Pirate Ship --- IR Obstacle Avoidance Switch

After completing the basic functions, we're now ready to give our robot an upgrade: obstacle avoidance capabilities. Obstacle avoidance is among a robot's most fundamental capabilities. It enables a robot to recognize and avoid obstacles that lie in its path. Here, we'll be using digital infrared obstacle avoidance sensors – they're relatively easy to use and thus ideal for beginners. Using these sensors will allow your robot to detect and avoid objects that might otherwise block its path.

Hardware Parts

- Infrared Sensor Switch x 3

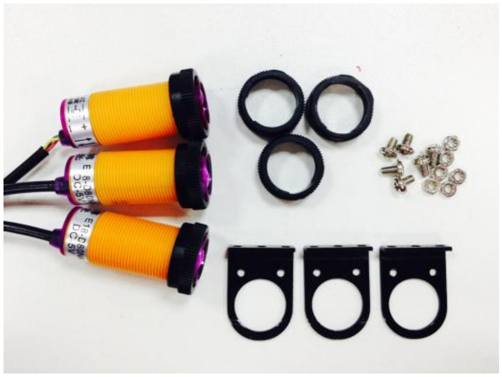
- Infrared Proximity Switch Bracket x 3

Assembly Directions

After finding the necessary parts, you're ready to start assembling. Assembly won't be too difficult – just follow the directions below and you'll have no problems.

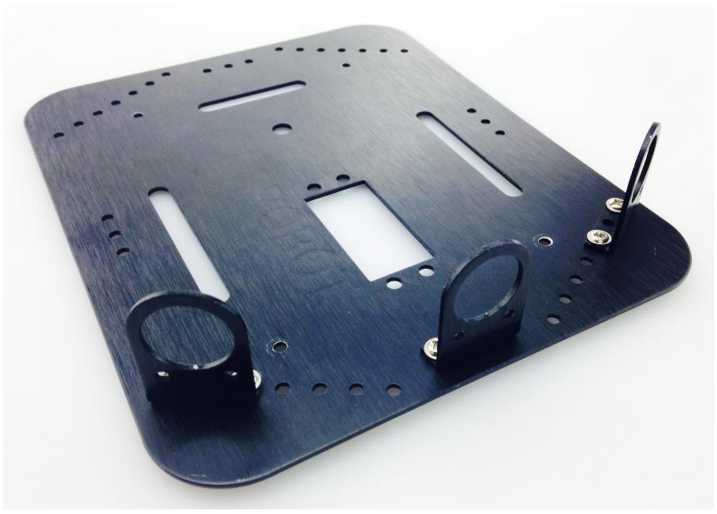
Step 1: Sort out and arrange materials

First, find the 3 support frames for the IR sensors. You'll need several M3x6mm screws and nuts to fasten these frames.



Step 2: Attach the IR support frames

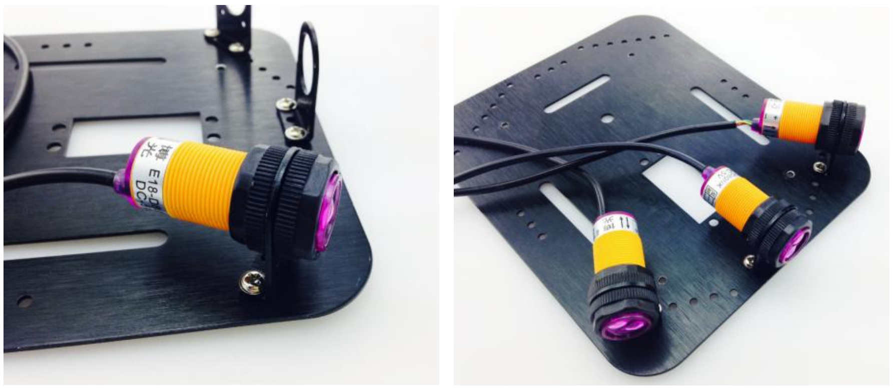
Screw on the three support frames to the Pirate's top plate. Make sure not to attach the support frames too close together – doing so will cause the sensors to mix up signals, which leads to errors.



Step 3: Assemble the IR sensors

Place the sensor through the fixed ring on the support frame. The sensor has two rings that can be adjusted to fasten the sensors into place. Use both rings to fasten the sensors into the support ring. The sensors should be facing outwards.

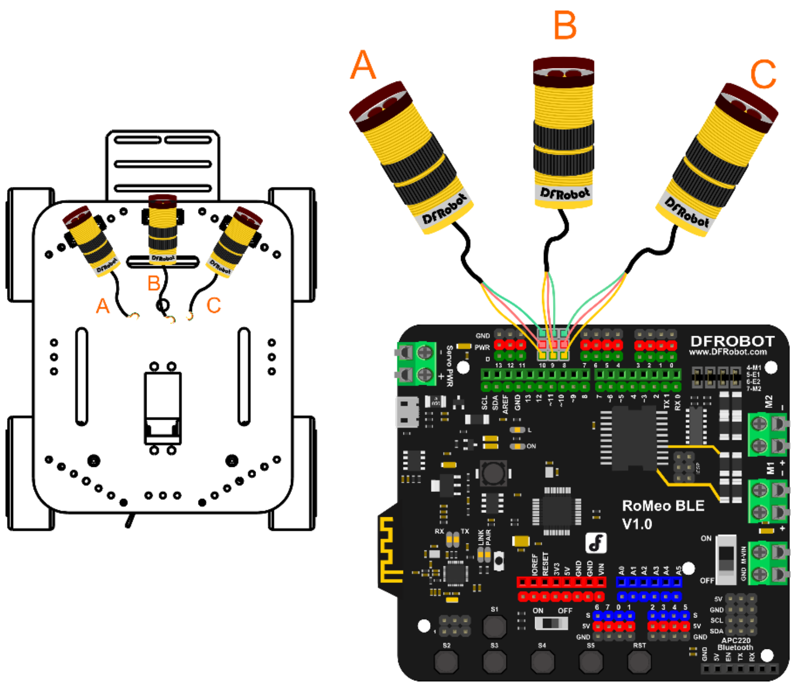
Repeat this process for all three sensors.

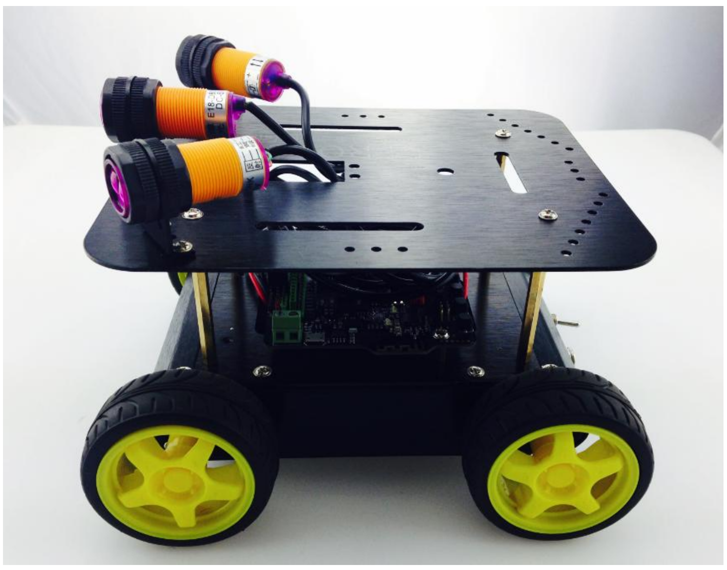


Connect the hardware

After assembling the sensors, don't rush to put the Pirate's top plate back on – before doing that, we need to first connect the sensors with the Romeo BLE control plate. The picture on the left-hand side shows the correct ABC placement of the sensors on the sensor board, which corresponds to pins 10, 9, and 8 on the BLE board. When connecting the sensors, be sure to check that you've connected them in the proper sequence. After connecting the sensors, re-attach the Pirate's top plate back on the body of the car.

Yellow: Signal line, Red: VCC (positive), Green: GND (negative)



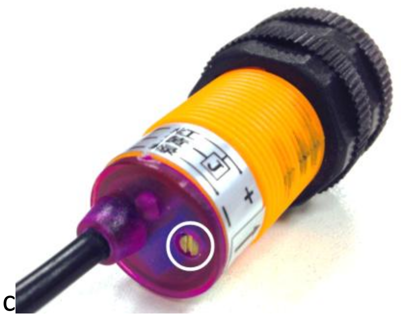


Adjusting the sensors

IR approach switches are a type of IR sensor that collects sent and received signals. If a signal is detected – i.e. an obstacle is detected in front of the Pirate – the LED on the back of the sensor will turn on. The screw head can be tightened to adjust the sensor’s distance detection to a range of 3 to 80 cm – that is, the sensor can detect objects as near as 3 cm or as far as 80 cm away.

Before downloading the code needed to configure the sensors, we need to first adjust their distance detection. First, take your USB cable and plug it into your microcontroller to power it on. Take a look at the below picture: the small, circled area is a screw head that can be turned to adjust the sensor’s distance detection. Take a **small box** (so as to imitate an obstacle) and place it in front of the head of the sensor. Once the sensor detects the box’s presence, the red LED light on the back of the sensor will light up. We want to calibrate the sensor to detect obstacles 15-20 cm away – to do this, place the box 15-20 cm away from the head of the sensor and tighten the screw head until the sensor’s LED lights up (which signifies that the object has been detected).

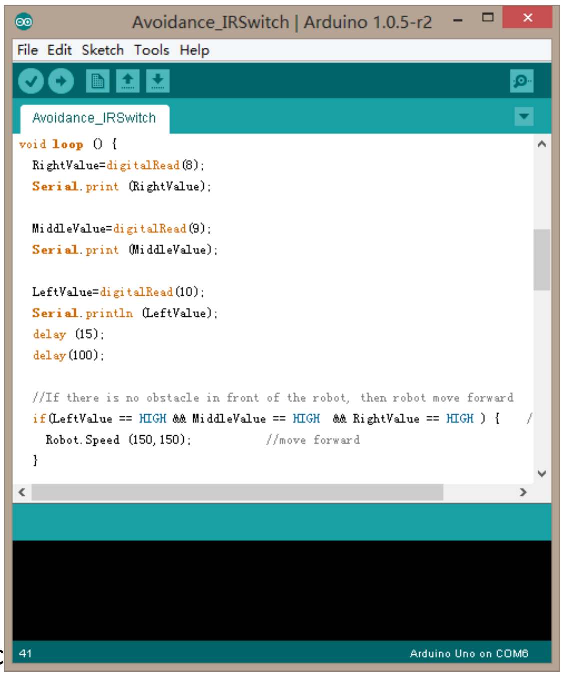
Adjust the left and right sensors in the same manner. We recommend that the left and right sensors be adjusted to the same distance (15-20 cm) as the middle sensor.



c

Coding

Since the length of the code is relatively long, we won’t delve into all the specifics. . Download the Arduino code, named “Avoidance_IRSwitch.ino”, from GitHub.


Avoidance_IRSwitch | Arduino 1.0.5-r2
File Edit Sketch Tools Help
Avoidance_IRSwitch
41
Arduino Uno on COM6

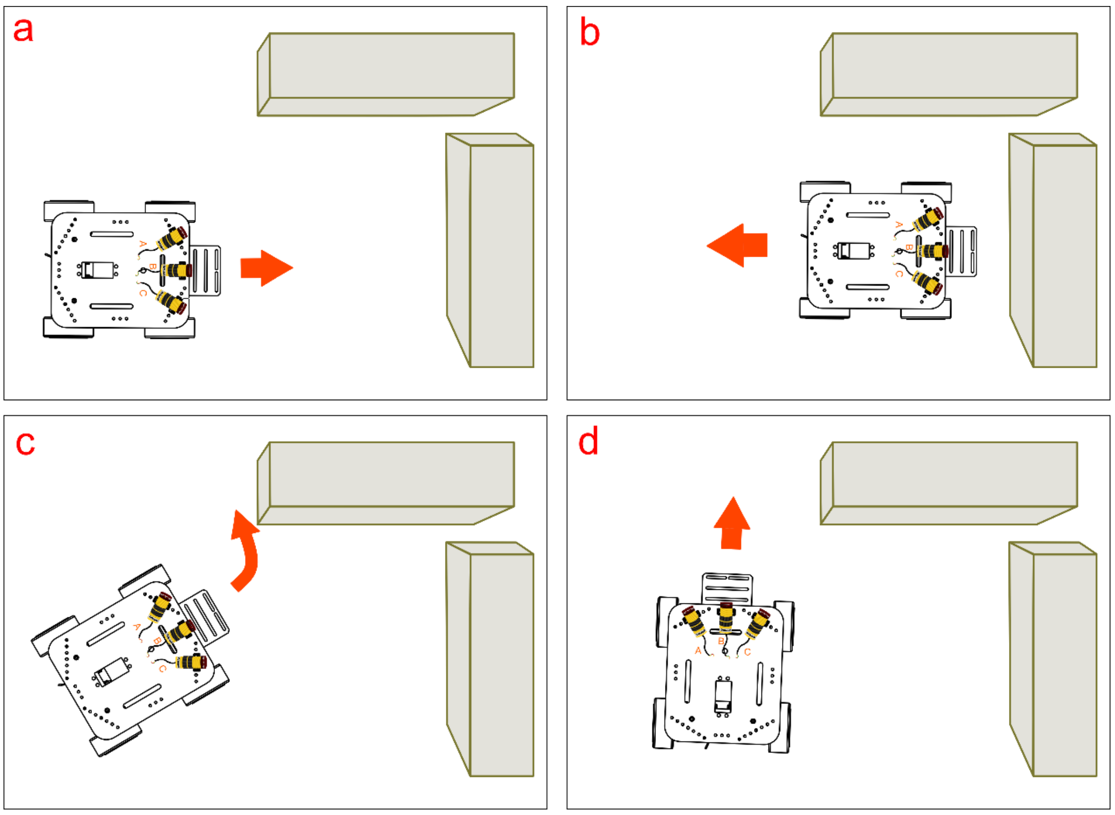
```
void loop () {
  RightValue=digitalRead(8);
  Serial.print (RightValue);

  MiddleValue=digitalRead(9);
  Serial.print (MiddleValue);

  LeftValue=digitalRead(10);
  Serial.println (LeftValue);
  delay (15);
  delay(100);

  //If there is no obstacle in front of the robot, then robot move forward
  if(LeftValue == HIGH && MiddleValue == HIGH  && RightValue == HIGH ) {    /
    Robot.Speed (150,150);            //move forward
  }
```

c

After downloading the code, upload it to your Romeo microcontroller. Once your car starts moving, you’ll be able to see the obstacle avoidance sensors in play – as soon as they detect an object in their path, they’ll light up and re-direct the car. If you feel like the sensor’s distance detection isn’t sufficiently strong, feel free to further adjust the sensors.

Obstacle avoidance: how it works

The following four diagrams (A, B, C, D) briefly illustrate how the three sensors work to avoid obstacles. At first, the car moves forward uninhibited (diagram A). Once it detects an obstacle, the sensors command the car to reverse (diagram B). The car will then attempt to move either left or right (diagram C). In the event that the car is surrounded by obstacles on all sides, the sensors will command the car to reverse and turn until it finds an open space without obstacles, whereupon the car will continue to move forward (diagram D).



The above diagrams depict the car facing an obstacle in front of its path. Consider: if the car has obstacles to both its left and front side, or both its right and front side, how would the car adjust its movement?

Of course, the Avoidance_IRSwitch.ino code provided isn't the only way of programming your sensors. Once you feel comfortable with the principles and basic code behind the sensors, feel free to write your own code and configure the sensors to your liking.

Code Synopsis

There's no need to discuss the basic code – let's just take a look at the part involving obstacle avoidance.

```
int RightValue;    //Right line tractor sensor on Pin 8
int MiddleValue;   //Middle line tractor sensor on Pin 9
int LeftValue;     //Left line tractor sensor on Pin 10

//reading 3 pins values of Line Tracking Sensor
RightValue=digitalRead(8);
MiddleValue=digitalRead(9);
LeftValue=digitalRead(10);
```

Use three variables – RightValue, MiddleValue, LeftValue – to record the 3 sensors' read values.

The digitalRead(pin) function is used to read the digital Input/Output port value. If these terms are still confusing, please check out our Terminology Manual or the Arduino website.

When the IR obstacle avoidance sensors detect an object in their path, they'll produce a LOW energy output. When there are no objects detected in their path, they'll produce a HIGH energy output.

The snippet of code below (example A) illustrates how the obstacle avoidance code works. If/when the three sensors detect an object in front of its path, they will produce a LOW energy output; they will then prompt the car to reverse, turn left, then attempt to move forward once again.

```
if(LeftValue==LOW && MiddleValue==LOW && RightValue==LOW ) {
   Robot.Speed (-200,-200);        //back off
   delay(800);
   Robot.Speed (-200,200);         //turn left
   delay(400);
   Robot.Speed (100,100);          //move forward
 }
```

If the left sensor detects an obstacle in its path, it will prompt the car to reverse, then turn **left**. See (B) below:

```
if(LeftValue == LOW) {    //obstacle on the left side
  Robot.Speed (-200,-200);    //back off
  delay(400);
```

```
      Robot.Speed (200,-200);    //turn right
      delay(250);
      Robot.Speed (100,100);    //move forward
    }
```

And if the right sensor detects an obstacle in its path, it will prompt the car to reverse, then turn **right**. See (C) below:

```
    if(RightValue == LOW ){       //obstacle on the right side
       Robot.Speed (-200,-200);     //back off
       delay(400);
       Robot.Speed (-200,200);     //turn left
       delay(250);
       Robot.Speed (100,100);     //move forward
     }
```

Lastly, if sensors do not detect any object in their path, then the car will move forward as normal. See (D) below:

```
    if(LeftValue==HIGH && MiddleValue==HIGH && RightValue==HIGH ) {
       Robot.Speed (150,150);          //move forward
    }
```

The code also contains the below snippet:

```
    if( MiddleValue == LOW){     //obstacle in middle
       Robot.Speed (-150,-150);     //back off
       delay(400);
       Robot.Speed (-200,200);      //turn left
       delay(300);
       Robot.Speed (100,100);    //move forward
    }
```

This is another example of the sensors detecting an obstacle in its path. In this instance, the obstacle detected is relatively further away from the sensors, which leads to the car to reverse at a slightly slower speed.

```
    Robot.Speed (-150,-150);    //back off
```

Here, we use Arduino’s Speed function to adjust the speed at which the car moves after receiving input from the sensors. Feel free to adjust the speed values (which range from -255 to 255) to your liking. Our robot can now detect and maneuver around obstacles. Interestingly, we can apply these same sensors to give our robot “tracking” capabilities. Unlike obstacle avoidance, which seeks to avoid detected objects, tracking capabilities command your car to follow an object once detected. Obstacle avoidance, tracking – these are powerful features that allow you to get creative. Time to start tinkering!

Considerations

Interestingly, we can apply these same sensors to give our robot “tracking” capabilities. Unlike obstacle avoidance, which seeks to avoid detected objects, tracking capabilities command your car to follow an object once detected. Obstacle avoidance, tracking – these are powerful features that allow you to get creative. With this in mind, get to tinkering and having fun!