Welcome to the IoT Brewing Company!

Operational Technology for Cybersecurity Workbook 2

Prepared by:
Chuck Bales
Dr. Kristine Christensen

Developed in partnership with the National Cybersecurity Training & Education Center

Ready to Clock Back In?

Congratulations on successfully completing your IoT Orientation and first brewmaster IoT project!

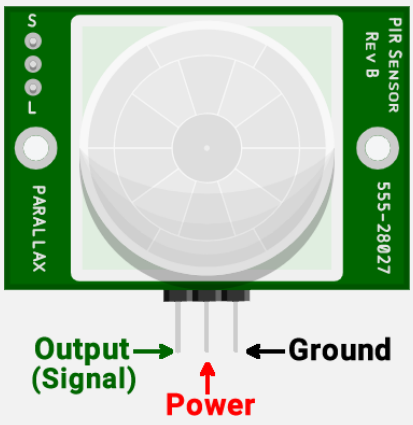
The embedded device that you implemented to open and close the valve on the vessel has greatly increased productivity and efficiency. Your boss saw how beneficial embedded systems and IoT can be within the brewing process and has asked you to continue your work in automating a whole process.

For the next project, the brewmaster would like for you to work with the packing and distribution department on an IoT solution for the beer packaging process. The brewery would like to create three conveyor lines for packaging the bottles. This will give ample time for the packaging staff to successfully place the bottles in the crates. To do this, you will need to divert the appropriate number of bottles down each line in succession. For example, every 24 bottles down the line will cause the conveyor diverter arm to divert the bottles from one packing station to one of the other two. To implement this solution, you will approach this in three project stages, each building on the last.

Please note: These exercises are normally accomplished using PLCs but the brewmaster wants to you to learn more about IoT, to keep costs low, and to benefit from the added functionality of an interconnected IoT system.

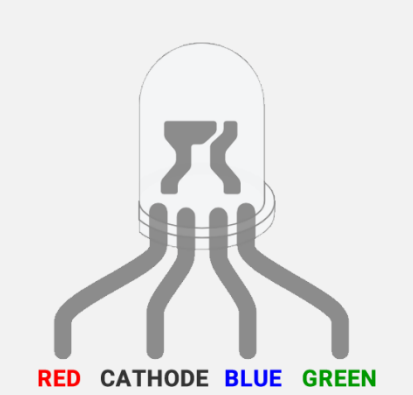
Brief Overview of Required Components

Each project will provide a list of components that you will need to add to your Tinkercad Circuit projects. The table below provides a brief description of each component.



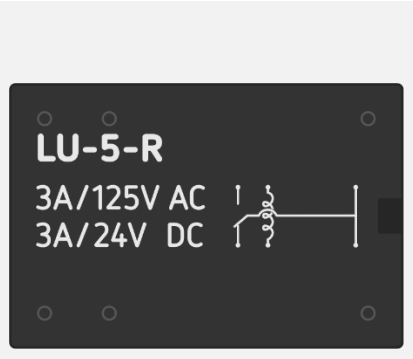
Passive InfraRed Motion Sensor (PIR)

- PIR Sensors are used to sense motion and can detect when an object is moved in or out of its range.
- These sensors have a 3-pin connection: one for signal, power, and ground. These pinout connections can vary depending on the PIR sensor you are using.
- The signal connects to a digital pin and the output will be set to 1 when sensor detects motion and 0 when there is no motion detected.
- Require little power, are inexpensive, and easy to use.



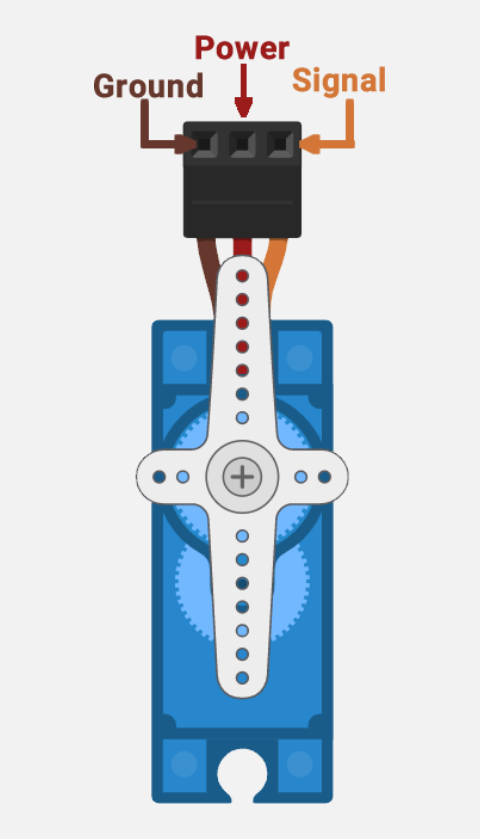
RGB (Red, Green, Blue) LED

- RGB LEDs have three LEDs inside the housing which allows you to mix colors by controlling the power / brightness of each of the three internal LEDs. Each LED takes a value from 0 to 255, which allows for the LED to display any RGB color.
- There are 4 pins, one for red, green, blue and the cathode pin which is connected to ground.
- Often used with Arduino's analogWrite() function to control the power to each of the LEDs.



Relay – SPDT (Single Pole Double Throw):

- A relay is an electrical switch that can be programmed and controlled by the Arduino microcontroller.
- It is used to control several circuits with one signal or when a high power circuit needs to be controlled by a low power signal.
- Referred to as an A/B switch because of its single input and ability to connect to and switch between two outputs.
- Has three connections – the input common, Normally Open (NO), and Normally Closed (NC).

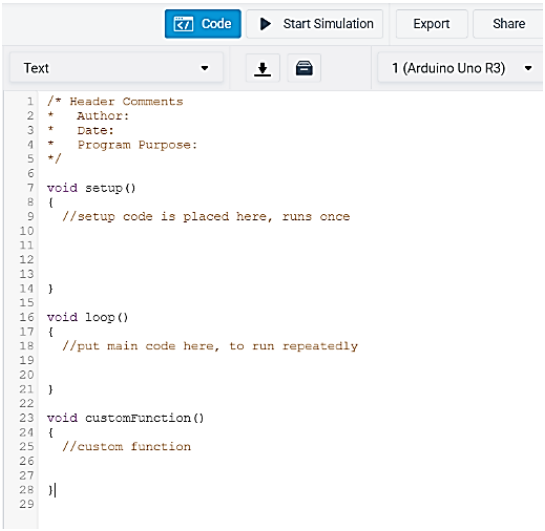


Micro Servo

- Micro servos are small servo motors.
- A servo motor actuator can move to a position accurately and are controlled by sending electrical pulses from the Arduino. These pulses tell the motor what position to move, making them ideal components for electronic applications.
- The shaft on servo motors can be positioned at various positions between 0 and 180 degrees.
- The servo motor has three connections: power, ground, and signal. The signal pin should be connected to the digital pin on the Arduino board.
- You will need to use the Servo library to allow the Arduino board to control the servo motor.

Pre-Project Activity 1: Advanced Arduino Programming

Using Arduino programming, a simplified version of C++, you were able to successfully complete the tasks assigned to you by the brewmaster.



Since you did so well with the tank project, the brewmaster has assigned new tasks which require knowledge of advanced programming concepts. You will need to understand and use logical operators, switch case statements, and even write your own custom functions. Each programming concept will be briefly discussed.

Logical Operators

Logical operators are used to control program flow and when decisions need to be made on multiple conditions. These are called compound conditional statements. The common logical operators include AND, OR, and NOT. Logical operators use Boolean logic and return either a true or false value.

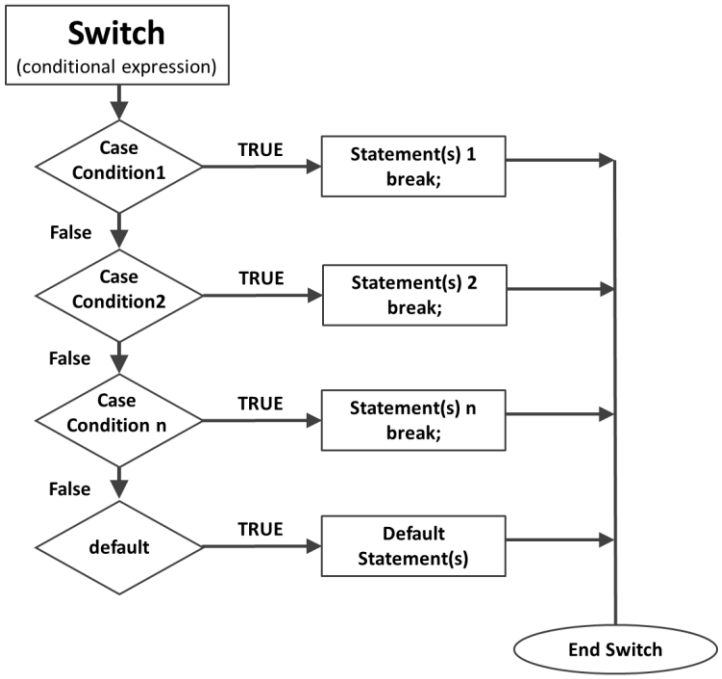
The table below provides a brief description and example of each of the operators.

Operator Name	Operator	Description	Example resulting in True when using x = 5 and y = 2
AND	&&	Returns true **only** when **both** conditions are true.	`(x > 1) && (y < 9)`
OR	||	Returns true if **either** of the two conditions are true. It also returns true if both conditions are true.	`(x > 1) || (y > 9)`
NOT	!	Results in a true if the operand is false. This reverses the value of the expression. If the condition is true than the NOT operator will return a value of false.	`!(x < 1)`

Switch Case Statements

Like `if` statements, the switch case is a conditional statement used to control the flow of programs and is used to specify when code should be executed after various conditions are met. The switch statement tests the value of a variable, located within parenthesis after the switch keyword, against a list of the values called **`cases.`** When a matching case is found, the statement(s) following the matched case is executed. The keyword **`break`** is typically included at the end of each case's statements and is used to break the program and exit the switch to avoid continuing the execution of the following expressions, known as "falling-through", until a break or the end of the switch statement is reached. When the **`default`** case is used, the statements following will be executed if none of the cases match the variable value. ***Note***: this type of conditional statement is only suitable for a limited number of cases.

The Switch Case Flow Diagram below illustrates how this conditional statement works. To the right of the diagram, you will see sample syntax and code for a Switch Case.



Switch/Case Syntax

```
switch (var) {
  case label:
    // statements
    break;
  case label:
    // statements
    break;
  default:
    // statements
    break;
}
```

Switch/Case Sample Code

```
int count = 2;
switch(count) {
 case 1:
      Serial.print("Count=1");
      break;
 case 2:
      Serial.print("Count=2");
      break;
 default:
      Serial.print("Default");
      break;
}
```

Output: `Count=2`

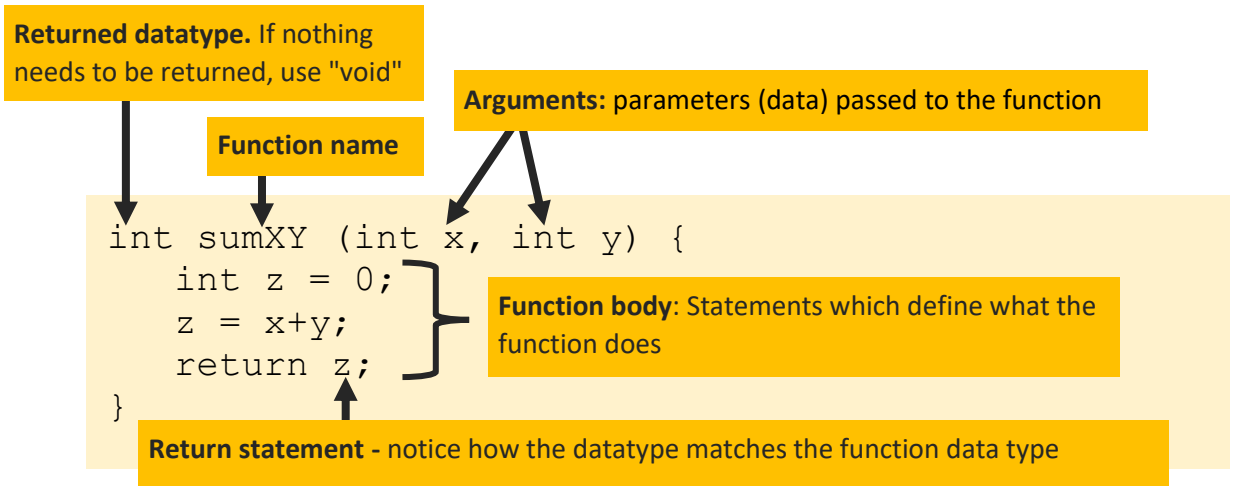
Functions and User Defined Functions

A **function** is a block of organized code that performs a defined task which the programmer can call in a program as well as return a value. Functions make your code reusable and are used when the same code needs to be run multiple times during a program's execution. Every Arduino sketch program has two special functions, the setup() and loop(). Arduino programing language has built-in functions that you can use to perform computations and control the Arduino board. For more information about the various functions available, check out the Arduino language reference by visiting the following website: https://www.arduino.cc/reference/en/

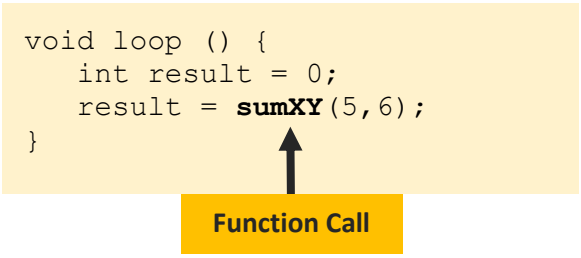
Advantages of Functions

Using functions provides the program with several advantages. Functions can keep a sketch program smaller by eliminating repetitive statements. If the code needs to be changed, it only needs to be modified in one function as opposed to multiple lines of code. Troubleshooting is also helped since the programmer will only need to look at one place to debug the functionality. Functions also provide modularity to a program so they can be used by other programs.

Anatomy of a Function



Example of calling the sumXY function in a program and passing arguments.



Writing Your Own Function (User Defined Functions)

1. **Determine the function type.**
    a. If the function does not need to return a value, then you will need to declare the function type as `void`. See the example below:

    example: void `functionName(){…}`

    b. If the function needs to return a value, then declare the datatype of the return value. The data type can be int, long, float, or Boolean.

    example: int `gradeFunction(){…}`

2. **Determine whether data needs to be passed to the function.** The data passed to the function are referred to as parameters. Parameters act like a placeholder, when the function is called, the values are passed to them. The parameters are included in parentheses following the function name. If you do not need to pass values to the function, then the parentheses will be empty.

    example: `int gradeFunction(int grade){…}`

3. **Write the statements that you want the function to perform.** The function body contains the statements that tells the function what to do. These statements are contained within the curly brackets.

    example:

```
int gradeFunction(int grade){
// statements
}
```

Calling a Function

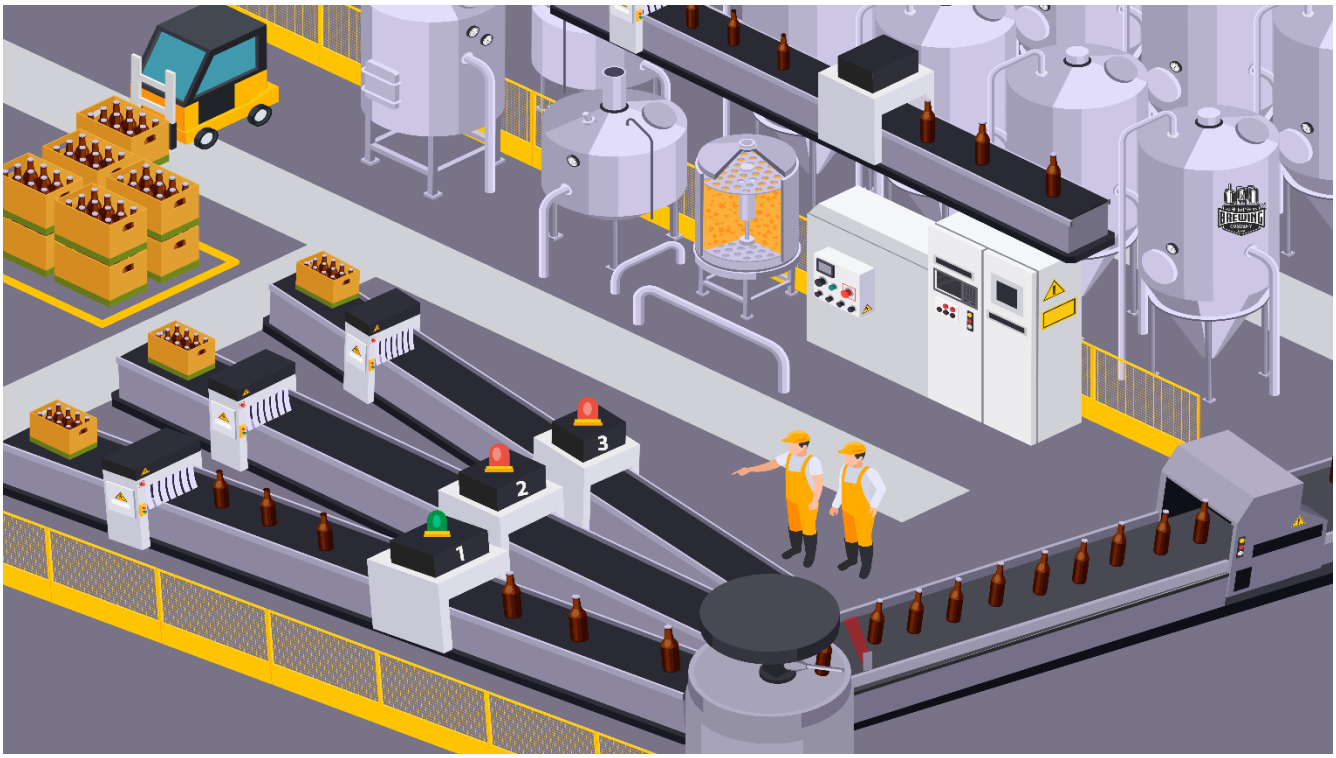
To call a function type its name in the program where you it to run, add parentheses after the function name, and include any parameters that the function requires inside the parentheses. When you call the function, the program leaves the 'main' program and executes the first line and all instructions inside the function. Once all statements have been executed, the program flow control leaves the function and returns to the next line after the function call.

example: `gradeFunction(90);`

Brewmaster Projects

The brewmaster has asked you to design IoT devices aimed at improving the brewing process. You will be designing and prototyping these devices using Tinkercad. Your job will be to develop new automated processes related to packaging the bottles after they are filled:

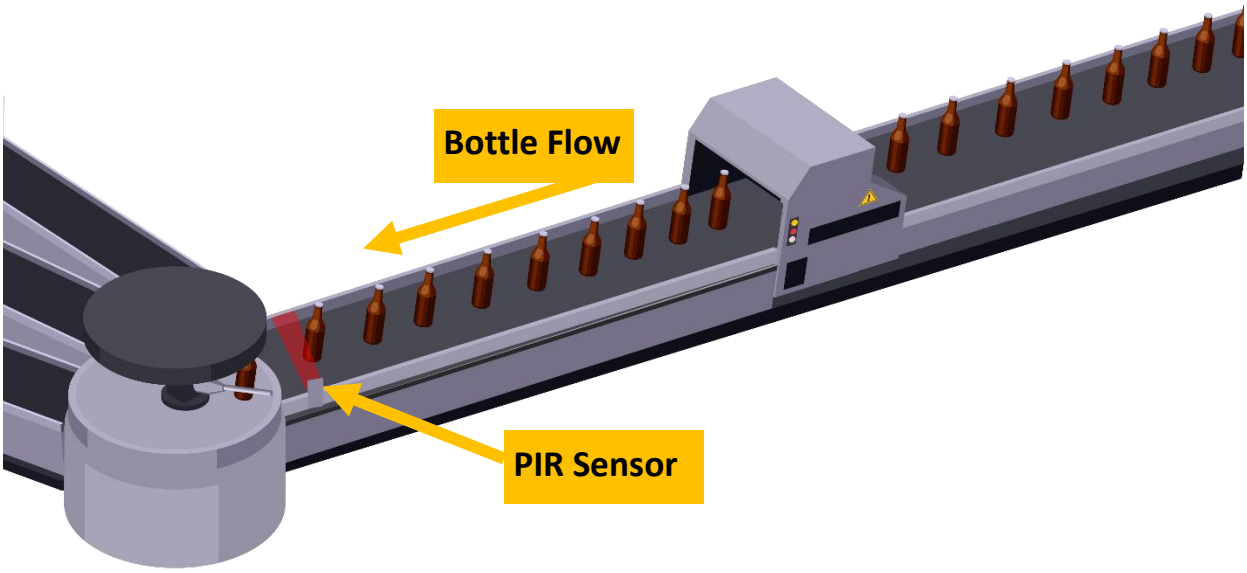
- **Brewmaster Project 1.** Use a PIR sensor to count the number of bottles traveling down the conveyor.
- **Brewmaster Project 2.** Add a servo to control the conveyor diverter arm to steer the bottles into one of three packaging stations.
- **Brewmaster Project 3.** Add a light over each packaging station to signal where bottles are being diverted.



Brewmaster Project 1

Instructions:

You have been asked by the brewmaster to use a sensor to keep track of the number of bottles that move down the line. This will help in the future as you will direct the bottles to the appropriate packaging station. The brewmaster has provided a PIR sensor for you to get started.



Brewmaster Project 1 Components, Wiring, and Code

Make It

For this project, you will need the following components.

- Arduino Uno
- Small Breadboard
- 1 x PIR Motion Sensor
- Jumper Wires

Input / Output Devices

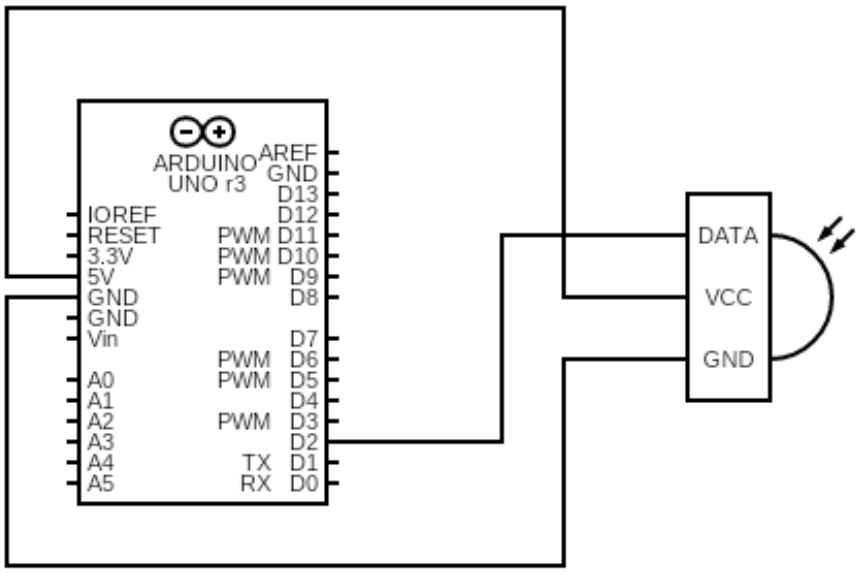
Input: PIR Sensor
Output: Serial Monitor

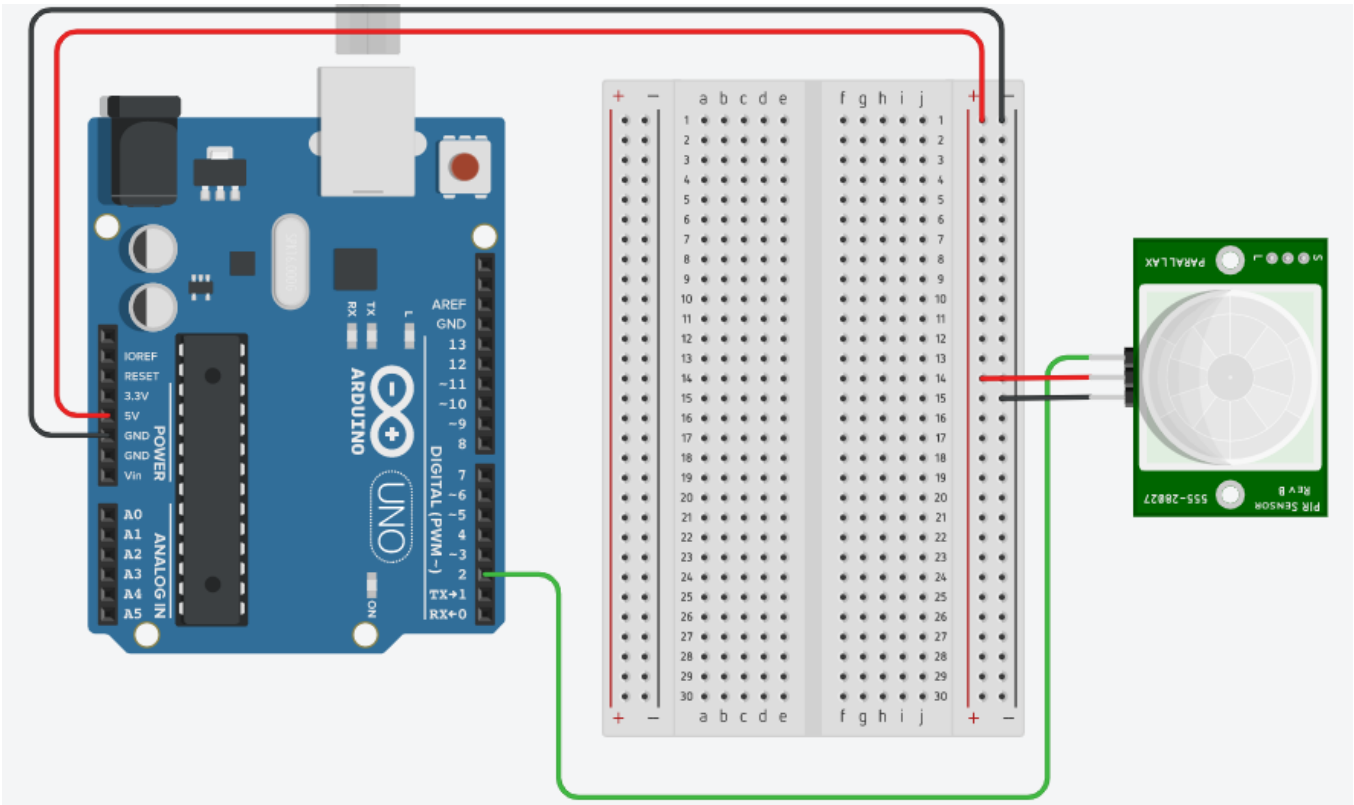
Helpful Wiring Notes:

- The green wire is the signal wire, the black wire is ground, and the red wire is for power.

Wire It

Use the schematic and circuit illustration below to wire this project in Tinkercad.





Code It

Below is the code you will need to add to the project in order for it to work. The program will use a PIR sensor to count the number of bottles passing it. Add comments to this code so that its purpose and functions can be clearly understood by other readers.

```
1  int bottleCount = 0;
2  int PIRstate = LOW;
3
4  int const PIRpin = 2;
5
6  void setup()
7  {
8     pinMode(PIRpin,INPUT);
9     Serial.begin(9600);
10 }
11
12 void loop()
13 {
14 if((PIRstate == LOW) &&(digitalRead(PIRpin) == HIGH))
15   {
16      bottleCount++;
17      PIRstate = HIGH;
18   }
19   else if((PIRstate == HIGH) && (digitalRead(PIRpin) == LOW))
20      PIRstate = LOW;
21
22   Serial.println(bottleCount);
23
24   delay(10);
25 }
```

Declare and initialize variables
Initialize the state of the sensor as inactivate.

Declare a constant for the digital pin (PIR sensor)

Set the I/O status of the digital pin
Initialize serial port for communication

If the sensor is inactive and a bottle enters the motion field

Increase the bottle count by one
Change the state of the sensor to active

Else if the sensor is active and there is no bottle in the field
Reset sensor state to inactive

Print bottle count to serial monitor

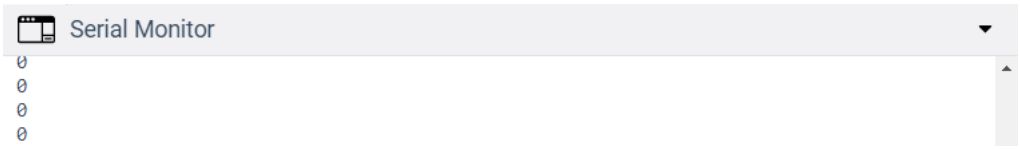
Delay for sensor to operate (10 milliseconds)

Working with the PIR Sensor

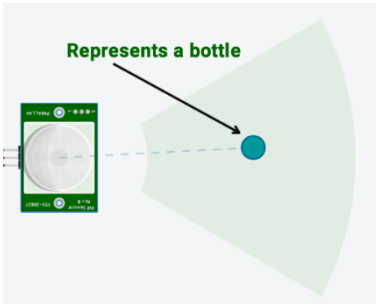
Events	PIRstate	PIRpin	bottleCount
1. No bottle	Sensor does not see anything (change state from HIGH to LOW)	No signal received by Arduino (LOW)	+0
2. Bottle moves into the field	Sensor sees something (change state from LOW to HIGH)	Signal is received by Arduino (HIGH)	+1
3. Bottle moves out of field	Sensor does not see anything (change state from HIGH to LOW)	No signal received by Arduino (LOW)	+0
4. Return to Step 1			

Test It

Once the wiring has been completed and the code has been successfully entered you are ready to start the simulation. To test your code, make sure that you have the Code window open and click on the Serial Monitor so that you can see the PIR motion sensor output.



Run the simulation and click on the PIR sensor in Tinkercad. You will see a circle, representing a bottle, in a green cone (range) in front of the sensor. Each time you drag the circle inside the cone it represents the bottle moving past the sensor. Click on the circle and drag it within the green cone. Each time it passes through the cone the bottle count will increase by one.



Note: If you do not see the circle or cone in front of the PIR sensor you may need to zoom in or out to resize the design space.

How did it go?

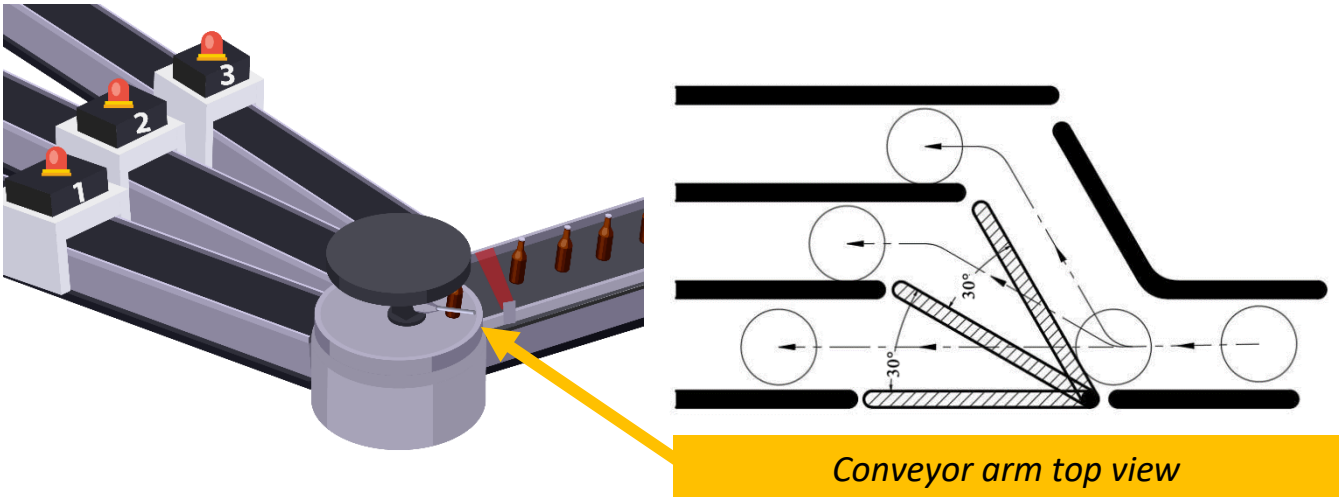
Congratulations, you now know how to count the number of bottles that pass the PIR motion sensor using an Arduino and programming. Save your work and now you are ready to proceed to Project 2.

Notes

Brewmaster Project 2

Instructions:

The brewmaster would now like you to control the conveyor diverter arm to send the appropriate number of bottles to one of the three packaging stations. Since you've used the PIR sensor to count bottles you can take advantage of it and implement a servo to control the diverter arm. Packaging size can vary from a 6, 12, or 24-pack of bottles. Currently, the brewery is packaging 24 bottles at a time. You will now need to control the servo in 60 degree increments every time 24 bottles pass the PIR sensor.



Conveyor arm top view

Brewmaster Project 2 Components, Wiring, and Code

Make It

Input / Output Devices

Input: PIR Motion Sensor
Output: Micro Servo

Add the following components to your circuit

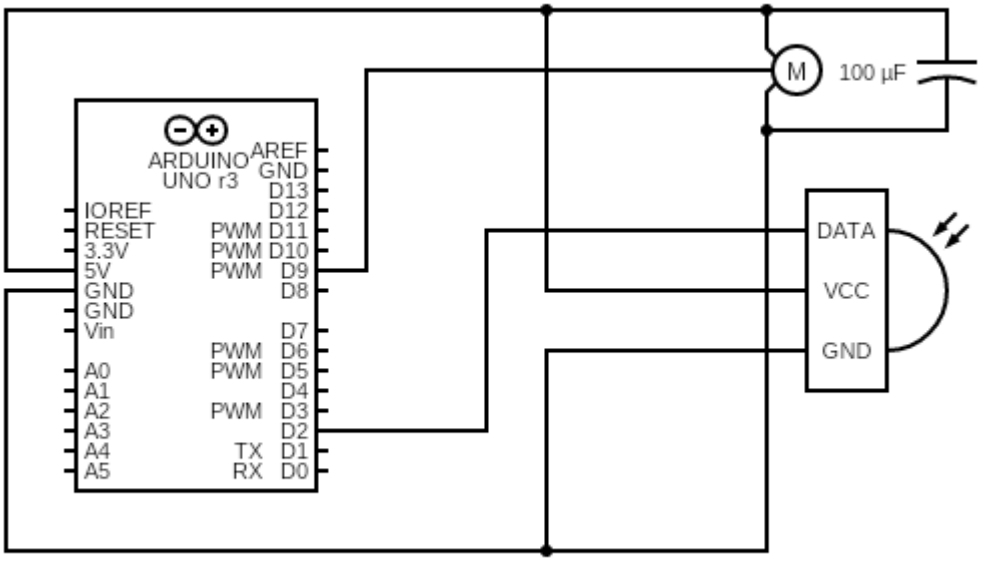
- 1 x Micro Servo
- 1 x 100 µF Capacitor
- Jumper wires

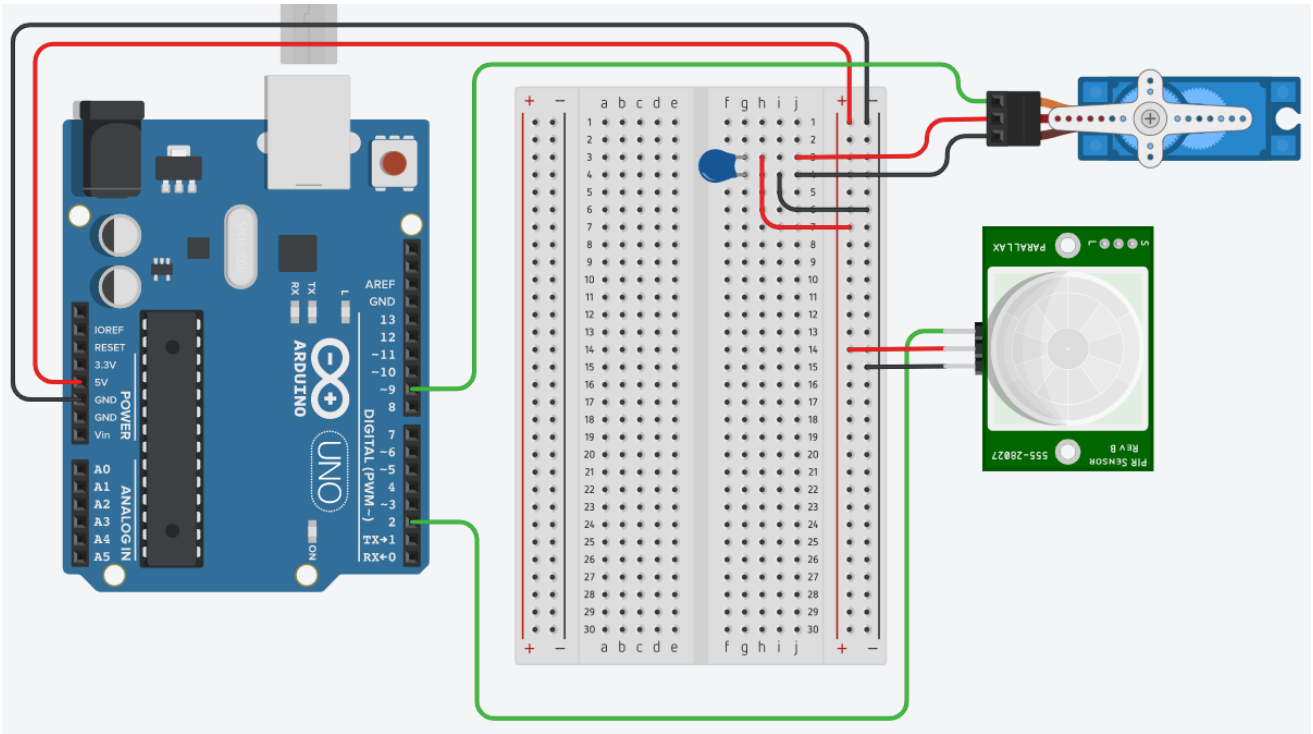
Helpful Wiring Notes:

- The orange wire is the signal wire, the brown wire is ground, and the red wire is for power.

Wire It

Use the schematic and circuit illustration below to wire to add the additional components to your breadboard.





Code It

Add the code highlighted **bold** below. The code additions will allow you to control the servo to change diverter arm to send the appropriate number of bottles to each packing station.

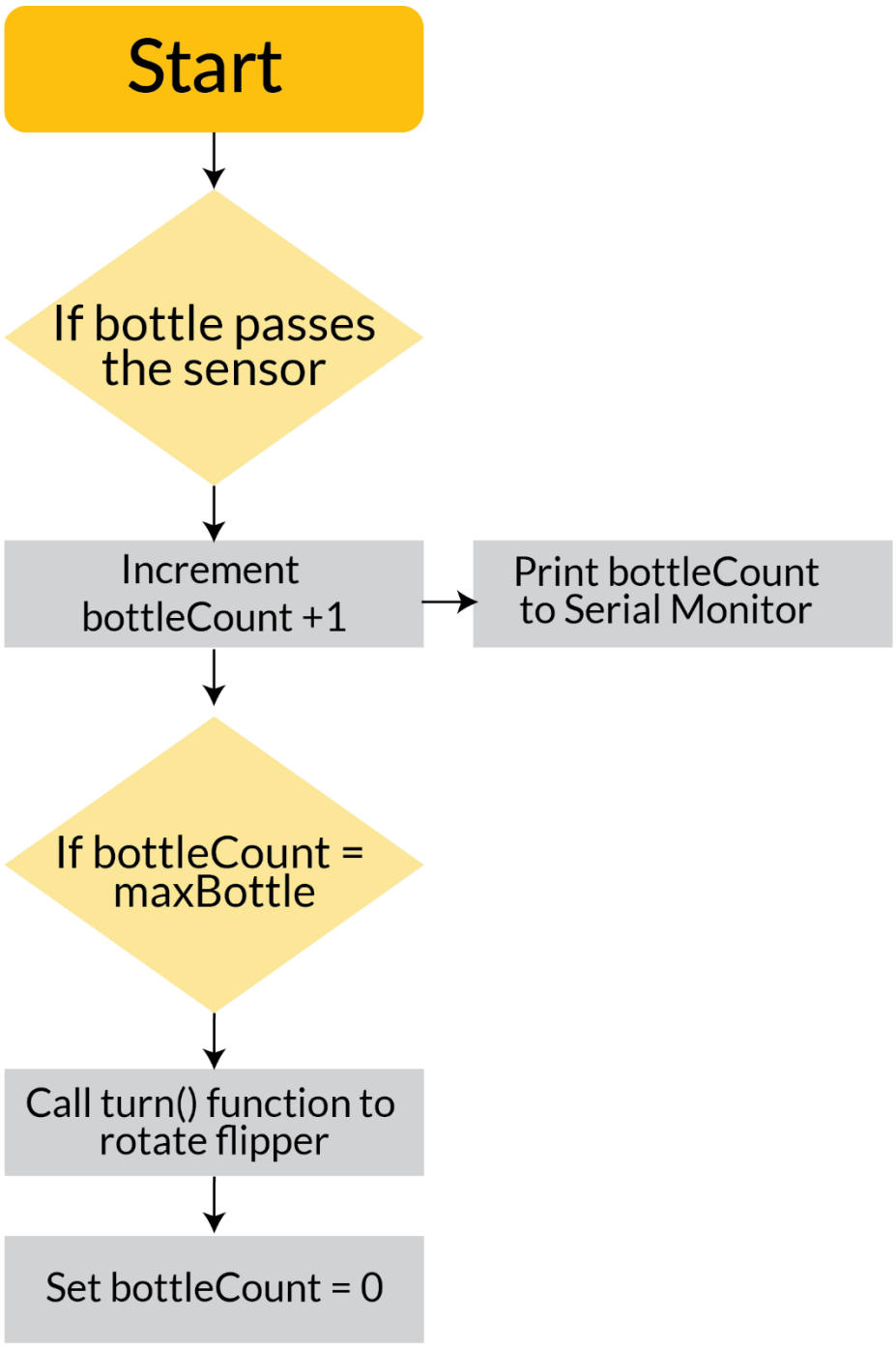
```
1  #include <Servo.h>                                   Import library for servo
2
3  int flipperPos = 0;                                  Declare and initialize variable for flipper
                                                        angle (degrees)
4  int bottleCount = 0;
5  int PIRstate = LOW;
6
7  int const PIRpin = 2;
8  int const flipperPin = 9;                            Declare constant for servo pin
9  int const maxBottle = 24;                            Declare constant for max bottle count per
                                                        chute
10
11 Servo flipper;                                       Create servo object named flipper
12
13 void setup()
14 {
15   flipper.attach(flipperPin);                        Attach servo to servo pin
16   pinMode(PIRpin,INPUT);
17   Serial.begin(9600);
18
19   flipper.write(0);                                  Initialize servo starting position (0
20 }                                                     degrees)
21
22 void loop()
23 {
24   if ((PIRstate == LOW) && (digitalRead(PIRpin)
   == HIGH))
25   {
26     bottleCount++;
27     PIRstate = HIGH;
28   }
29   else if ((PIRstate == HIGH) &&
   (digitalRead(PIRpin) == LOW))
30     PIRstate = LOW;
31
32   Serial.println(bottleCount);
33
34   if (bottleCount == maxBottle)                      If the maximum number of bottles per
35   {                                                  chute is reached
36     turn();                                          Call the turn() function to move the
                                                        conveyor arm
37     bottleCount = 0;                                 Reset the bottle count to zero
38   }
```

```
39
40    delay(10);
41  }
42
43  void turn()
44  {
45    if (flipperPos <= 60)
46      flipperPos += 30;
47    else
48      flipperPos = 0;
49
50    flipper.write(flipperPos);
51  }
```

Turn() function – this function will rotate the flipper in 30-degree increments (0 degrees, 30 deg, 60 deg)
If the flipper is less than or equal to 30 degrees
Rotate servo arm 30 degrees

Else if the flipper is greater than 60 degrees reset servo to start position (0 degrees)
Move the servo arm to the current flipper angle
Return back to the main loop of the program

Program Flow



Test It

Once the additional components and code have been added, test to see whether the servo moves after 3 bottles pass the PIR motion sensor. Hint: change the variable value of maxbottle to 3.

How did it go?

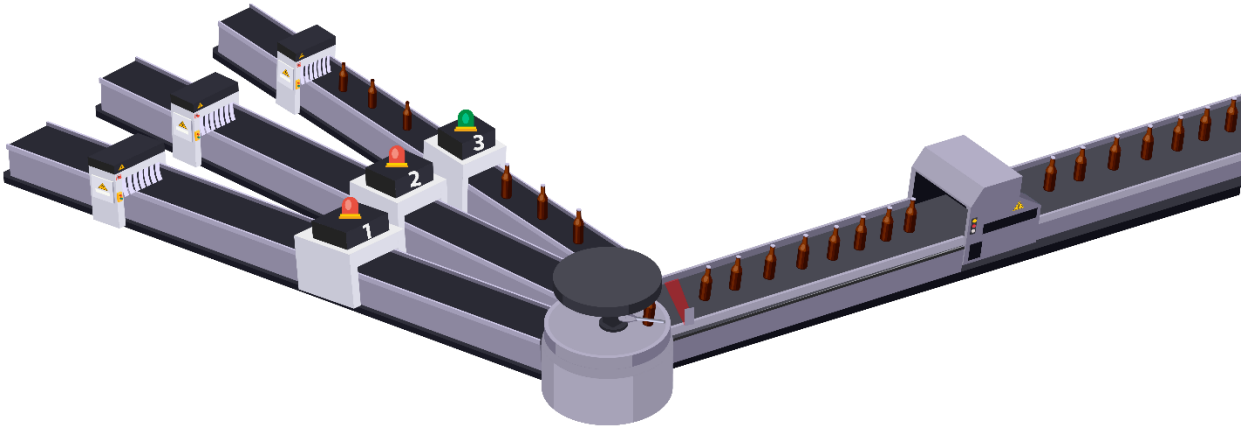
Congratulations, you now know how to divert the bottles to each packing station based on count. Save your work and now you are ready to proceed to Project 3.

Notes

Brewmaster Project 3

Instructions:

As a safety feature, the brewmaster would like to add a light above each packaging station which will indicate which station the bottles are being diverted to. The lights should turn green when bottles are being diverted to that station and red when they are not. For this, you will use an RGB LED which can be set to green or red.



Brewmaster Project 3 Components, Wiring, and Code

Make It

Add the following component to your circuit:

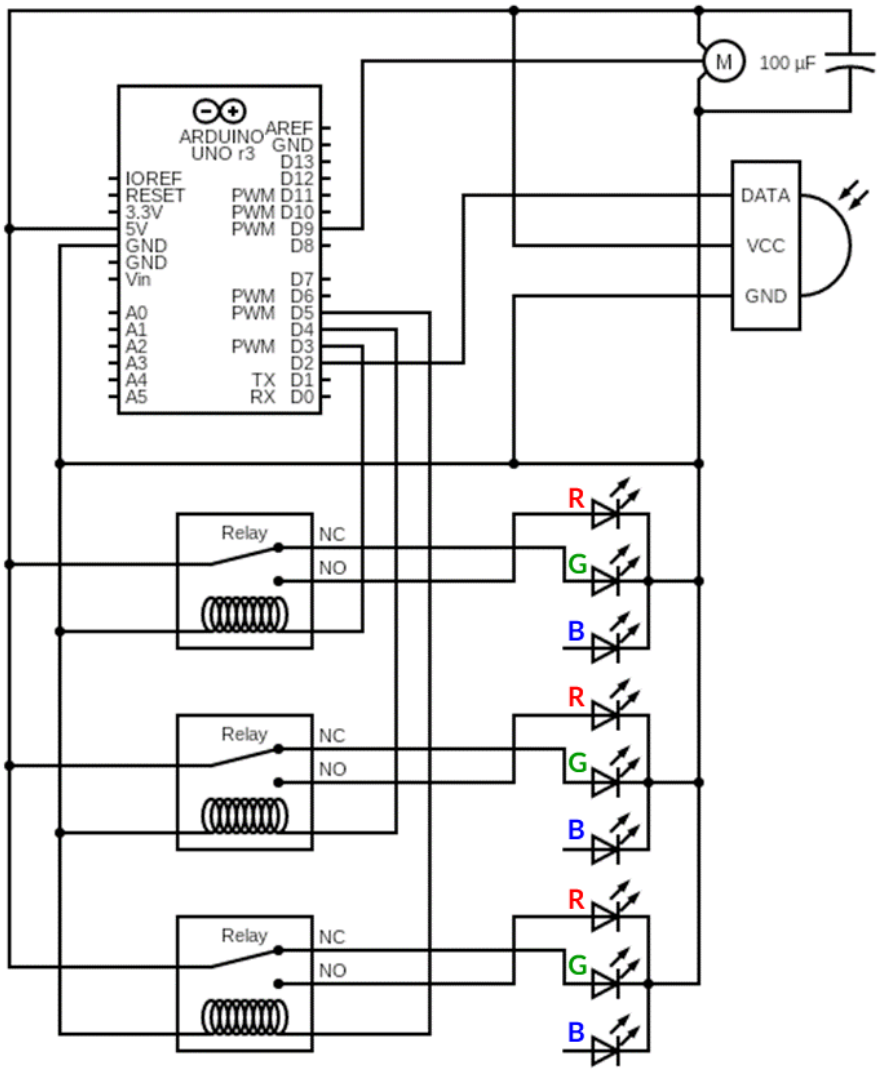
- 1 x Ultrasonic Distance Sensor (Parallax Ping)
- Jumper wires

Input / Output Devices

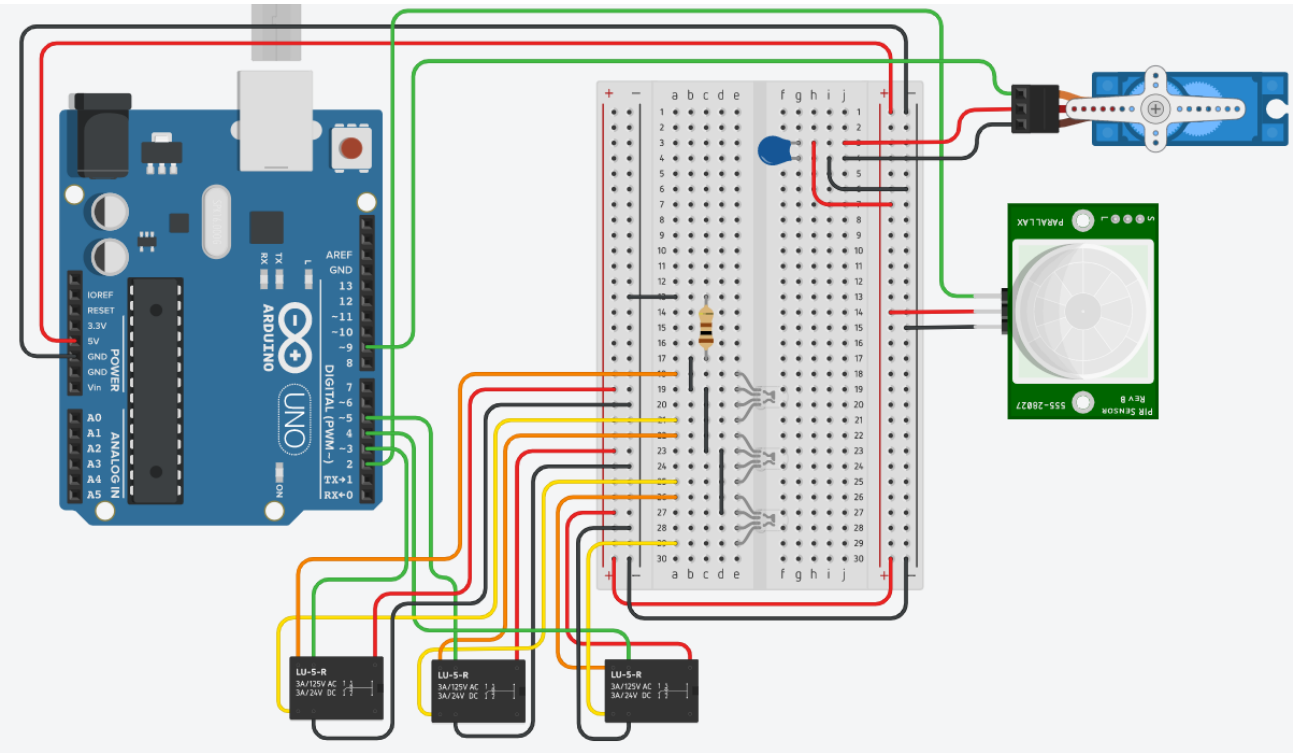
Input: Ultrasonic distance sensor, pushbutton
Output: RGB LED, Micro Servo

Wire It

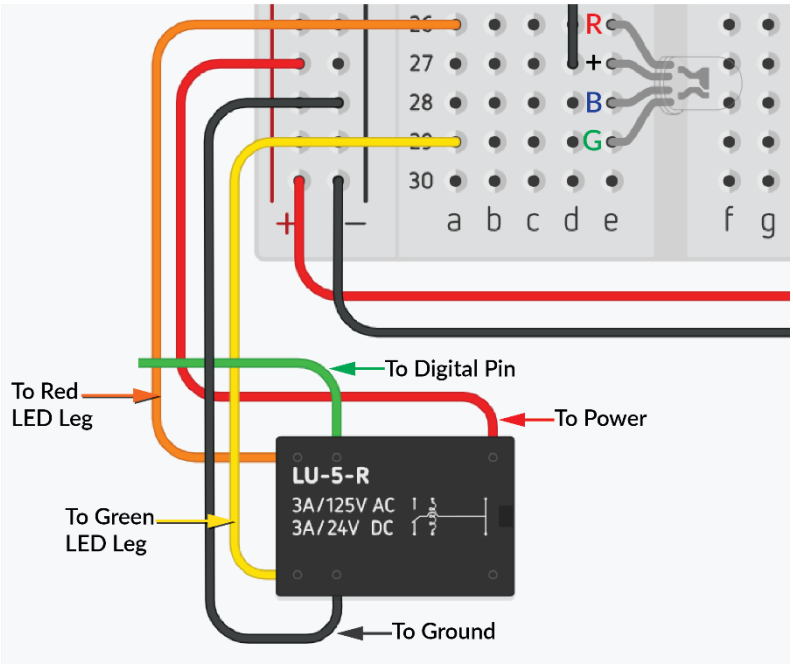
Use the schematic and circuit illustration below to wire to add the additional components to your breadboard.



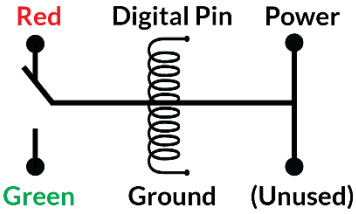
Please Note: The RGB LED Pin for the blue color is not connected in this circuit since we are not using a blue LED color.



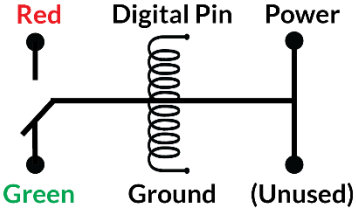
A Closer Look: Relay Illustration



Activate Digital Pin (High) to Close Switch and Turn Red LED On



Deactivate Digital Pin (Low) to Open Switch and Turn Green LED on



Code It

Add the highlighted code. The code changes the color of the RGB LED for each of the three packing stations. The LED will be green when bottles are being diverted to a packing station and red when they are not.

```
#include <Servo.h>

int flipperPos = 0;
int bottleCount = 0;
int PIRstate = LOW;

int const PIRpin = 2;
int const flipperPin = 9;
Int const maxBottle = 3;

int const rightLED = 3;
int const centerLED = 4;
int const leftLED = 5;

Servo flipper;

void setup()
{
  flipper.attach(flipperPin);
  pinMode(PIRpin,INPUT);

  pinMode(rightLED,OUTPUT);
  pinMode(centerLED,OUTPUT);
  pinMode(leftLED,OUTPUT);
  Serial.begin(9600);

  flipper.write(0);
  color();
}

void loop()
{
 if ((PIRstate == LOW) &&(digitalRead(PIRpin) == HIGH))
  {
    bottleCount++;
    PIRstate = HIGH;
  }
  else if((PIRstate == HIGH)&&(digitalRead(PIRpin) ==
LOW))
    PIRstate = LOW;

  Serial.print(flipperPos);
  Serial.print(" ");
  Serial.println(bottleCount);
```

Declare constants for RGB LED pins

Set the I/O status of the digital pins for the RGB LEDs

Call the color() function

```
44
45    if (bottleCount == maxBottle)
46    {
47      turn();
48      color();
49      bottleCount = 0;
50    }
51
52    delay(10);
53  }
54
55  void turn()
56  {
57    if (flipperPos <= 60)
58      flipperPos += 30;
59    Else
60      flipperPos = 0;
61
62    flipper.write(flipperPos);
63  }
64
65  void color()
66  {
67    digitalWrite(rightLED , HIGH);
68    digitalWrite(centerLED , HIGH);
69    digitalWrite(leftLED , HIGH);
70
71    switch (flipperPos)
72    {
73      case 0:
74        digitalWrite(rightLED ,LOW);
75      break;
76
77      case 30:
78        digitalWrite(centerLED ,LOW);
      break;

      case 60:
        digitalWrite(leftLED ,LOW);
      break;
    }
  }
```

If the maximum number of bottles per chute is reached

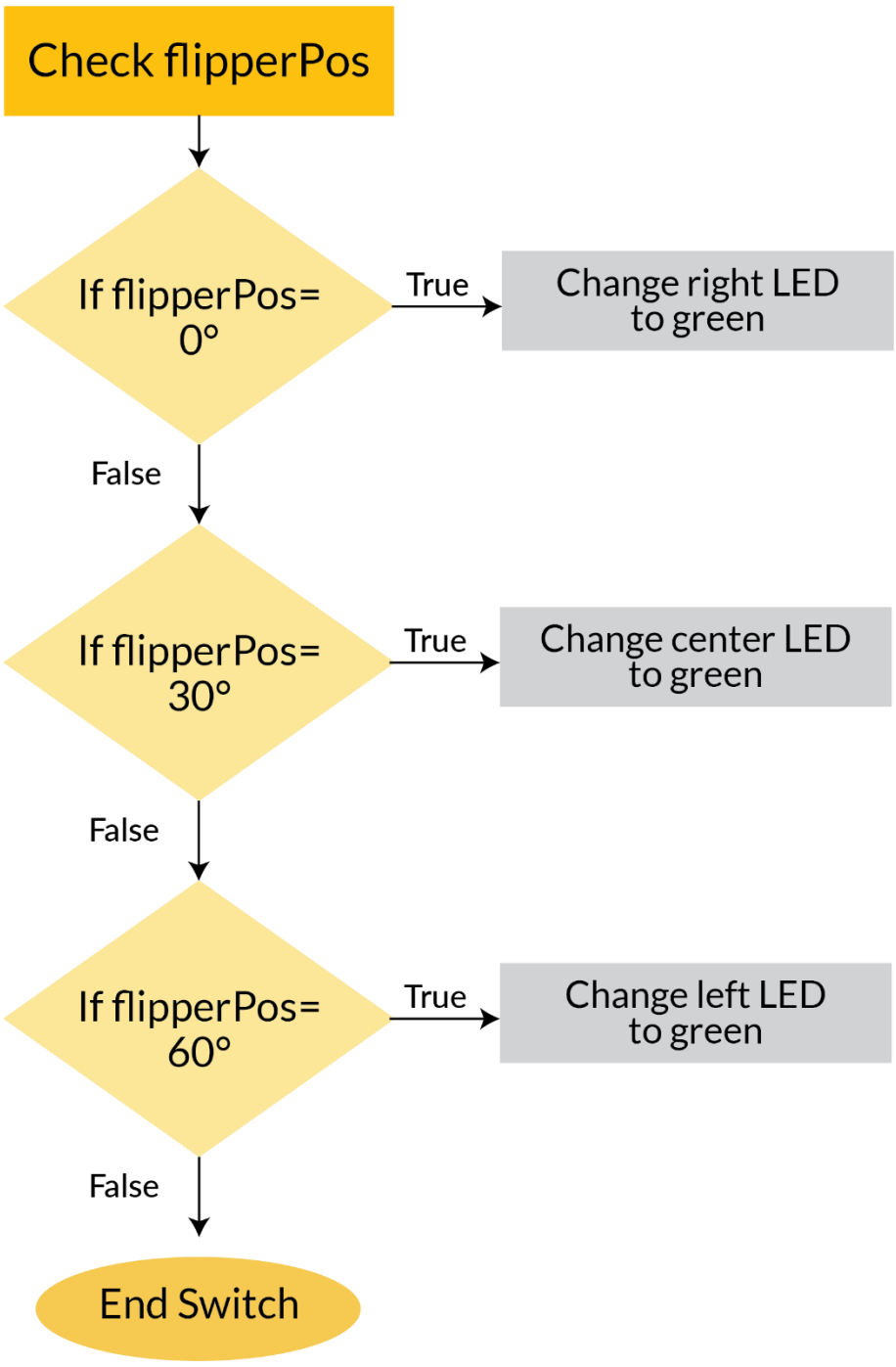
Call the color() function

Color() function – this function will alternate the color of the RGB LEDs based on the angle of the flipper
Sets all LEDS to RED

Sets correct LED to GREEN by

If the flipper angle is 30 degrees turn off the

Color Function Flow



Test It

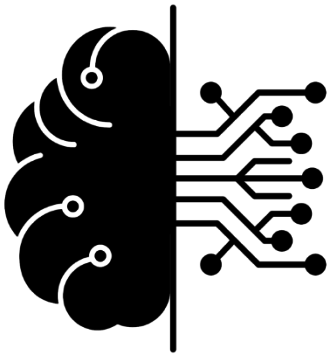
Once the additional components and code have been added, test to see whether the LEDs change color to red or green to signal where the bottles are being diverted.

How did it go?

Congratulations, you used relays to control the color of the RGB LED indicator so that workers will know which packing station bottles are being directed.

Notes

Looking for Additional Challenges?



Project 1: Add a serial output to calculate the bottle speed.

Project 2: Add a serial output showing which chute is active.

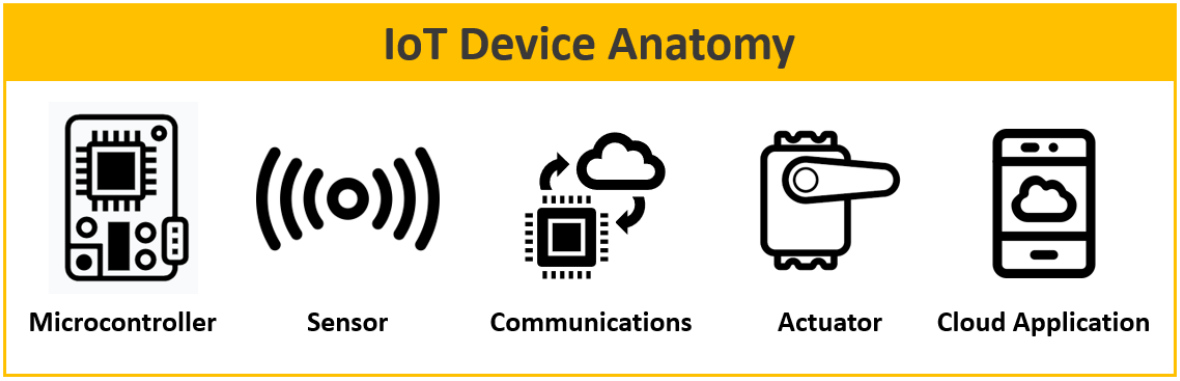
Project 3: Add additional sensors and actuators to improve the process.

Making the Brewmaster Projects Internet of Things Devices

Tinkercad is limited in its simulation capabilities and does not provide any devices to simulate an IoT device. However, there are only minor changes and adjustments that need to be undertaken to create fully IoT devices from the Brewmaster Projects. To make the Brewmaster Projects true Internet of Things devices requires the following modifications:

1. Switch the Arduino Uno microcontroller with a board capable of communicating with a network such as the Arduino Nano 33, Arduino MKR1000, Arduino MKR WiFi1010, or similar. In some cases, where the power of the IoT board is limited to 3.3 V, slight alteration to the circuit may be necessary such as changing the resistor values.
2. Utilize an IoT cloud service, such as the Arduino IoT Cloud, to create widgets or apps to interface with the IoT device. The button and LED used in the Brewmaster Projects may be duplicated on the IoT cloud service so that activations can be performed, and alerts can be visualized, via the service page or app.
3. Add programmatic elements to initiate communication with a webpage or IoT cloud service.

For more information visit the Arduino IoT Cloud homepage: https://www.arduino.cc/en/IoT/HomePage



Glossary of Terms

Actuator	A type of component that changes energy into motion. Motors are a type of electrical actuator.
Analog	Something that can continuously vary over time.
Anode	The positive end of a diode (remember that an LED is a type of diode).
Arduino	An open-source programmable microcontroller designed to allow users to prototype electronics projects quickly and easily.
Boolean	A datatype that indicates something binary, such as *on* or *off*, *1* or *0*.
Breadboard	A prototyping platform that allows you to build electronic circuits.
Capacitor	A component that can hold an electrical charge.
Cathode	The negative end of a diode.
Comments	Statements that are used to help others understand the purpose of your code.
Constants	A named identifier that cannot change its value in a program.
Circuit	A circular path from a power supply, through a load, and then back again to the other end of the power supply. Current flows in a circuit only if it is closed, that is, if the outgoing and return path are both uninterrupted (or closed). If either path is interrupted (or open) then current will not flow through the circuit.
Digital	A system that deals with discrete values (typically 1s & 0s)
Function	A block of code that executes a specific task.
Internet of Things	A network of connected devices enabling them to communicate over the internet.
Library	A software extension of the Arduino API that expands the functionality of a program.
Light emitting diode (LED)	A type of diode that lights up when electricity passes through it. LEDs are polarized components that only allow electricity to flow through them in one direction.
Microcontrollers	The brains of the Arduino, this is a small computer that you will program to listen for, process, and display information.

Ohms (Ω)	Unit of measurement of resistance. Represented by the omega symbol.
Polarized	The leads of polarized components (e.g. LEDs or capacitors) have different functions, and thus must be connected the right way. Polarized components connected the wrong way might not work, might be damaged, or might damage other parts of your circuit. Non-polarized components (e.g. resistors) can be connected either way.
Prototyping	An initial stage of product development in which a working model is constructed and tested before the final product is manufactured.
Pull-down Resistor	A pull-down resistor (or pull-up resistor) is a resistor used to ensure a known state for a digital signal. It is typically used with buttons and switches to ensure an errant high state is not read when the button or switch is not pressed.
Resistor	A measure of how efficiently a material will conduct electricity
Sensor	A component that measures one form of energy (like light or heat or mechanical energy) and converts it to voltage or current.
Servo	Servo motors are rotary actuators that can move to a position accurately and are controlled by sending electrical pulses from the Arduino.
Sketch	The term given to programs written in the Arduino IDE.
Switch	A component that can open or close an electrical circuit.
Ultrasonic Distance Sensor	The Ultrasonic Distance Sensor is an input sensor that measures the non-contact distance between moving or stationary items.
Valve	A device for controlling the passage of fluid or air.
Variables	A datatype that stores values which are likely to change as your program runs. A variable's type depends on the type of information you want to store, and the maximum size of the information.

* Many of the terms included in the Glossary were taken from the Arduino Glossary (https://www.arduino.cc/glossary/en/)

Resources & References

Arduino Resources

- Arduino.cc - https://www.arduino.cc/
- What is Arduino? - https://www.arduino.cc/en/Guide/Introduction
- Digital Pins - https://www.arduino.cc/en/Tutorial/DigitalPins
- Arduino IoT Cloud - https://www.arduino.cc/en/IoT/HomePage