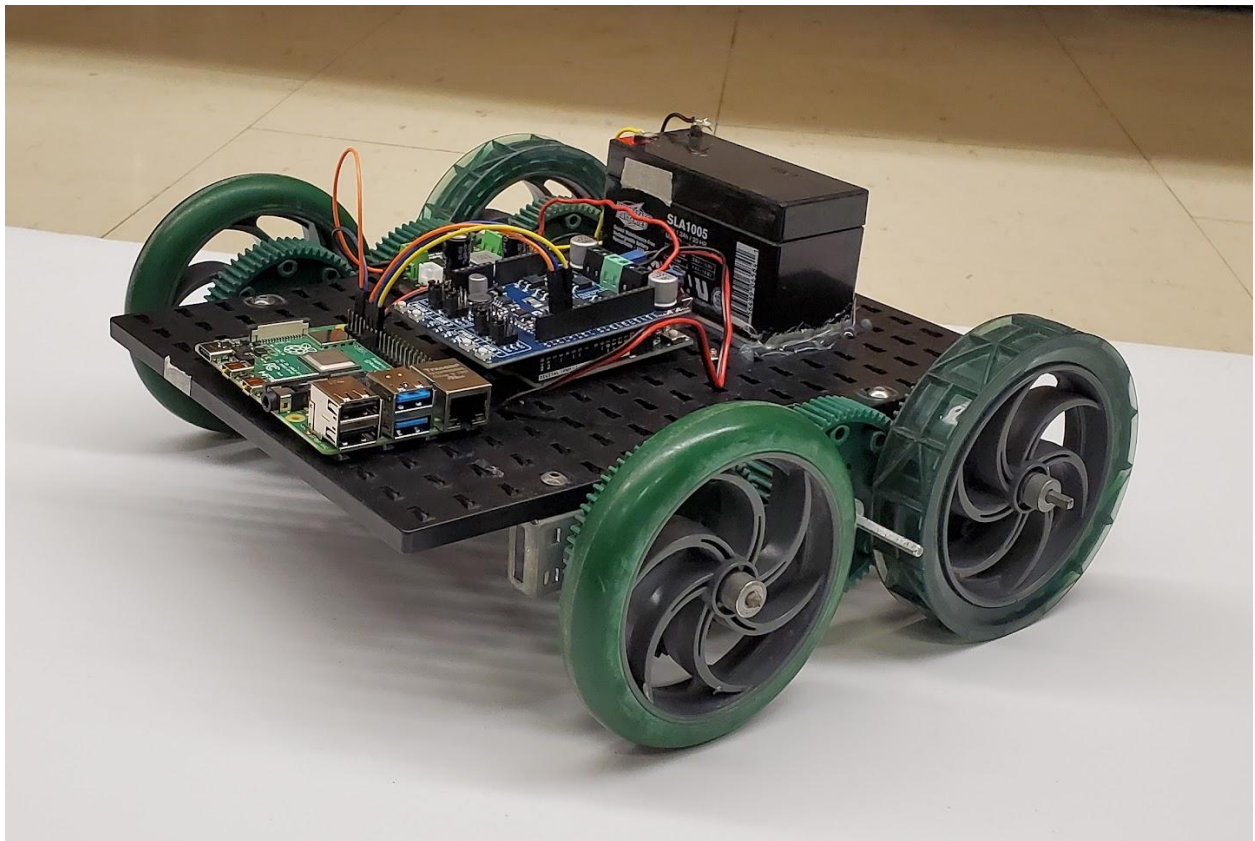


Remote Control Car

Project
Ali Rajan, Sahil Butt



Project Description

The goal of this project is to build a dual-motor, four-wheel drive car that can be controlled remotely over the local network using an Arduino and a Raspberry Pi. The SSH protocol will be used to access the car's Raspberry Pi from another computer, which will allow for remote control. The computer's keyboard will control the car; the Raspberry Pi will process the keyboard inputs and send signals to the Arduino, which in turn will control the motors.

Design Proposal

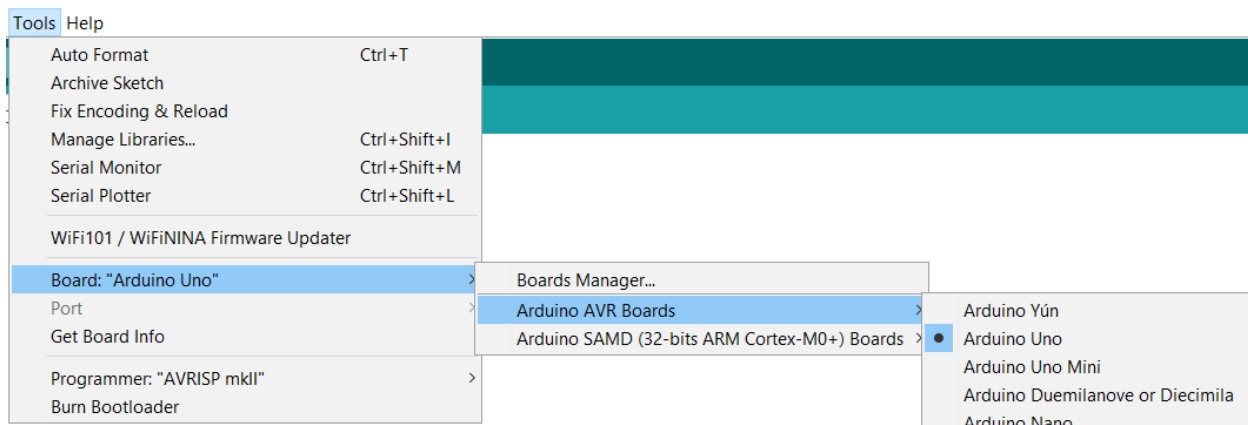
The following components are needed for the design:

- ❖ 12V lead acid battery
- ❖ Switch
- ❖ Wire
- ❖ Male to female jumper wires
- ❖ 5A 350KHz 25V buck DC to DC converter
- ❖ Voltmeter (for troubleshooting)
- ❖ Arduino Uno R3 with its USB cable
- ❖ Shield-MDD10 motor driver shield for Arduino
- ❖ Prebuilt chassis made from:
 - Two 7.2V VEX 2-wire 393 motors
 - VEX robotics 2.75" traction wheel (4pk) (276-1496)
 - VEX robotics 4" wheel (4pk) (276-1497)
- ❖ Raspberry Pi 4 with a compatible micro SD card
- ❖ USB-C to USB-A cable
- ❖ Router
- ❖ Computer with Linux
- ❖ SD card reader
- ❖ Keyboard, mouse, and display (for the Raspberry Pi)

Instructions

Step 1: Setting Up the Arduino IDE

The software required to code the Arduino must be installed on the computer. Download version 1.8 of the Arduino Uno IDE from this link: [UNO R3 | Arduino Documentation](#). Once installed, go to Tools > Board > Arduino AVR Boards and select "Arduino Uno."



Connect the Arduino to the computer and select the port by navigating to Tools > Ports. Choose the port that has “Arduino Uno” in its name (this varies from device to device).

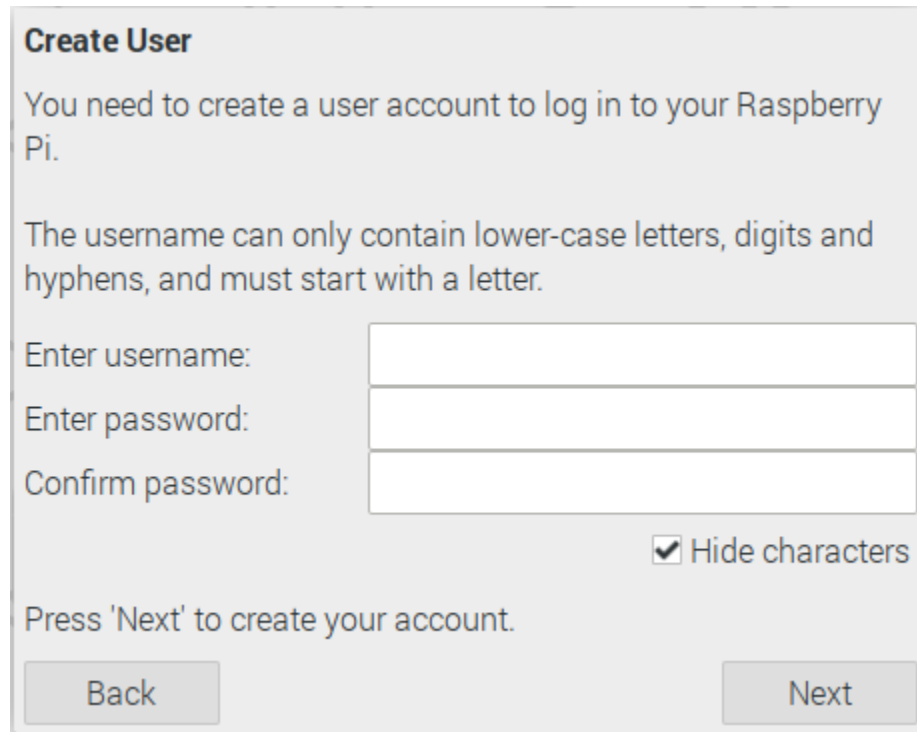
Step 2: Configure the Raspberry Pi

First, the Raspberry Pi operating system must be installed on a micro SD card using a computer. This can be done as follows:

1. Download the Raspberry Pi imager for Ubuntu x86 from this link: [Raspberry Pi OS](#).
2. Go to the terminal and navigate to the “Downloads” folder by typing “cd Downloads” to change directories.
3. Install the imager to the computer by typing in “sudo dpkg -i <name of downloaded file>” and then enter your password.
4. Once installed, connect the SD card to the computer using an SD card reader and run the Raspberry Pi imager.
5. In the imager, select the 32-bit Raspberry Pi operating system, and the connected SD card for the storage.
6. Write and verify to install the operating system on the SD card.



Once this is done, the SD card can be safely ejected. Insert the SD card into the Raspberry Pi. Connect the keyboard, mouse, and display to the Pi, then power it up using the USB-C port. Follow the instructions in the “Welcome to Raspberry Pi” initial setup. When creating a user, be sure to note the username and password as they will be needed later.



Create User

You need to create a user account to log in to your Raspberry Pi.

The username can only contain lower-case letters, digits and hyphens, and must start with a letter.

Enter username:

Enter password:

Confirm password:

☒ Hide characters

Press 'Next' to create your account.

Step 3: Building the Basic Motor Control Circuit

First, the basic motor control circuit (which excludes the Raspberry Pi) must be built. This is done by:

1. Mounting the motor shield onto the Arduino.
2. Connecting the motors to the shield via the motor output terminals. The “forward” direction of each motor is determined by which of the motor’s wires are connected to the positive and negative motor output terminals.
3. Connecting the positive terminal of the battery to the switch, and then connecting the other terminal of that switch to the motor shield (so that the battery, switch, and DC to DC converter are in series).
4. Connecting the constant output from the converter into the shield’s power input terminals. The Arduino is compatible with the 12V supplied by the battery.

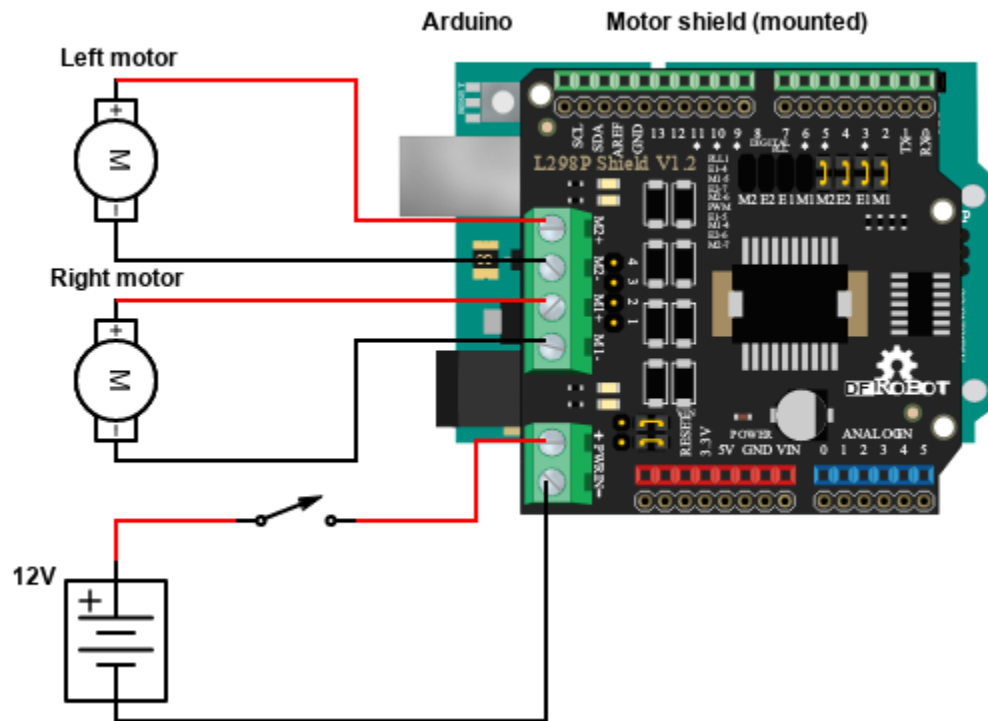


Diagram 1: the basic motor control circuit.

Step 4: Code the Arduino

Once the development environment is set up for the Arduino, the Arduino code for the motor control can be written. Each motor has a PWM and DIR pin, which control the speed and direction of the motor (respectively). These pins must be set as outputs in the code. The motor spins according to the following truth table for the values of PWM and DIR:

PWM	DIR	Result
LOW	Don't care	Motor does not spin
HIGH	LOW	Motor spins forward
HIGH	HIGH	Motor spins backward

Table 1: truth table for the motor's spin direction.

Note that the motor's "forward" direction depends on the connection with the positive and negative motor output terminals. Also, an analog output to the PWM pin can be used to control the motor's speed.

The Arduino takes in three inputs (which will eventually be sent from the Raspberry Pi), one activation input, and one input per motor. The activation input is similar to a power switch for the motors; when the activation signal is LOW, the motors do not spin, regardless of the values of motor 1 and 2's inputs. When the activation signal is HIGH, the motors spin in different directions depending on their respective inputs. The car changes states as follows:

Activation	Motor 1 Input	Motor 2 Input	Motor 1 Spin Direction	Motor 2 Spin Direction	Resulting State
HIGH	HIGH	HIGH	Forward	Forward	Car moves forward
HIGH	LOW	LOW	Backward	Backward	Car moves backward
HIGH	LOW	HIGH	Backward	Forward	Car turns left
HIGH	HIGH	LOW	Forward	Backward	Car turns right
LOW	Don't care	Don't care	Stop	Stop	Car stops

Table 2: truth table for the car's state changes.

In summary, there is one speed pin and one direction pin per motor. There are three inputs, one activation input, and one input per motor. Additionally, the shield has an enable pin which must receive a HIGH signal at all times in order to be functional. The pins were assigned as follows:

Purpose	Pin Number
Motor 1 direction control	8
Motor 1 speed control	9
Motor 2 direction control	13
Motor 2 speed control	11
Motor shield enable pin	6
Motor 1 input	2
Motor 2 input	3
Activation input	4

Table 3: Arduino pin numbers used in the example code.

The code for pin assignment is as follows. Note that a HIGH signal is also outputted to the shield's enable pin in the code.

```
1 // The pins for the motor on the right side of the car
2 int motor1Dir = 8;
3 int motor1Speed = 9;
4 // The pins for the motor on the left side of the car
5 int motor2Dir = 13;
6 int motor2Speed = 11;
7
8 int enablePin = 6; // For the motor shield
9 // Pins for inputs from the Raspberry Pi
10 int motor1InputPin = 2;
11 int motor2InputPin = 3;
12 int activationPin = 4; // The motors spin when the input from this pin is HIGH
13
14 void setup()
15 {
16     pinMode(motor1Dir, OUTPUT);
17     pinMode(motor1Speed, OUTPUT);
18     pinMode(motor2Dir, OUTPUT);
19     pinMode(motor2Speed, OUTPUT);
20
21     pinMode(enablePin, OUTPUT);
22     digitalWrite(enablePin, HIGH);
23     pinMode(motor1InputPin, INPUT);
24     pinMode(motor2InputPin, INPUT);
25     pinMode(activationPin, INPUT);
26 }
```

A separate function can be written for each of the car's states using digitalWrite and analogWrite calls. These functions can have the motor speed as a parameter. Using the information in table 2, this is implemented as follows:

```
32 /**
33  * Makes the motors spin forward.
34  * @param speed The speed of the motors, an integer in the range 0 to 255 (inclusive).
35  */
36 void moveForward(int speed = 100)
37 {
38     digitalWrite(motor1Dir, HIGH);
39     analogWrite(motor1Speed, speed);
40     digitalWrite(motor2Dir, HIGH);
41     analogWrite(motor2Speed, speed);
42 }
```

```
44 /**
45  * Makes the motors spin backward.
46  * @param speed The speed of the motors, an integer in the range 0 to 255 (inclusive).
47  */
48 void moveBackward(int speed = 100)
49 {
50     digitalWrite(motor1Dir, LOW);
51     analogWrite(motor1Speed, speed);
52     digitalWrite(motor2Dir, LOW);
53     analogWrite(motor2Speed, speed);
54 }
55
56 /**
57  * Makes the car turn left.
58  * @param speed The speed of the motors, an integer in the range 0 to 255 (inclusive).
59  */
60 void turnLeft(int speed = 100)
61 {
62     // Make the right motor spin forward
63     digitalWrite(motor1Dir, HIGH);
64     analogWrite(motor1Speed, speed);
65     // Make the left motor spin backward
66     digitalWrite(motor2Dir, LOW);
67     analogWrite(motor2Speed, speed);
68 }
69
70 /**
71  * Makes the car turn right.
72  * @param speed The speed of the motors, an integer in the range 0 to 255 (inclusive).
73  */
74 void turnRight(int speed = 100)
75 {
76     // Make the right motor spin backward
77     digitalWrite(motor1Dir, LOW);
78     analogWrite(motor1Speed, speed);
79     // Make the left motor spin forward
80     digitalWrite(motor2Dir, HIGH);
81     analogWrite(motor2Speed, speed);
82 }
```

Note that to make the car stop, either of the `moveForward` and `moveBackward` functions can be called with a speed argument of 0. At this point, the motors can be tested by calling the movement functions inside the program's loop.

For the input processing, a separate function can be written that takes in each of the three input values as parameters and executes the function for the resulting state (as outlined in table 2). This is done as follows:


```
84 /**
85  * Handles the inputs from the Raspberry Pi to remote control the motors.
86  * @param motor1Input The input for motor 1.
87  * @param motor2Input The input for motor 2.
88  * @param activate If this is LOW, the motors are stopped.
89  */
90 void remoteControl(int motor1Input, int motor2Input, int activate)
91 {
92     if (activate == LOW)
93     {
94         moveForward(0);    // Makes the motors stop spinning
95     }
96     // Each permutation of motor1Input and motor2Input results in a different state
97     else if (motor1Input == HIGH && motor2Input == HIGH)
98     {
99         moveForward();
100    }
101    else if (motor1Input == LOW && motor2Input == LOW)
102    {
103        moveBackward();
104    }
105    else if (motor1Input == HIGH && motor2Input == LOW)
106    {
107        turnLeft();
108    }
109    else if (motor1Input == LOW && motor2Input == HIGH)
110    {
111        turnRight();
112    }
113 }
```

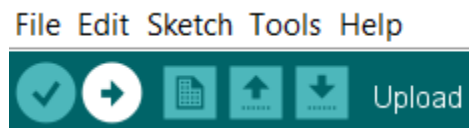
Finally, the loop function must be written to process new inputs in fixed intervals of time. First, new variables are declared for the input values from each of the three input pins and the time elapsed since the last time input was processed. These should be global variables, just like the variables for the pin numbers.

```
13 int motor1Input;
14 int motor2Input;
15 int activate;
16 unsigned long prevInputTime = 0;    // The time that input was previously processed
```

The input can be read from the pins using `digitalRead`, and the function to process the input can be called with these input values as arguments. The `millis` function can be used to determine the time elapsed since the last time input was processed, so the loop function can be written as follows:

```
115 void loop()
116 {
117     if (millis() - prevInputTime >= 25)    // Process new inputs every 25 milliseconds
118     {
119         // Reading the inputs from the pins
120         motor1Input = digitalRead(motor1InputPin);
121         motor2Input = digitalRead(motor2InputPin);
122         activate = digitalRead(activationPin);
123         // Changing the state of the motors based on the inputs
124         remoteControl(motor1Input, motor2Input, activate);
125         prevInputTime = millis();    // Updating the previous input processing time
126     }
127 }
```

Once the code is written, connect the Arduino to the computer via USB and upload the code by pressing the “Upload” button in the top-left corner of the Arduino IDE.



Step 5: Code the Raspberry Pi

The Raspberry Pi can be used as a computer when connected to a keyboard, mouse, and display. It can be powered from its USB-C port. It comes preinstalled with Python and the RPi.GPIO Python library, which is used to send signals to the general-purpose input/output (GPIO) pins. To implement keyboard control when connecting from another computer via SSH, the Python library “sshkeyboard” can be used. The library can be installed by opening the Raspberry Pi’s terminal and typing “pip install sshkeyboard”. Note that internet connection is required to download the library, which can be accessed using the Pi’s ethernet port.

Once the library is installed, the code that processes keyboard inputs and sends signals to the Arduino must be written. While the Pi comes preinstalled with the Thonny Python IDE, any IDE can be used. First, create a Python file and import the RPi.GPIO and sshkeyboard libraries as follows:

```
1  import RPi.GPIO as GPIO
2  import sshkeyboard
```

Next, three output pins must be assigned (according to table 2); one output pin per motor, and one activation pin. In this example, pins 17 and 27 are the outputs for motors 1 and 2 (respectively), and pin 22 is the activation pin. The code is as follows:

```
4  # Pins controlling the motors
5  motor1Input = 17
6  motor2Input = 27
7  activate = 22  # The motors spin only when the activation pin sends a HIGH signal
8
9  # Setting up the pin modes
10 GPIO.setmode(GPIO.BCM)
11 GPIO.setup(motor1Input, GPIO.OUT)
12 GPIO.setup(motor2Input, GPIO.OUT)
13 GPIO.setup(activate, GPIO.OUT)
```

Separate functions can be written for each of the car's states (as outlined in table 2). Note that motor 1 and 2's inputs are inputs for the Arduino but outputs for the Raspberry Pi. The code is as follows:

```
15  # Makes the car move forward
16  def moveForward():
17      GPIO.output(motor1Input, GPIO.HIGH)
18      GPIO.output(motor2Input, GPIO.HIGH)
19      GPIO.output(activate, GPIO.HIGH)
20
21  # Makes the car move backwards
22  def moveBackward():
23      GPIO.output(motor1Input, GPIO.LOW)
24      GPIO.output(motor2Input, GPIO.LOW)
25      GPIO.output(activate, GPIO.HIGH)
26
27  # Makes the car turn left
28  def turnLeft():
29      GPIO.output(motor1Input, GPIO.HIGH)
30      GPIO.output(motor2Input, GPIO.LOW)
31      GPIO.output(activate, GPIO.HIGH)
32
33  # Makes the car turn right
34  def turnRight():
35      GPIO.output(motor1Input, GPIO.LOW)
36      GPIO.output(motor2Input, GPIO.HIGH)
37      GPIO.output(activate, GPIO.HIGH)
38
39  # Makes the car stop
40  def stop():
41      GPIO.output(motor1Input, GPIO.LOW)
42      GPIO.output(motor2Input, GPIO.LOW)
43      GPIO.output(activate, GPIO.LOW)
```

Note that for each state, the values of motor 1 and 2's inputs correspond to those in the Arduino's code. Refer to table 2 for the state represented by each permutation of motor 1 and 2's inputs.

Now, the keyboard control must be implemented. The sshkeyboard library has the function "listen_keyboard" that listens for keyboard events and executes the given function "on_press" upon key presses. The on_press function must take in the pressed key as a parameter, which is represented by a string. A function can be written for on_press, which changes the state of the car depending on the key pressed. In this example, the car moves forward when the up arrow is pressed, backward when the down arrow is pressed, turns left when the left arrow is pressed, and turns right when the right arrow is pressed. The space bar stops the car. The car stays in the same state until a key that changes its state is pressed. The function for on_press can be coded as follows:

```
45 # Handles the key inputs when connecting to the Raspberry Pi via SSH
46 def keyboardHandler(key):
47     print("Pressed", key)
48     # Changing the state of the motors according to the key pressed
49     if key == "space":
50         stop()
51     elif key == "up":
52         moveForward()
53     elif key == "down":
54         moveBackward()
55     elif key == "left":
56         turnLeft()
57     elif key == "right":
58         turnRight()
```

Unlike a typical Raspberry Pi Python program, an infinite loop is not required when using listen_keyboard from sshkeyboard. The keyboard listener only needs to be assigned the on_press function to work properly, which is done as follows:

```
60 # Assigning the function that will handle key presses when connecting via SSH
61 sshkeyboard.listen_keyboard(on_press=keyboardHandler)
```

After writing the code, save the Python file. Note the directory and file name as it will be needed later. The file path is "/home/tej4m1-6a/car.py" in this example.

Step 6: Set Up a Wireless Network and Configure SSH

In order to connect to the Raspberry Pi via SSH, a separate wireless network should be set up. It is not recommended to connect the Raspberry Pi to a router that is connected to the internet. Set

up a password-protected wireless network with WPA to ensure compatibility with the Pi (the process varies depending on the router used). In this example, the D-Link DIR-625 was used. The network was created using the setup wizard on the router's page.

Connect the Raspberry Pi to this network through the Wi-Fi icon in the menu bar. Also, connect the computer that will be used for SSH to the same network, either wirelessly or wired. To configure SSH on the Raspberry Pi, go to Preferences > Raspberry Pi Configuration > Interfaces on the Pi and enable SSH.

For ease of access, a static IP address should be assigned to the Pi. This can be done by right-clicking on the Wi-Fi icon in the menu bar and clicking on "Wireless & Wired Network Settings" and selecting "wlan0" in the interfaces. Enter the desired IP address in "IPv4 Address" and then click "Save." Be sure to note this address as it will be used when connecting to the Raspberry Pi via SSH. The IP address 192.168.0.2/24 was used in this example. A reboot is required for the changes to take effect.

Step 7: Modify the Circuit to Incorporate the Raspberry Pi

Now that the Raspberry Pi has been coded, it must be connected to the circuit to operate in headless mode. Since the Pi is sensitive to voltage fluctuations, a DC to DC converter should be used for its voltage input. Modify the existing circuit to incorporate the Raspberry Pi as follows:

1. Remove the connection between the positive and negative terminals of the battery and the Arduino (and remove the switch).
2. Connect the positive terminal of the battery to the switch, and the other terminal of the switch to the DC to DC converter's positive terminal (so that the battery, switch, and converter are in series). Connect the converter's ground to the battery's negative terminal.
3. On the DC to DC converter, set the output mode to 5V (ensure that it is not set to have variable voltage). Using the male-to-female jumper cables, connect one of the Raspberry Pi's 5V pins to the positive terminal of the output and one of the Pi's grounds to the ground of the output.
4. Connect the converter's constant output to the Arduino through its voltage input pin and one of its ground pins.
5. Connect the pins used for the Raspberry Pi's outputs to the pins used for the Arduino's inputs using the male-to-female jumper cables. Ensure that each Arduino pin assigned for input is connected to its corresponding output pin on the Raspberry Pi. In this example, the following pins were assigned on the Raspberry Pi:

Purpose	Raspberry Pi Pin Number	Corresponding Input Pin Number on the Arduino
Motor 1 output	17	2
Motor 2 output	27	3
Activation	22	4

Table 4: Raspberry Pi pin numbers used in the example code.

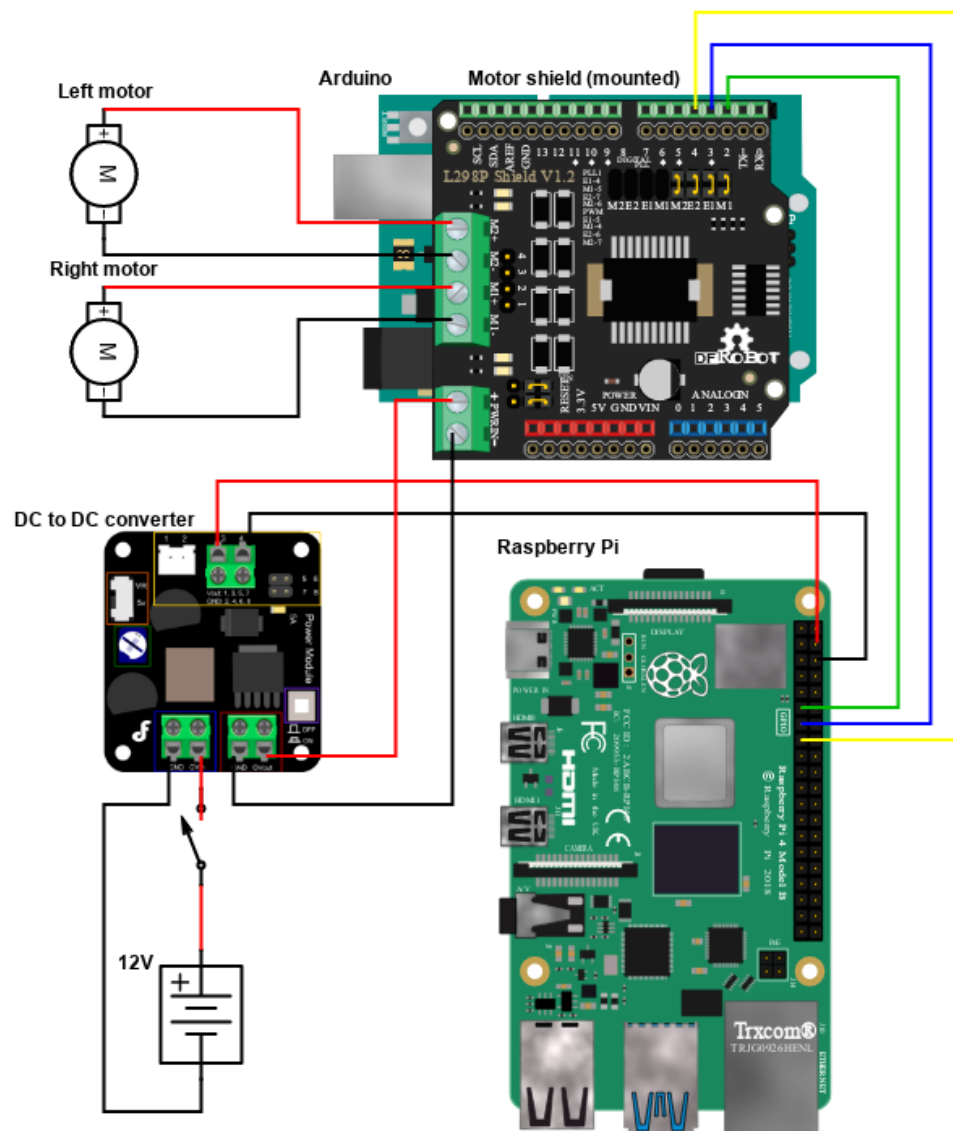


Diagram 2: the complete circuit.

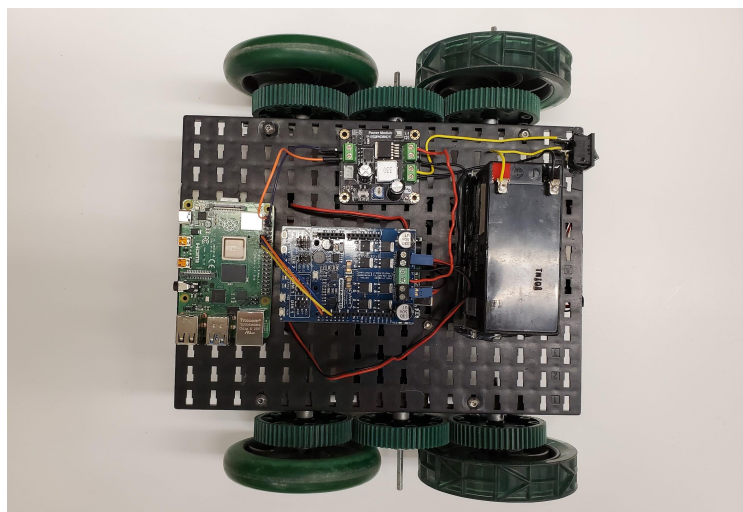
Once the circuit is completely built, its components should be secured onto the chassis using materials such as velcro straps and tape. Optionally, the wires connected to the switch and the battery can be soldered.

Step 8: Control the Car Remotely

To control the car, the computer to be used for remote control must first be connected to the same network as the Raspberry Pi (as in step 6). Once this is done, do the following to connect to the Pi via SSH and run the code:

1. In the computer's terminal, type the command "`ssh <username>@<IP>`" in the computer's terminal, replacing `<username>` with the username of the Raspberry Pi account being used (as noted in step 2) and `<IP>` with the Raspberry Pi's IP address (as noted in step 6). Enter the Raspberry Pi's password (as noted in step 2) when prompted.
2. Execute the Python file written for the Pi by typing "`python <path>`" in the terminal, replacing `<path>` with the path of the Python file. The file path was "`/home/tej4m1-6a/car.py`" in this example, so the command "`python /home/tej4m1-6a/car.py`" would be entered (alternatively, typing "`python car.py`" would achieve the same result since the file was in the home directory).
3. The car can be controlled using the computer's keyboard. The car moves forward when the up arrow is pressed, backward when the down arrow is pressed, turns left when the left arrow is pressed, and turns right when the right arrow is pressed. The space bar stops the car. The car stays in the same state until a key that changes its state is pressed.

The code's execution can be stopped by triggering a keyboard interrupt, which is done by pressing `Ctrl + C`. To exit SSH control, simply type "`exit`" in the terminal.



Potential Issues and Troubleshooting

Issue 1: Components Are Not Powered On When Connected to the Battery

Check the voltage across the battery's terminals. If it is less than 12V, it may be insufficient to power certain components in the circuit, so the battery should either be charged or replaced. This should be one of the first troubleshooting steps when any of the components show signs of receiving low/zero voltage (such as not turning on).

Issue 2: Motors Do Not Spin in the Basic Motor Control Circuit

If the motors do not spin in the basic motor control circuit (from step 3), try the steps in troubleshooting issue 1. If that does not fix the issue, check the voltage across the motor output terminals on the motor shield. If the voltage is 0, check if the correct pins were assigned as outputs in the Arduino code. If this does not fix the issue, the motor shield should be replaced. As a last resort, the Arduino should be replaced if the new motor shield does not fix the issue.

Issue 3: Motors Spin in Opposite Directions (Incorrectly)

If the Arduino code is written so that both motors should spin in the same direction, but the motors spin in opposite directions, the polarity of one motor's supply voltage must be reversed. This can be done by switching the motor's wires connected to the motor shield's output terminals. In other words, the wire initially connected to the positive terminal must be connected to the negative, and wire initially connected to the negative terminal must be connected to the positive.

Issue 4: Raspberry Pi Does Not Boot Up on Battery Power

Check the output voltage from the DC to DC converter. If the converter does not output 5V ($\pm 0.25V$), the Raspberry Pi may not power on. Too low of a voltage is insufficient for the Pi, and too high of a voltage will damage it. First, ensure that the output voltage is not set to variable. If the voltage is too low, try the steps in troubleshooting issue 1. If this does not fix the issue, the motor shield should be replaced. The shield should also be replaced if the voltage is too high.

Issue 5: Keyboard Inputs Do Not Change the Car's State

First, check if the Raspberry Pi's code is working correctly. Do this by temporarily removing the jumper cables for the inputs into the Arduino (but keep them plugged into the Raspberry Pi's GPIO pins). Run the code from the terminal using SSH (as in step 8), and check if pressing each of the arrow keys results in the correct state. Do this by pressing a key and measuring the voltage

in each of the Pi's three output pins relative to the battery's ground. Refer to table 2 to determine if the correct signal is being outputted in the GPIO pins assigned for the Arduino's activation input, motor 1's input, and motor 2's input. If an incorrect signal is being outputted, the issue is most likely in the code; verify that the correct signals are being outputted to the GPIO pins in the code (using table 2). In the unlikely event that the Raspberry Pi is misfiring signals, it should be replaced.

If the correct signals are being outputted to the Raspberry Pi's GPIO pins, check the Arduino's code to verify if it is processing inputs from the Pi correctly. Refer to table 2 to determine what each pin's input value should be for all of the states. A possible issue is that the Arduino is attempting to process new inputs too frequently. In the example code, inputs were processed every 25 milliseconds; this value can be increased to eliminate potential timing errors.

References

[UNO R3 | Arduino Documentation](#)

[Start up your Raspberry Pi](#)

[SHIELD-MDD10 10Amp 7V-30V DC Motor Driver Shield for Arduino \(2 Channels\)](#)

[Raspberry Pi OS](#)

[Arduino Motor Shield Tutorial : 6 Steps \(with Pictures\) - Instructables](#)

[raspberrypi-gpio-python / Wiki / BasicUsage](#)

[sshkeyboard](#)

[Raspberry Pi Documentation - Remote Access](#)

[Set up a static IP-address | The Raspberry Pi Guide](#)

[Raspberry Pi Documentation - GPIO and the 40-pin Header](#)