           //sine wave...
      }
```

The variables ch1_ds and ch1_st store the pin numbers assigned to inputs of the rotary and flip switch assigned to motor/actuator 1. Essentially, signals with two different signal processing circuits are constantly output for each channel but using these switches and the logic above, the correct signal is output to the terminals of the power supply. Note that two signals are generated because of the differing signal processing circuits. The sine and triangular waves require a low pass filter to be applied to the PWM signal to demodulate them whereas this is unnecessary for the DC and square signals. It should also be taken into account that if a low pass filter were applied to the square output signal, the output signal would be different(DC) from what is required. In order for the switches to function as required, current has to run through the input circuit that houses the switches. This current is provided by a constant analog DC signal from one of the pins of the Raspberry Pi. The same input circuit is implemented for the other motor/actuator.

Each of these signals required different implementations of PWM as described below:

- for the square wave generator, pulses of non-variable width were created and output to the pin (as shown in the code excerpt below)

    - `Serial.println("Channel 1= Square wave");`
    - `analogWrite(PWM1, (int)(val*(255/Vcc)));`

  the code is setup such that the duty cycle of the signal can be controlled by the user

- for the triangular wave generator, the width of the pulses is increased to a max value and upon reaching this value, the width of the pulses is decreased to a minimum value and then the cycle repeats itself (as shown in the code excerpt below)

  ```
  if(dir==1){
         val = val+0.1;
          analogWrite(PWM, (int)(val*(255/Vcc)));
         if((val+0.1)>max){
    dir=0;
         }
         delay(250);
      }else if(dir==0){
         val = val-0.1;
         analogWrite(PWM, (int)(val*(255/Vcc)));
         if((val-0.1)<min){
    dir=1;
         }
         delay(250);
  ```

  the code is setup such that the max and min values of the signal are user-controllable

- for the sine wave generator, the width of the pulses follows a sinusoidal function that is used to initialise an array which is defined in the setup() function of the Arduino Uno and then called in a loop that calls the analogWrite() function to create the individual pulses (as shown in the code excerpt below)

  ```
   for(int i=0; i<sampling_f2; i++){
        t=(float)i/sampling_f;
        sin_w[i]=
  (int)(127.0*(sin(2*pi*sampling_f*t)+1.0));
    }         //run in void setup()
   for(int i=0; i<sampling_f2; i++){
      //analogWrite(SC1_B, 255);
         analogWrite(PWM, sin_w[i]);
         delay(2);
      } //run in void loop()
  ```

  - the code is setup such that the sampling frequency of the signal are user controllable

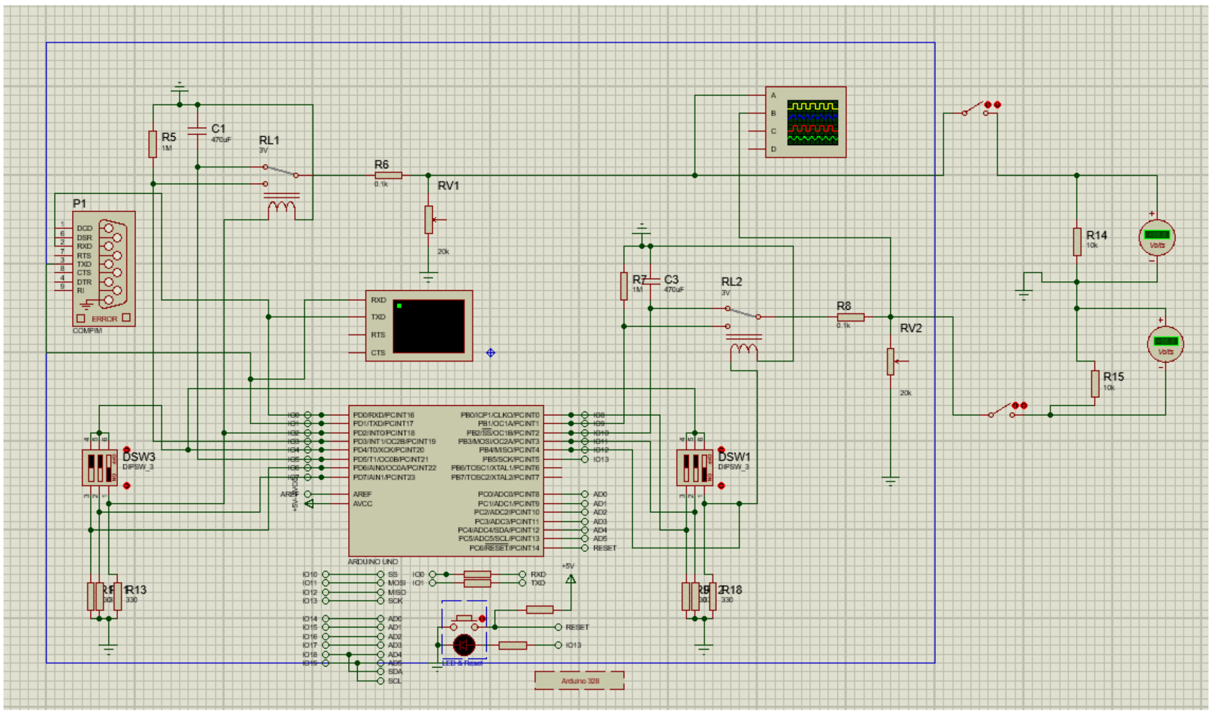
Aside from a hardware interface to control the outputs, a software interface was required. Fortunately, Raspberry Pi has an in-built serial communication protocol that was manipulated and tuned to create a text-based software interface that allows the user to create variables/attributes of the output signal like magnitude and signal type. This was implemented using the code in the following excerpt:

```
 if(digital_input==1){
      Serial.println("What is the desired output type for Actuator
      1?");
      Serial.print("For DC, enter 0, for a square waveform enter 1,
      for a triangular waveform enter 2, for a sine wave enter 3 -
      ");
      delay(1000);
      if(Serial.parseInt()==0){
            digitalWrite(ch1_ds,0);
      }else if(Serial.parseInt()==1){
            digitalWrite(ch1_ds,1);
      }else if(Serial.parseInt()==2){
            digitalWrite(ch1_st,1);
      }else if(Serial.parseInt()==3){
            digitalWrite(ch1_st,0);
      }
             Serial.flush();

      Serial.println("What is the desired output type for Channel
      2?");
      Serial.print("For DC, enter 0, for a square waveform enter 1,
      for a triangular waveform enter 2, for a sine wave enter 3 -
      ");
      if(Serial.parseInt()==0){
            digitalWrite(ch2_ds,0);
      }else if(Serial.parseInt()==1){
             digitalWrite(ch2_st,1);
      }else if(Serial.parseInt()==2){
             digitalWrite(ch2_st,1);
      }else if(Serial.parseInt()==3){
            digitalWrite(ch2_st,0);
      }
}
```

However, this code doesn't work as intended when using a virtual serial port in Proteus. Additional software is required in order to obtain the design effect. The only feature of the Serial communication left unaffected by this minor inconvenience was the system's ability to recognise the output signal type at each channel and report this. The code responsible for this is nested in the appropriate conditional loops of which an example is shown below;

```
if(digitalRead(ch1_ds)==0&&digitalRead(SC1_B)==1){
    val = 5;
    Serial.println("Channel 1= DC");
    analogWrite(PWM1, (int)(val*(255/Vcc)));

}
```

Circuit Schematic of PWM controlled motor/Actuator:



Notes

Due to the modularity of these devices, we highly encourage you to come up with some projects yourself! Ones we think are very good may have an opportunity to be shared on our social media feed!

Appendix

Ultrasonic Sensor

Define the ultrasonic sensor pin.

```
const int pingPin = 7;

//This will run only one time.
void setup(){
       Serial.begin(9600);
       //Set pins as outputs

void loop() {
       long duration, inches, cm;

       pinMode(pingPin, OUTPUT);
       digitalWrite(pingPin, LOW);
       delayMicroseconds(2);
       digitalWrite(pingPin, HIGH);
       delayMicroseconds(5);
       digitalWrite(pingPin, LOW);

       // The same pin is used to read the signal from the PING))): a HIGH pulse
       // whose duration is the time (in microseconds) from the sending of the
       ping
       // to the reception of its echo off of an object.
       pinMode(pingPin, INPUT);
       duration = pulseIn(pingPin, HIGH);

       // convert the time into a distance
       inches = microsecondsToInches(duration);
       cm = microsecondsToCentimeters(duration);

       Serial.print(inches);
       Serial.print("in, ");
       Serial.print(cm);
       Serial.print("cm");
       Serial.println();

       delay(100);
}

long microsecondsToInches(long microseconds) {
       // According to Parallax's datasheet for the PING))), there are 73.746
       // microseconds per inch (i.e. sound travels at 1130 feet per second).
       // This gives the distance travelled by the ping, outbound and return,
       // so we divide by 2 to get the distance of the obstacle.
       // See:
https://www.parallax.com/package/ping-ultrasonic-distance-sensor-downloads/
```

```
	return microseconds / 74 / 2;
}

long microsecondsToCentimeters(long microseconds) {
	// The speed of sound is 340 m/s or 29 microseconds per centimeter.
	// The ping travels out and back, so to find the distance of the object we
	// take half of the distance travelled.
	return microseconds / 29 / 2;
}
```