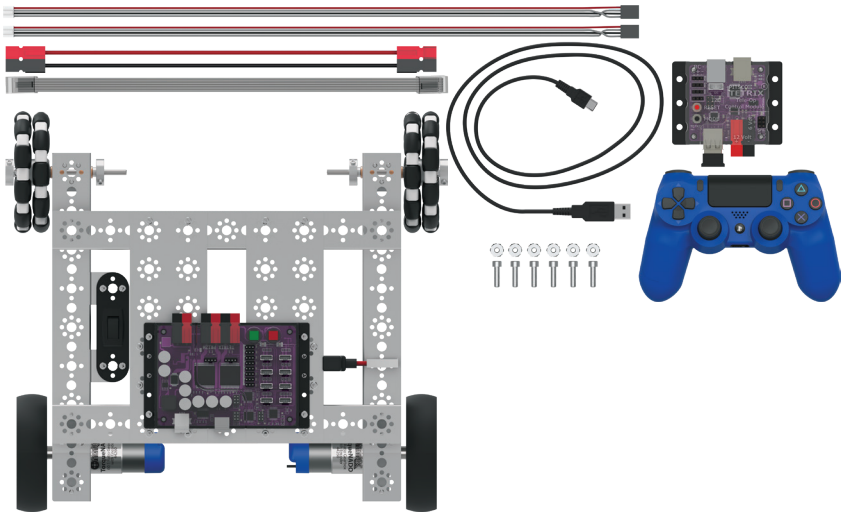
Tele-Op Module Activity 2 – TaskBot Control

Overview

In this activity, you will understand how the Tele-Op module enables you to control a robot using the buttons, triggers, joysticks, and sensors associated with the PS4 DUALSHOCK 4 gaming controller. You will first need to build the TaskBot using instructions from the *PRIZM® Programming Guide*. The TaskBot should include the PRIZM controller, the Line Finder and Ultrasonic Sensors, and the servo/flag attachment that is used for the final activity in the *PRIZM Programming Guide*.

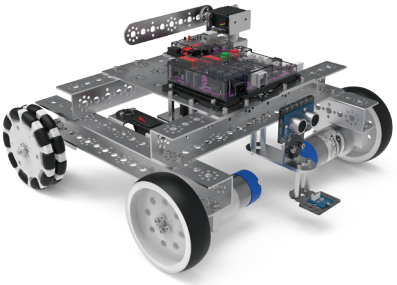
Parts Needed

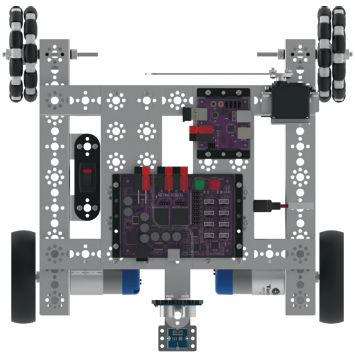
- Built TETRIX® MAX TaskBot from *PRIZM Programming Guide*
- Tele-Op module
- PS4 DUALSHOCK 4 gaming controller
- Powerpole extension cable
- Daisy-chain data cable
- 2 motor encoder cables
- USB cable
- Socket head cap screws
- Kep nuts



Hardware Connections

1. Connect one end of the Powerpole extension cable to the other battery connection port on PRIZM. Connect the other end of the Powerpole extension cable to the battery connection port on the Tele-Op module.
2. Connect one end of the daisy-chain data cable to the I2C expansion port on PRIZM. Connect the other end of the daisy-chain data cable to the I2C port on the Tele-Op module.
3. Choose a location on the TaskBot to attach the Tele-Op module. Make sure that the location is close enough to PRIZM that the power connection cable and the daisy-chain data cable will reach between the Tele-Op module and PRIZM. Attach the Tele-Op module using leftover socket head cap screws and kep nuts from your TETRIX MAX set.
4. This activity uses the motor encoders built into the TETRIX MAX TorqueNADO® Motors. Connect one end of a motor encoder cable to a TorqueNADO motor. Connect the other end of the cable to the encoder port on PRIZM that corresponds to the motor port for that motor (Motor 1 to Encoder Port 1, Motor 2 to Encoder Port 2).
5. Make sure that the Ultrasonic Sensor on the TaskBot is connected to Port D2 on PRIZM.
6. Make sure that the Line Finder Sensor is connected to Port D3 on PRIZM.
7. Make sure that the servo that raises and lowers the flag on the TaskBot is connected to Servo Port 1 on PRIZM.



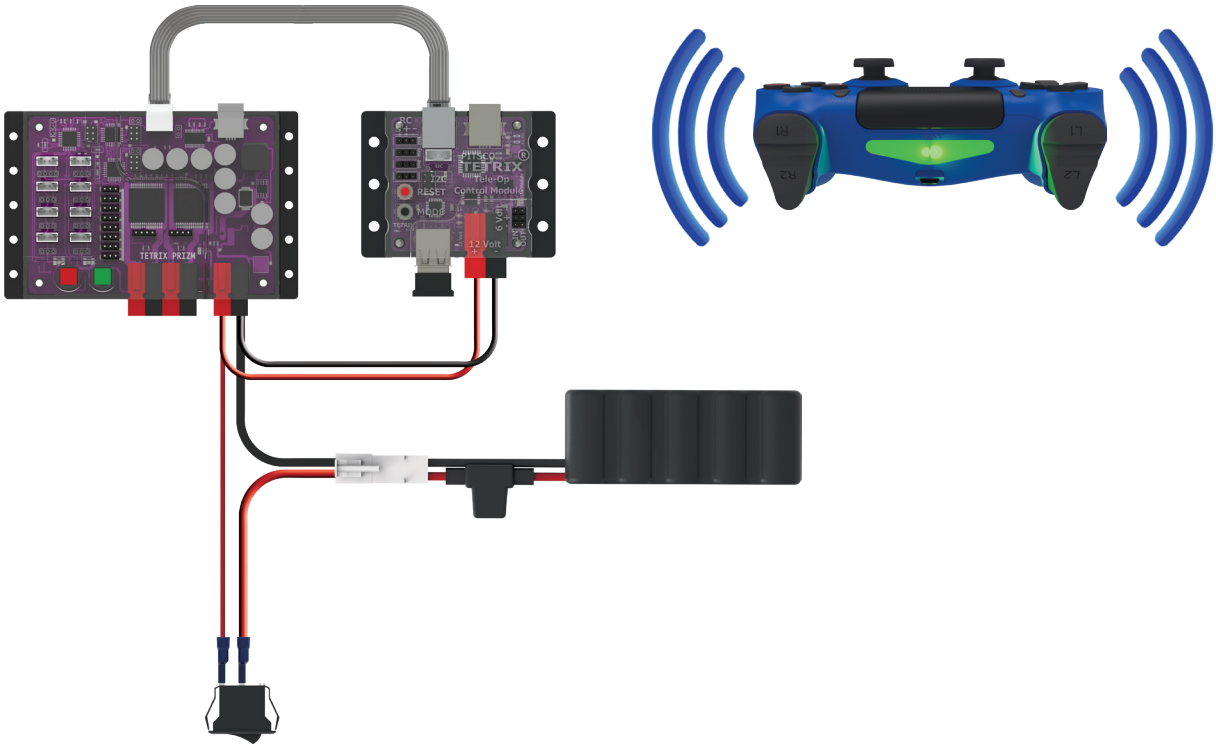


Bluetooth Connections

1. Make sure all devices that your PS4 controller has previously been paired with are off. This enables the PS4 controller to search for and connect to your Tele-Op module.
2. Make sure the PS4 controller is off.
3. Plug the Bluetooth dongle into the Tele-Op module's USB port.
4. Turn on the PRIZM controller. The PRIZM controller and the Tele-Op module should both have a blue LED that lights up, indicating they have power. You should also notice a green flashing LED on the Tele-Op module, indicating that the Bluetooth dongle is compatible. If you do not see a green flashing light, then no Bluetooth dongle is connected or the dongle is not compatible with the Tele-Op module. Refer to the Tele-Op technical guide for more information on Bluetooth compatibility.
5. To pair the PS4 controller with the Bluetooth dongle, the controller must be put into discovery mode. On the PS4 controller, hold down the Share and the Power buttons at the same time for about five seconds until you see the controller's light bar flash white in a rapid pattern. The controller is now in discovery mode.
6. Press the black button on the Tele-Op module. The green LED on the Tele-Op module should remain on and stop flashing. Also, the light bar on the game controller should be a solid green color to indicate the controller is paired and ready to be used to control the Tele-Op module.
7. After the PS4 controller has been paired with the Bluetooth dongle in the Tele-Op module, you don't need to go through this process again unless you want to pair a different device or Bluetooth dongle. Simply turn on PRIZM and the Tele-Op module and then press the Power button on the PS4 controller. The controller will automatically connect to the Bluetooth dongle in the Tele-Op module.

Note: When you connect the PS4 controller to the Tele-Op module, the game controller pairs with the Bluetooth dongle that is inserted into the Tele-Op module's USB port. If you remove the dongle from one Tele-Op module and plug it into another, the game controller can now be used to control the second Tele-Op module.

Note: When you turn off power to PRIZM and the Tele-Op module, the PS4 controller will automatically turn off after about 10 seconds.



Opening the Sketch

On your computer, open the Arduino Software (IDE). The sketch for this activity is stored with the examples that come with the Tele-Op library. To open the sketch, select **File** > **Examples** > **TETRIX_TeleOp** > **PS4_Examples** > **TaskBot_Activity_1**. A new sketch window will open with the example sketch.

Executing the Code

1. Connect the USB cable to a USB port on your computer or device. Connect the other end of the cable to the USB port on PRIZM.
2. Make sure the PRIZM and Tele-Op module are turned on and that your PS4 controller is paired with your Tele-Op module. There should be a solid green LED showing on both the PS4 controller and the Tele-Op module.
3. In the Arduino Software (IDE), make sure the correct COM port is selected to communicate with PRIZM. Go to **Tools** > **Port** and then select the correct COM port.
4. In the software window, click the Upload button to upload the sketch. After the sketch has been uploaded, unplug the USB cable from PRIZM.
5. Start the program by pressing the green Start button on PRIZM. You should now be able to control the movement of the TaskBot using the PS4 controller. Try out these different controls.

Note: The Tele-Op module has a USB port that is used to flash its firmware. This USB port should not be used when uploading sketches. Make sure you use the USB port on PRIZM. Uploading a sketch to the Tele-Op module will overwrite its firmware, making it unusable until it can have its firmware flashed again.

Set Motor Power

Drag your finger across the touch pad from left to right to set the speed of your motors. The further you drag to the right, the faster your motors will turn. You can adjust the speed at any time by touching a different position on the touch pad.

Tank Drive Navigation

Use the joysticks to drive the TaskBot around. The left joystick should control the left motor and the right joystick should control the right motor. If they are opposite, turn off your TaskBot. Switch the motor ports and encoder ports that each TorqueNADO is plugged into, turn your TaskBot back on, and restart the program. You should find that the farther you push the joysticks up, the faster the TaskBot will move forward. The farther you pull the joysticks down, the faster the TaskBot will move backward.

Direction Pad Control

You can also steer the TaskBot using the four buttons on the direction pad. This enables you to steer the TaskBot with your left hand while freeing up your right hand to press other buttons.

Note: Due to wheel slippage, you might notice the TaskBot not making an exact 90-degree turn. The friction between the wheel and the surface you are driving on is always an issue when trying to make precise movements.

90-Degree Turns

To make a 90-degree turn, press the L2 or R2 button. Using the encoders, the TaskBot will automatically perform a 90-degree turn to the left or right.

Crawl Mode

While driving the TaskBot, hold down the L1 button to enter crawl mode. This cuts the motor power down to 15% of its current speed, slowing the robot down. Although the robot is much slower in crawl mode, you have much more precise control of the robot to get it to go exactly where you want it to go.

Turbo Mode

While driving the TaskBot, hold down the R1 button to enter turbo mode. This increases the motor power to a full 100 percent of the speed set by the PS4's touch pad. Turbo mode enables you to move your TaskBot much faster but makes it more difficult to have precise control.

Pitch and Roll Mode

You can also use the PS4 controller's internal gyroscope to control your TaskBot. Hold down the L1 and R1 buttons at the same time to enter pitch and roll mode. While continuing to hold these buttons down, tilt the controller forward to move forward, back to go backward, left to turn left, and right to turn right.

Line-Following Mode

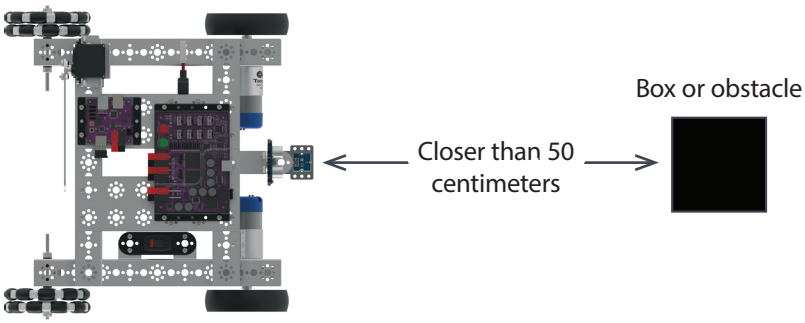
Drive the TaskBot up to a line so that the Line Finder Sensor is directly over the line. The TaskBot should line up with the line so that it faces the same direction that the line goes. Hold down the L2 and R2 buttons at the same time to enter line-following mode. You should see the TaskBot start following the line.

Raising the Flag

Use the Cross, Square, Triangle, and Circle buttons to set the flag to different positions.

Proximity Warning

While driving the TaskBot, you might notice that the PS4 controller will vibrate if you get too close to an object. Using the Ultrasonic Sensor to detect distance, PRIZM will turn on the PS4 rumble motors if an object comes within 50 centimeters.



Moving Forward

After you've experimented with the different methods of controlling the TaskBot using the PS4 controller, it's time to dig into the code that runs the TaskBot. Look at the code for the TaskBot_Activity_1 sketch in the Arduino Software (IDE). The code is broken down into eight sections.

Initialization

The beginning of the sketch starts by including the PRIZM and Tele-Op libraries and creating instances within each library class so that we can use the library functions.

This section also has all the global variable declarations. Recall that a global variable can be used anywhere in the sketch. Four of the variables are float-type variables that allow the use of decimal values. This is important for setting the percentages of the motor power for the turbo (100%), crawl (15%), and normal (35%) modes.

There are also four integer variables – one for storing the *x* value from the touch pad, one for storing the distance measured by the Ultrasonic Sensor to operate the PS4 controller's rumble motors, and two for storing the unique byte values of different button combinations.

```
#include <PRIZM.h>
#include <TELEOP.h>

PRIZM prizm;
PS4 ps4;

float highSpeed = 1;
float lowSpeed = highSpeed*.15;
float medSpeed = highSpeed*.35;
float powerMultiplier = medSpeed;
int touchx = 0;
int lineBtnCombo = 5;
int pitchRollBtnCombo = 10;
int obstacleDist = 50;
```

void setup()

The **void setup()** section has two new commands related to the Tele-Op module. The **ps4.setDeadZone** commands set the size of the dead zones for the left and right joysticks on the PS4 controller. By setting a dead zone, you identify the size of an area for the joystick's *x*- and *y*-axis that is considered neutral. When the joystick returns to this neutral dead zone area, the joystick is considered to be centered.

Setting a higher dead zone value gives you a bigger neutral area but reduces the sensitivity of the joystick outside the dead zone. On the other hand, a small dead zone gives you more precise sensitivity but reduces the neutral area of the joystick.

```
void setup() {
 prizm.PrizmBegin();
 ps4.setDeadZone(LEFT,10);
 ps4.setDeadZone(RIGHT,10);
 prizm.setMotorInvert(2,1);
 prizm.setServoPosition(1,180);
}
```

void loop()

The main loop starts with the **ps4.getPS4()** command. This command gets the status of all the PS4 controller's buttons, triggers, joysticks, and sensors and returns the information to PRIZM. This command needs to be frequently repeated so that as the user uses the controller, the status of what buttons are pressed gets communicated to PRIZM.

```
ps4.getPS4();
```

The first if statement determines if there is an obstacle within the designated detection range of the Ultrasonic Sensor. If the sensor detects an obstacle, the PS4's rumble motors are turned on. Otherwise, they are stopped.

```
if (prizm.readSonicSensorCM(2) < obstacleDist){ps4.setRumble(SLOW);}
else{ps4.setRumble(STOP);}
```

The second if statement determines if there is a difference between what is stored in the **touchx** variable and the current *x* value of the PS4's touch pad. The **ps4.Touchpad(TOUCHX)** command is what returns the *x* value of the touch pad. If there is a difference between these two values, then the **touchPadControl()** function is called. Otherwise, this function is skipped over. The **touchPadControl()** function is where the speed of the motors are set using the touch pad.

```
if (touchx != ps4.Touchpad(TOUCHX)){touchPadControl();}
```

Next, there are three while loops. These while loops determine if certain button combinations are pressed. The **ps4.buttons_1** command returns a unique byte value depending on which buttons are pressed from Group 1. Group 1 buttons include L1, L2, R1, R2, Triangle, Square, Circle, and Cross. Like Group 1, the **ps4.buttons_2** command also returns a unique byte value based on which buttons from Group 2 are pressed. Group 2 includes the Up, Down, Left, Right, L3, R3, Share, and Options buttons.

```
while (ps4.buttons_1 == lineBtnCombo){lineFollow();}
while (ps4.buttons_1 == pitchRollBtnCombo){pitchRollControl();}
while (ps4.buttons_2 != 0){directionButtonControl();}
```

The first while loop will continuously call the **lineFollow()** function if the value returned from **ps4.buttons_1** equals the **lineBtnCombo** variable, which is 5. Five is the unique value for when both the L2 and R2 buttons are pressed at the same time. To simplify, the sketch will continuously run the **lineFollow()** function if the L2 and R2 buttons are both pressed.

Similar to the first while loop, the second while loop continuously calls the **pitchRollControl()** function when the L1 and R1 buttons are pressed. When these two buttons are pressed, the **ps4.buttons_1** command will return 10, which is the same value that is set for the **pitchRollBtnCombo** variable, and the **pitchRollControl()** function is executed.

The third while loop is for controlling the robot with the direction buttons. The direction buttons are part of Group 2, which is why **ps4.buttons_2** is used. The only time **ps4.buttons_2** returns 0 is when none of the buttons in Group 2 are pressed. This while loop uses a not (!=) comparison to say, "while not none of the buttons in Group 2 are pressed, call the **directionButtonControl()** function." *Not none* doesn't make a lot of sense in English, but in coding, a double negative makes a positive just like it does in math. So in other words, this while loop says, "while at least one of the Group 2 buttons is pressed, call the **directionButtonControl()** function."

Following the three while loops, we have a function call for **adjustMotorPowers()**. This function determines if the TaskBot should be in crawl mode, normal mode, or turbo mode. This function will be explained in further detail a little later.

The next command line is for driving the TaskBot with the joysticks. This command sets the motor power level based on the vertical positions of the left (for the left motor) and right (for the right motor) joysticks. The commands **ps4.Motor(RY)** and **ps4.Motor(LY)** get the *y* values of the joysticks and map these values to motor power values between -100 for all the way down and 100 for all the way up. However, these joystick positions are multiplied by the **powerMultiplier** variable. The **powerMultiplier** variable is either 15% (0.15) for crawl mode, 35% (0.35) for normal mode, or 100% (1.00) for turbo mode. This reduces the maximum speed of the motors when in crawl mode or normal mode.

```
prizm.setMotorPowers(ps4.Motor(RY)*powerMultiplier, ps4.Motor(LY)*powerMultiplier);
```

The L2 and R2 buttons are used to turn the TaskBot left or right by 90 degrees. The next three lines of the main loop provide this functionality.

```
if (ps4.Button(L2)==1){prizm.setMotorDegrees(360,270,360,-270);delay(1200);}
if (ps4.Button(R2)==1){prizm.setMotorDegrees(360,-270,360,270);delay(1200);}
prizm.resetEncoders();
```

The first if statement turns the TaskBot left, and the second if statement turns the TaskBot right. The command **ps4.Button(*BUTTON*)** will return 1 if *BUTTON* is pressed or 0 if *BUTTON* is not pressed. In this command, *BUTTON* can be any of the following PS4 buttons:

- L1
- L2
- L3
- R1
- R2
- R3
- UP
- DOWN
- RIGHT
- LEFT
- TRIANGLE
- CROSS
- CIRCLE
- SQUARE
- SHARE
- OPTIONS
- POWER
- TOUCH

These if statements also use motor encoder commands to turn the motors a specified number of degrees (+ or - 270 degrees) at a specified speed (360 degrees per second). Notice that each if statement also includes a delay to wait for the motors to complete the command before continuing with the loop. After completing the turn, the encoders are reset for the next iteration of the loop.

The last four lines of the main loop are used to control the servo that raises and lowers the flag. These if statements use the same **ps4.Button(*BUTTON*)** command to check the status of the Cross, Square, Triangle, and Circle buttons. When one of these buttons is pressed, the servo that controls the flag is set to a corresponding position.

```
if (ps4.Button(CROSS)==1){prizm.setServoPosition(1, 180);}
if (ps4.Button(SQUARE)==1){prizm.setServoPosition(1, 135);}
if (ps4.Button(TRIANGLE)==1){prizm.setServoPosition(1, 90);}
if (ps4.Button(CIRCLE)==1){prizm.setServoPosition(1, 0);}
```

void adjustMotorPowers()

This called function determines the speed of the TaskBot based on whether the user wants to be in crawl mode, normal mode, or turbo mode. It is composed of a three-part conditional statement that runs one of three sets of commands. The if statement determines if the L1 button is pressed using the **ps4.Button(L1)** command. If the L1 button is pressed, the **powerMultiplier** variable is set to the **lowSpeed** value (0.15, or 15%) and the LED on the PS4 controller is changed to red.

```
void adjustMotorPowers(){
  if(ps4.Button(L1)==1){
    powerMultiplier = lowSpeed;
    ps4.setLED(RED);}
  else if(ps4.Button(R1)==1){
    powerMultiplier = highSpeed;
    ps4.setLED(GREEN);}
  else {
    powerMultiplier = medSpeed;
    ps4.setLED(YELLOW);}
}
```

If the L1 button is not pressed, then the second part of the conditional statement is checked – the else if part. This statement determines if the R1 button is pressed using the **ps4.Button(R1)** command. If the user presses the R1 button for turbo mode, then the **powerMultiplier** is set to the **highSpeed** value (1.00, or 100 percent). The PS4's LED is also set to green.

If neither of the first two conditions are met, then a third set of commands is executed – the else part. Since the user does not want crawl mode or turbo mode, then the only other option is normal mode. The **powerMultiplier** is set to the **medSpeed** variable (0.35, or 35%), and the PS4's LED is set to yellow.

void touchPadControl()

The **touchPadControl()** function sets the maximum speed of the TaskBot using the PS4's touch pad. The command **ps4.Touchpad(TOUCHX)** returns a value of 0 to 1920 depending on the *x*-coordinate of where the user last touched the PS4's touch pad. This value is stored in the **touchx** variable so it can be used later.

Because the touch pad can go all the way up to 1920 but our motors have a max power of only 100, the map function is used to set the **highSpeed** variable to a value between 0 and 100. Then, the **highSpeed** variable is converted to a decimal number between 0 and 1 to represent a percentage of the maximum motor power.

```
void touchPadControl(){
  touchx = ps4.Touchpad(TOUCHX);
  highSpeed = map (touchx,0,1920,0,100);
  highSpeed = highSpeed / 100;
  medSpeed = highSpeed*.35;
  lowSpeed = highSpeed*.15;
}
```

After the **highSpeed** variable is set, the **medSpeed** and **lowSpeed** variables can be set. The **medSpeed** variable is set to 35% of the **highSpeed** value, and the **lowSpeed** variable is set to 15% of the **highSpeed** value.

void lineFollow()

The **lineFollow()** function gets called when the L2 and R2 buttons on the PS4 controller are pressed at the same time. Remember that in the main loop, this function is called from inside a while loop. As long as the L2 and R2 buttons remain pressed, the sketch will not break out of the while loop, and this function will continuously be executed. This is why the first line of the function is the **ps4.getPS4();** command. Without this command, the button and joystick status of the PS4 controller would never get updated, and the sketch would never break out of the while loop. Including this command at the beginning of the function enables the program to know when either the L2 or R2 button is released and the time to stop line following.

```
void lineFollow(){
  ps4.getPS4();
  ps4.setLED(BLUE);
  if(prizm.readLineSensor(3) == 0){prizm.setMotorPowers(125,30);}
  if(prizm.readLineSensor(3) == 1){prizm.setMotorPowers(30,125);}
}
```

The other three lines of code in this function should look familiar by now. The PS4 controller's LED is set to blue to indicate the TaskBot is in line-following mode. The two if statements read the Line Finder Sensor and cause the TaskBot to turn one way if the sensor detects dark and the other way if the sensor detects white.

void pitchRollControl()

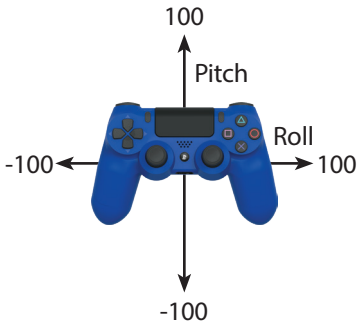
When the PS4 controller's L1 and R1 buttons are pressed at the same time, the TaskBot goes into pitch and roll mode, and this function is called from the main loop. Similar to the **LineFollow()** function, this function is called from within a while loop. The program will not break out of the while loop until either the L1 or R1 button is released. Because of this, it is important to check the status of the PS4 controller's buttons every iteration of the while loop. This is why the first command inside the function is the **ps4.getPS4()** command. This command updates the status of the controller's buttons, joysticks, and gyroscope sensor values, and it is the gyroscope sensor values that determine the pitch and roll.

```
void pitchRollControl(){
  ps4.getPS4();
  ps4.setLED(BLUE);
  prizm.setMotorPowers(ps4.Motor(PITCH)+ps4.Motor(ROLL),ps4.Motor(PITCH)-ps4.Motor(ROLL));
  ps4.setLED(RED);
}
```

Also inside the function are two commands to change the color of the controller's LED. These two commands work together to cause the LED to flash between blue and red, indicating that the TaskBot is in pitch and roll mode.

The TaskBot's motors are controlled with the third command in this function. Although it looks complicated, the **prizm.setMotorPowers()** command has just two values in it: the power for Motor 1 and the power for Motor 2. These two values are separated by the comma.

For Motor 1, the command adds the pitch and roll measurements from the PS4 controller's gyroscope. For Motor 2, the controller's roll measurement is subtracted from the controller's pitch measurement. The table below shows the TaskBot's behavior when the PS4 controller is tilted to its maximum tilt.



Controller Tilt	Pitch	Roll	Motor 1 P + R	Motor 2 P - R	Result	Motion
All the way forward	100	0	100	100	Forward at full speed (both motors forward)	
All the way backward	-100	0	-100	-100	Backward at full speed (both motors backward)	
All the way left	0	-100	-100	100	Rotate left/ counterclockwise (left motor backward, right motor forward)	
All the way right	0	100	100	-100	Rotate right/clockwise (left motor forward, right motor backward)	
Forward and left	100	-100	0	200	Pivot left forward (left motor stationary, right motor forward)	
Forward and right	100	100	200	0	Pivot right forward (left motor forward, right motor stationary)	
Backward and left	-100	-100	-200	0	Pivot right backward (left motor backward, right motor stationary)	
Backward and right	-100	100	0	-200	Pivot left backward (left motor stationary, right motor backward	

Notice that in the last two table entries, the TaskBot moves the opposite direction of the controller tilt, but because of the backward motion, the TaskBot ends up facing the direction of the tilt.

void directionButtonControl()

The last called function of the sketch is the function for controlling the TaskBot using the PS4 controller's direction pad. In the main loop, this function is called from within the third while loop and is executed when one of the buttons from Button Group 2 is pressed.

```
void directionButtonControl(){
  ps4.getPS4();
  if (touchx != ps4.Touchpad(TOUCHX)){touchPadControl();}
  adjustMotorPowers();

  if (ps4.buttons_2 == 1){prizm.setMotorPowers(100*powerMultiplier, 100*powerMultiplier);}
  if (ps4.buttons_2 == 2){prizm.setMotorPowers(-100*powerMultiplier, -100*powerMultiplier);}
  if (ps4.buttons_2 == 4){prizm.setMotorPowers(-100*powerMultiplier, 100*powerMultiplier);}
  if (ps4.buttons_2 == 8){prizm.setMotorPowers(100*powerMultiplier, -100*powerMultiplier);}
  if (ps4.buttons_2 == 5){prizm.setMotorPowers(25*powerMultiplier, 100*powerMultiplier);}
  if (ps4.buttons_2 == 6){prizm.setMotorPowers(-100*powerMultiplier, -25*powerMultiplier);}
  if (ps4.buttons_2 == 9){prizm.setMotorPowers(100*powerMultiplier, 25*powerMultiplier);}
  if (ps4.buttons_2 == 10){prizm.setMotorPowers(-25*powerMultiplier, -100*powerMultiplier);}
}
```

Because this function is called from within a while loop, it is necessary to include the **ps4.getPS4()** command to update the status of the PS4 controller's buttons, joysticks and sensors.

Next, the function includes an if statement to determine if the value of the touch pad has changed. If the user is using the touch pad to adjust the motor speed, then the **touchPadControl()** function is called to adjust the crawl, normal, and turbo mode speeds.

After that, the **adjustMotorPowers()** function is called to adjust the **powerMultiplier** based on if the TaskBot is in crawl mode, turbo mode, or normal mode.

Finally, the function uses eight if statements to determine which buttons from the directional pad are pressed and to adjust the motors accordingly. The **ps4.buttons_2** command is used to get the uniquely assigned value based on what button or buttons are pressed on the direction pad.

The table here shows the direction pad buttons, the unique value associated with each button combination, and the resulting action when each button combination is pressed.

Button	Group 2 Value	Group 2 Binary Value	Action
	1	1	Go forward
	8	1000	Rotate left
	2	10	Go backward
	4	100	Rotate right
	9	1001	Pivot forward left
	10	1010	Pivot backward right
	6	110	Pivot backward left
	5	101	Pivot forward right

Hacking the Code

Start a new sketch. Using the TaskBot_Activity_1 sketch as an example, program the TaskBot to autonomously follow a line until an obstacle is detected with the Ultrasonic Sensor. When an obstacle is detected, the TaskBot should stop moving and wait for the user to take control using the PS4 controller to drive around the obstacle and back to the line. When the TaskBot is back on the line, the user should be able to press a button on the PS4 controller to have the TaskBot start autonomously following the line again.