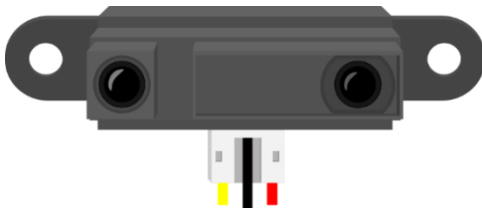


Pirate Ship --- GP2YA21 Obstacle Avoidance

After assembling the Pirate and completing its basic functions, we're ready for an upgrade: obstacle avoidance capabilities. Obstacle avoidance is among a robot's most fundamental capabilities, as it enables the robot to recognize and avoid obstacles that lie in its path. Here, we'll be using the GP2YA21 sensors: compared to normal digital sensors, the GP2YA21's distance detection is more precise; rather than just outputting "Yes/No" signals, the GP2YA21 can feed more detailed information to your car, which helps avoid objects in its path.

Hardware components

- GP2YA21 distance sensors x 1



- DF05BB 5.1Kg robot servo x 1



- Infrared Sensor Mounting Bracket x 1

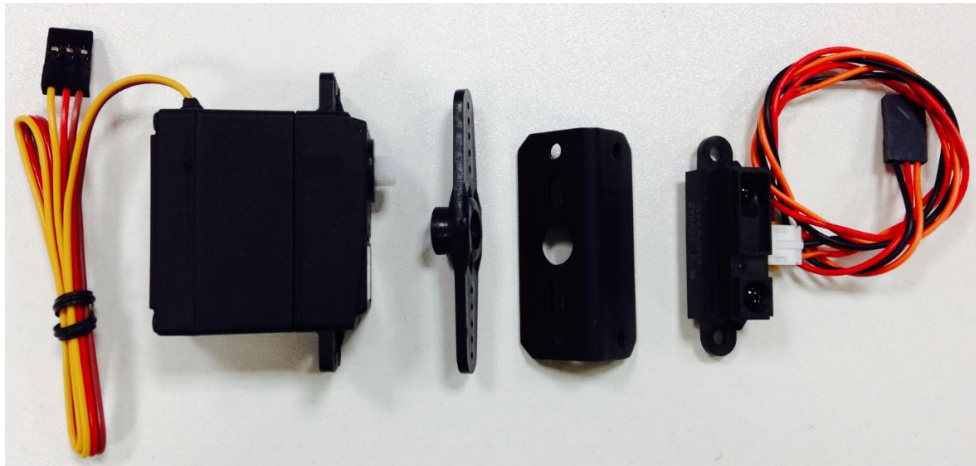


Assembly Directions

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.

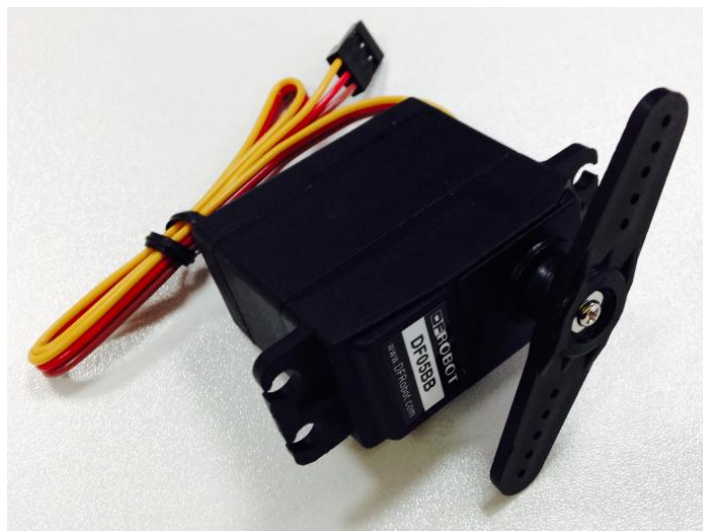
Step 1: Sort out and arrange materials

First, find and sort out all the materials from your parts bag. You'll find the necessary screws and bolts in the separate bag containing the robot servo parts.



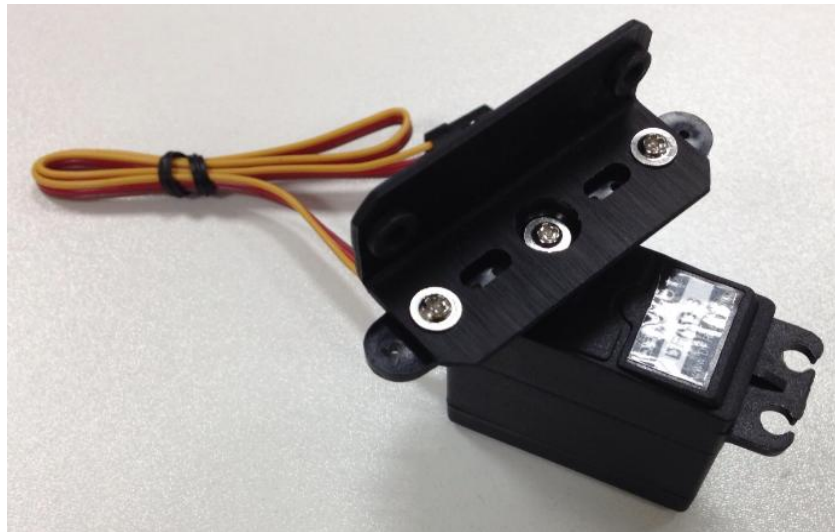
Step 2: Assemble the servo

Find the necessary screws to attach the servo horn to your servo. Screw in the horn as shown below.



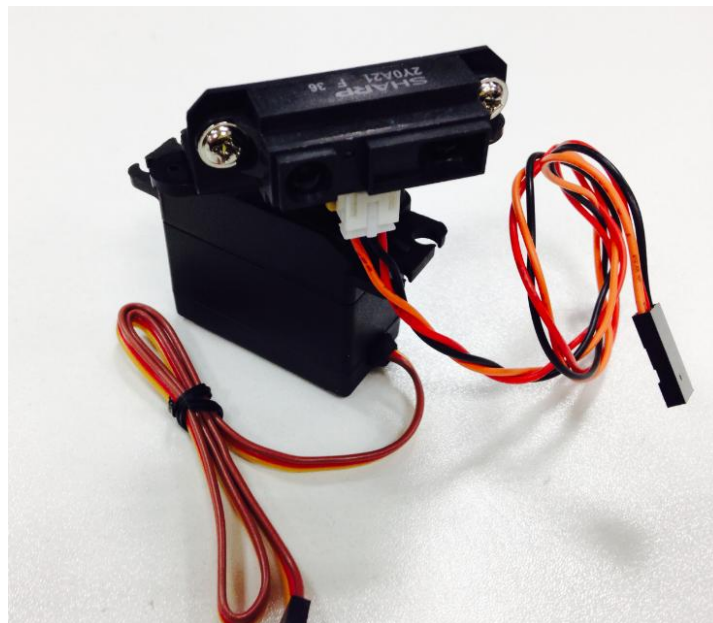
Step 3: Attach the IR sensor support frames

Use the necessary screws to attach the IR sensor support frames to the servo.



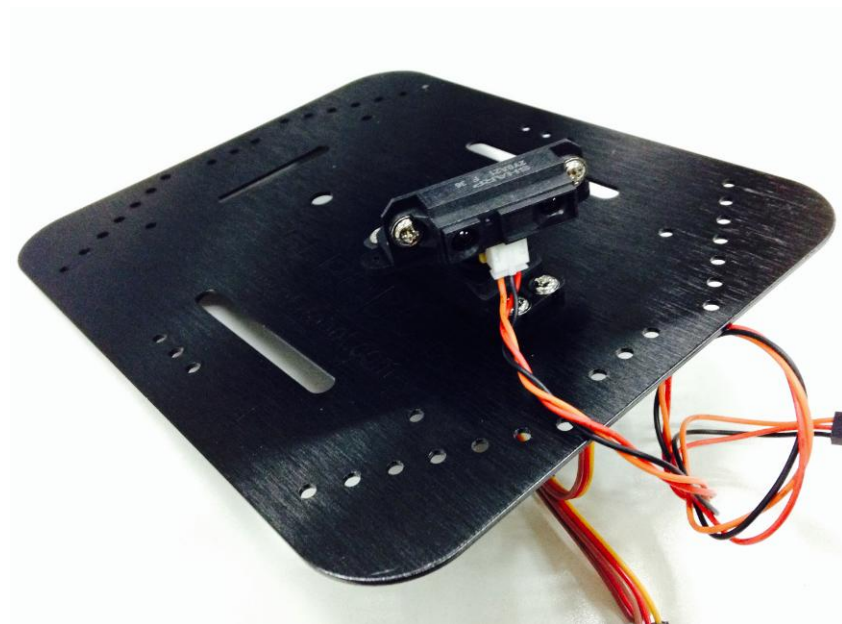
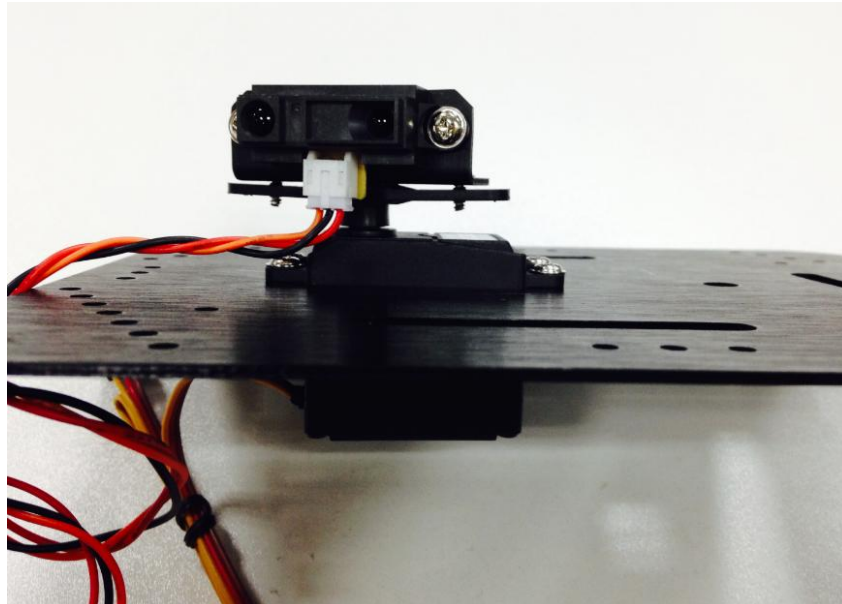
Step 4: Assemble the IR sensors

Use the provided M3 nuts and bolts to mount the sensors onto the two holes on the support frame — see below.



Step 5: Attach the IR sensors

After attaching the sensors to the servo, we now need to attach the servo to the top plate of the Pirate. After finding the four appropriate holes on the top plate, use four M3 screws and nuts to screw the servo into place.



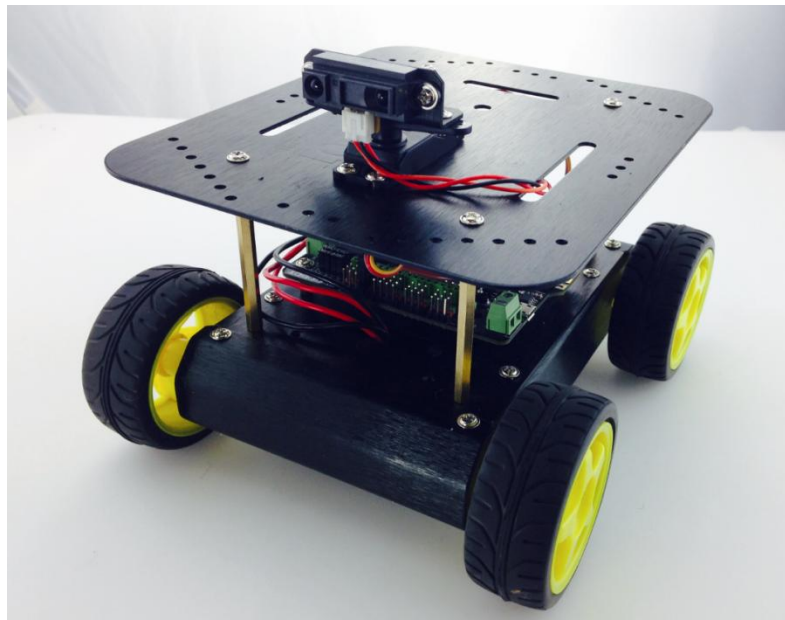
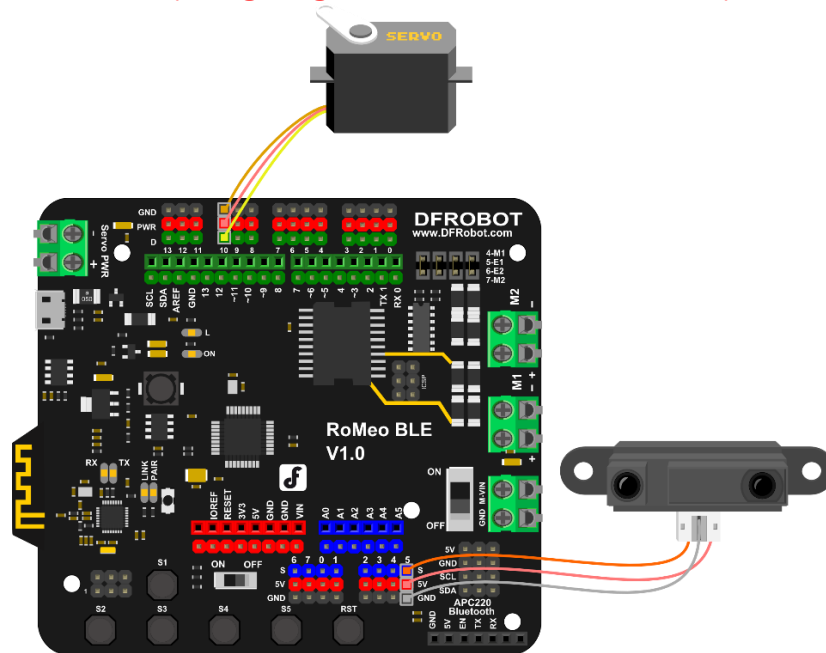
Congrats, you've completed the first step in installing your sensors! Before attaching the top plate back onto the Pirate, we still need to solder the wires onto the microcontroller.

Connect the hardware

Soldering the servo and sensors here is fairly straightforward. For the servo, solder the three wires to digital port 3. For the sensors, solder the three wires to analog port 5. Refer to the diagram below when soldering — make sure to pay attention to the proper sequence in soldering each wire.

Servo - D3 (Yellow: Signal wire, Red: VCC, Brown: GND)

GP2YA21 Sensors - A5 (Orange: Signal wire, Red: VCC, Black: GND)



Adjust the Sensors

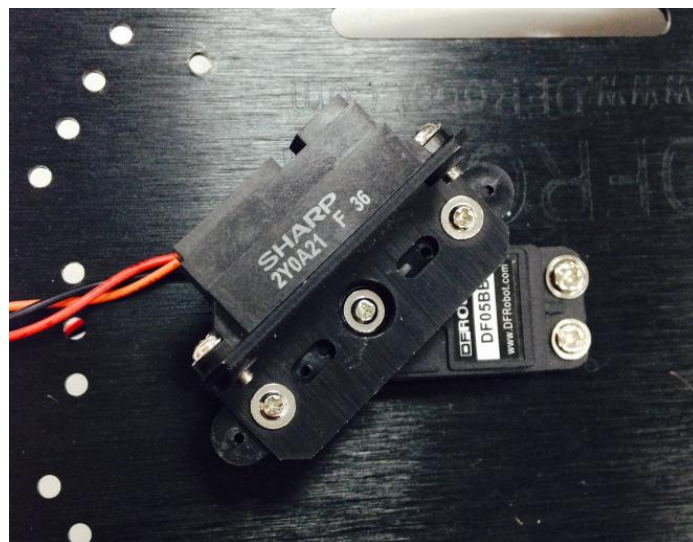
Servo Adjustment

In setting up our servo, we first need to configure and initialize it for use — in this case, we want to configure it to a 90° angle. How do we configure our servo? First, download and run the “Servo_Test.ino” sample code.

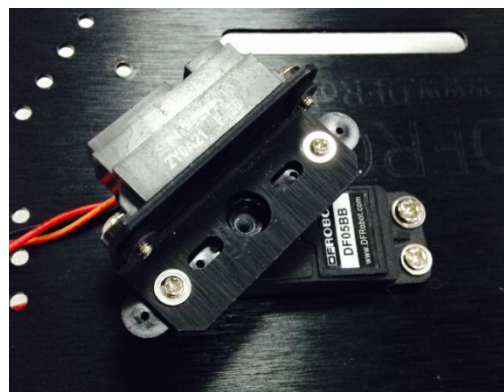
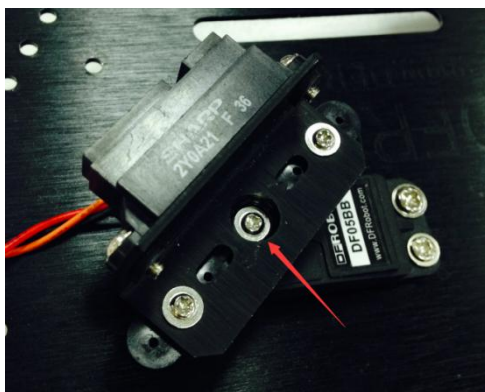
```
#include <Servo.h>
Servo myservo;  // create servo object to control a servo

void setup() {
  myservo.attach(10);  // attaches the servo on pin 3
}
void loop() {
  myservo.write(90);  // tell servo to go to position in 90
  delay(15);          // waits 15ms
}
```

After downloading the code and running it, your servo should switch angles like in the picture below:

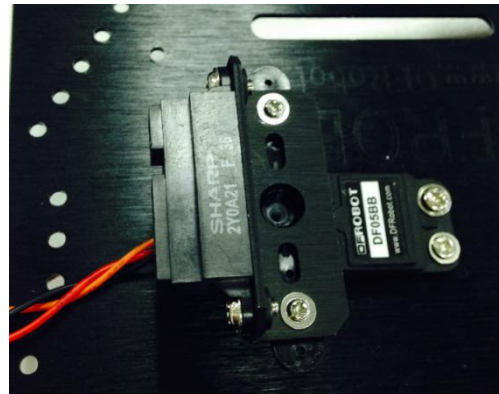
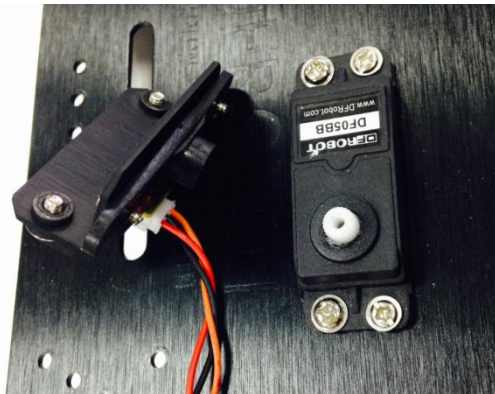


During this process, **do not unplug your USB cord!** Instead, unscrew the middle screw attaching the servo in place.



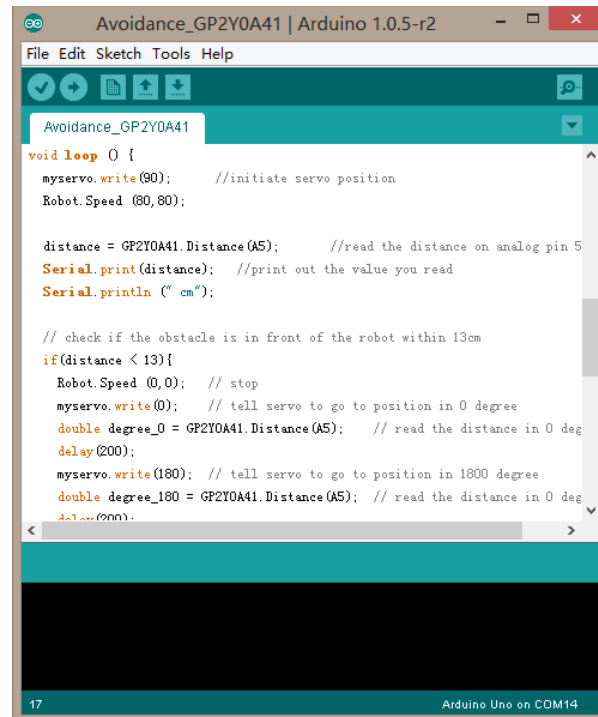
If you try to gently prod the servo, you'll discover that it'll refuse to budge. This is because we've configured the code to keep the servo at a 90° position; the servo is locked at this angle. With this in mind, be sure not to forcibly pull or tug at the servo! Next, remove the middle screw holding the sensors in place.

Remove the sensors attached to the servo. Place the sensors on the front of the car (shown in the picture below). After moving the sensors, re-screw the servo's middle screw.



Coding

Since the length of the code is relatively long, we won't delve into all the specifics. . Download and run the Arduino code "Avoidance_GP2Y0A41.ino."



```
void loop() {
  myservo.write(90); //initiate servo position
  Robot.Speed(80,80);

  distance = GP2Y0A41.Distance(A5); //read the distance on analog pin 5
  Serial.print(distance); //print out the value you read
  Serial.println(" cm");

  // check if the obstacle is in front of the robot within 13cm
  if(distance < 13){
    Robot.Speed(0,0); // stop
    myservo.write(0); // tell servo to go to position in 0 degree
    double degree_0 = GP2Y0A41.Distance(A5); // read the distance in 0 deg
    delay(200);
    myservo.write(180); // tell servo to go to position in 1800 degree
    double degree_180 = GP2Y0A41.Distance(A5); // read the distance in 0 deg
    delay(200);
  }
}
```

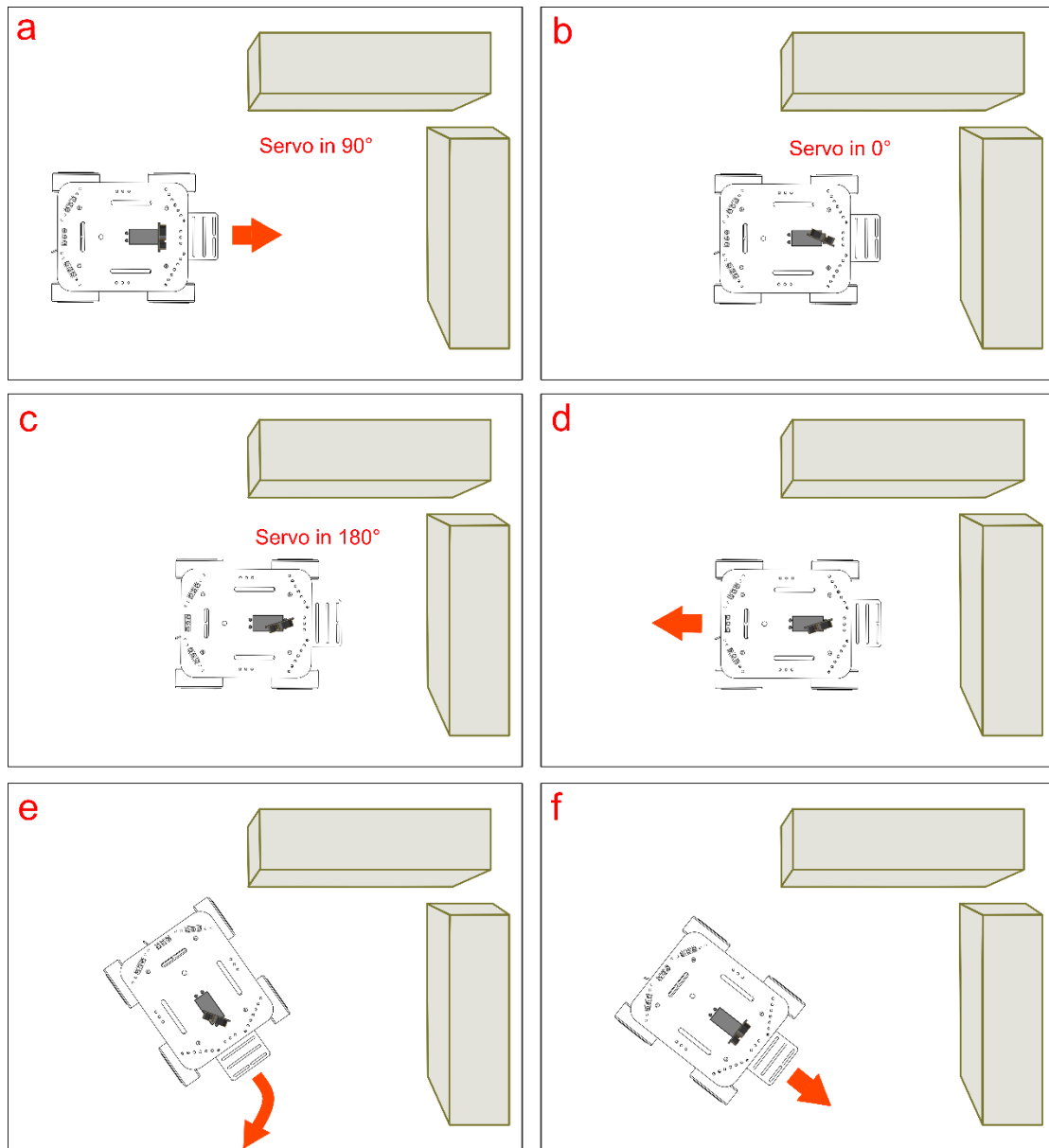
After downloading the code, upload it to your Romeo microcontroller. Once your car starts moving, you'll be able to see the obstacle avoidance sensors in action. If you feel like the sensor's distance detection isn't sufficiently strong, feel free to further adjust the sensors.

Next, we'll briefly go over the concepts behind obstacle avoidance — this way, you'll be better able to understand and modify the code on your own.

Obstacle avoidance: how it works

The following six diagrams (A, B, C, D, E, F) 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 stop; the servo will shift to 0° , survey its surroundings (diagram B), then shift to 180° and survey its surroundings once more (diagram C). The sensors will process the readings from the 0° and 180° angles to determine which direction the nearest obstacle is in — in the case of the below diagrams, that would be the obstacle in the 0° reading (diagram B).

In light of that reading, the car will reverse (diagram D) and then turn right (diagram E) until it senses no objects in front of it; then, the car will continue to move forward (diagram F).



Code Synopsis

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

Here, we'll be using two Arduino libraries — Servo and GP2Y041. The Servo library is used to control the servo, while the GP2Y041 library is used for the sensor's distance detection.

```
#include<Servo.h>
Servo myservo;    // create servo object to control a servo
```

In order to use the Servo library, we must declare `myservo` as an object in the void `setup()` portion of our code.

```
#include <GP2Y0A41.h>
InfraredSensor GP2Y0A41;
long distance;
```

We also have to declare `GP2Y0A41` as an object in order to initialize our sensor's distance detection. The `distance` variable is used to store the readings from our sensors.

We won't rehash the void `setup()` part of the code since we previously discussed this in the Basic Functions portion of this guide. Below, we'll discuss the code that relates to the servo.

```
myservo.attach(10);
```

This line of code is used to configure the pin/port responsible for the servo. Since we soldered our servo to digital port 10, our code here initializes digital port 10. If you wire your servos to a different pin/port (i.e. port 11) and change your code accordingly (`myservo.attach(11)`), that will also work.

The function is as follows:

```
myservo.attach(pin);
```

This function is used to initialize the servo pin.

After initializing the servo pin, we want to use the `Serial` function. What does `Serial` do? Simply put, it's a channel for Arduino to communicate with the computer. Arduino uses `Serial` to send digital data to the computer — the computer can also use `Serial` to send digital data back to Arduino. Here, we use the function to enable Arduino to send the sensor's digital readings to the computer.

When using the `Serial` function, be sure to place it in the void `setup()` portion of your code.

```
Serial.begin(9600);
```

The function is as follows:

```
Serial.begin(speed);
```

This function has one parameter, `speed`, which is used to set the baud rate. When using certain wireless modules, we need to apply baud rate settings. But if we just input a number using `Serial`, this doesn't pose any problem.

The above function should be placed in the `void loop()` portion of your code.

Let's take a look at some new functions.

```
myservo.write(90);
```

The function is as follows:

```
myservo.write(pos);
```

This function is used to configure the angle of the servo. The `pos` value corresponds to the angle that you want to fix the servo at (i.e. 0~180°).

We don't want to set a random value for this function — optimal values are determined by the type of servo used. Here, we use an 180° servo; thus, we can select any value between 0~180°. In this case, we'll input 90° as our value.

```
distance = GP2Y0A41.Distance(A5);
```

This function determines how far obstacles are from your car. We use analog pin 5 (the pin connected to our sensors) to read and detect distance. Then, we use the `distance` function to store this reading.

The function below:

```
GP2Y0A41.Distance(analogPin);
```

This function only has one parameter, `analogPin`. It's used to read the distance values given by our sensors.

If we want to directly read these values on your computer, we can use the following two lines of code:

```
Serial.print(distance);    //print out the value you read  
Serial.println (" cm");
```

`Serial.print` can be used in many different manners — for more info, check out the Terminology Manual or the official Arduino website.

Next, we'll be dealing with the code that determines object distance. The code sets out to determine how close an object is — here, we've set a parameter for how the car should move if an object is within 13 cm of the car or further than 13 cm from the car. To do this, we need to use the if...else function.

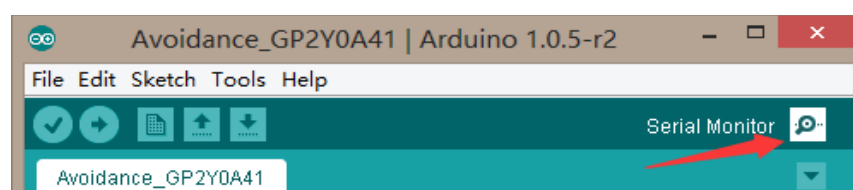
Let's check out the following to get a better sense of the logic behind obstacle avoidance.

```
if (Distance < 13) {  
  Robot stop;  
  Servo rotate to 0°;  
  Sensor reads distance;  
  Servo rotate to 180°;  
  Sensor reads distance;  
  if (distacne in 0° > distacne in 180°) {  
    Robot turn back;  
    Robot turn right;  
  }  
  else {  
    Robot turn back;  
    Robot turn left;  
  }  
}  
else {  
  Robot go forward;  
}
```

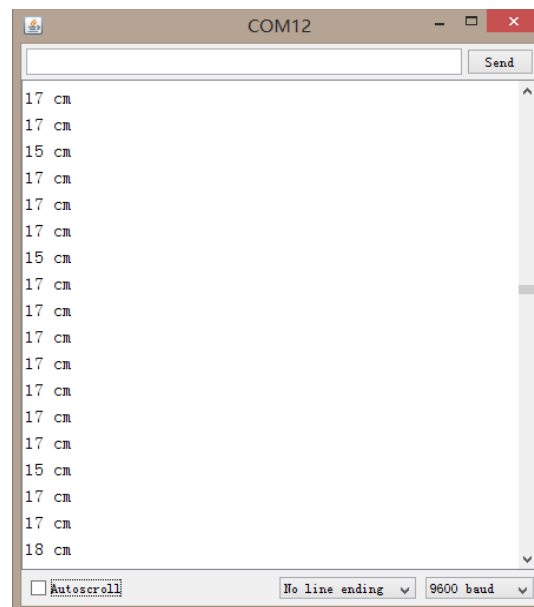
This code illustrates the principles in the “Obstacle avoidance: how it works” section of this guide. Simply put: if the object is detected less than 13 cm away from the sensors, then the car will stop. Then, the servo will rotate to a 0° and 180° angle, respectively. The sensors will process the readings from the 0° and 180° angles to determine which direction the nearest obstacle is in — after making that determination, that car will reverse, then turn left or right until it senses no obstacles in its path, whereupon it will move in a forward direction again.

How do we use Serial to observe the sensor's values?

As mentioned before, we can use the Serial function to see the readings given by the sensors. The Arduino IDE has a Serial Monitor which can be used to track those readings as they're being fed back to the computer.



After downloading the code and opening the Arduino IDE, click on the button labeled “Serial Button” as shown in the picture above. A box will appear. In the bottom right corner of that box is a drop-down menu that lets you select the baud rate. In selecting the baud rate, we want to be consistent with what we wrote in our code (i.e. `Serial.begin(9600)`). Since we wrote `Serial.begin(9600)` before, we want to select a 9600 baud rate in the drop-down menu.



Based on the values of your sensors’ readings, you can tweak and modify your code to your liking. We’ve set our code to detect and respond to objects at a distance of 13 cm, but you can adjust your code to a value of 17 cm or whatever you prefer.

Something to keep in mind: our sensors have a distance detection range of 4 to 30 cm; as such, make sure you don’t set your code to detect objects beyond 30 cm or below 4 cm.

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!