Internet of Things

J Component

Trash Talker Using Bolt IoT and Arduino Uno

Things used in this project

Hardware components

	Bolt IoT Bolt WiFi Module	× 1
	Ultrasonic Sensor - HC-SR04 (Generic)	× 1
	Arduino UNO & Genuino UNO	× 1
	Jumper wires (generic)	× 7

Software apps and online services

BOLT	Bolt IoT Bolt Cloud
	Arduino IDE

17MCA0014, 17MCA0015

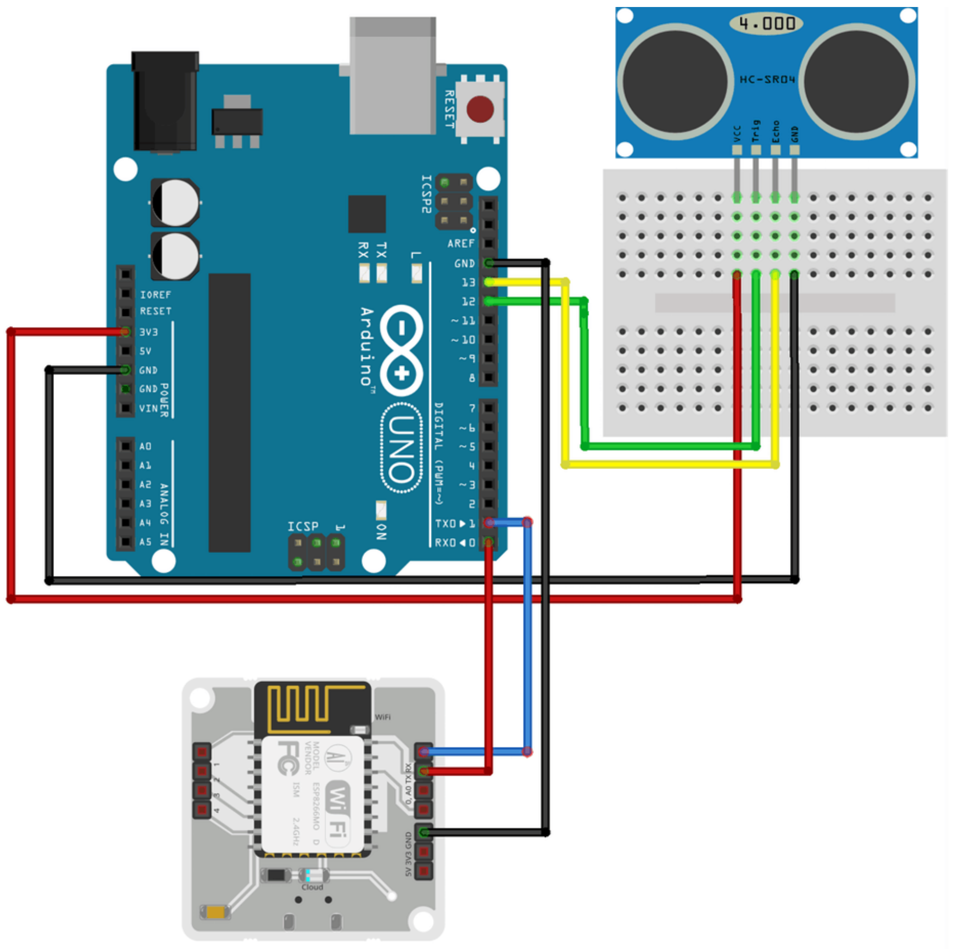
Overview

We often throw out the trash into the local cans for garbage collection once our main bin fills up. Except, we usually doesn't check our trash can often enough.

What he does check quite often is his WhatsApp, Facebook timeline and SMS's.

Hence we decided to come up with the Trash Talker system. It's essentially a setup which uses the popular Ultrasonic Sensor (HC-SR04), the Bolt Wi-Fi module, and an Arduino Uno to measure how much of the trash can has filled up. Once above a threshold, it sends an SMS to us to check the trashcan and hence help save the olfactory systems of the entire house.

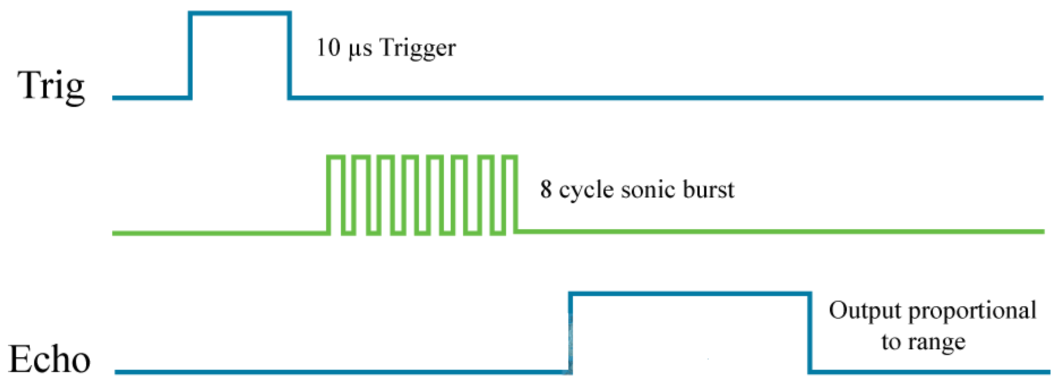
Hardware Setup



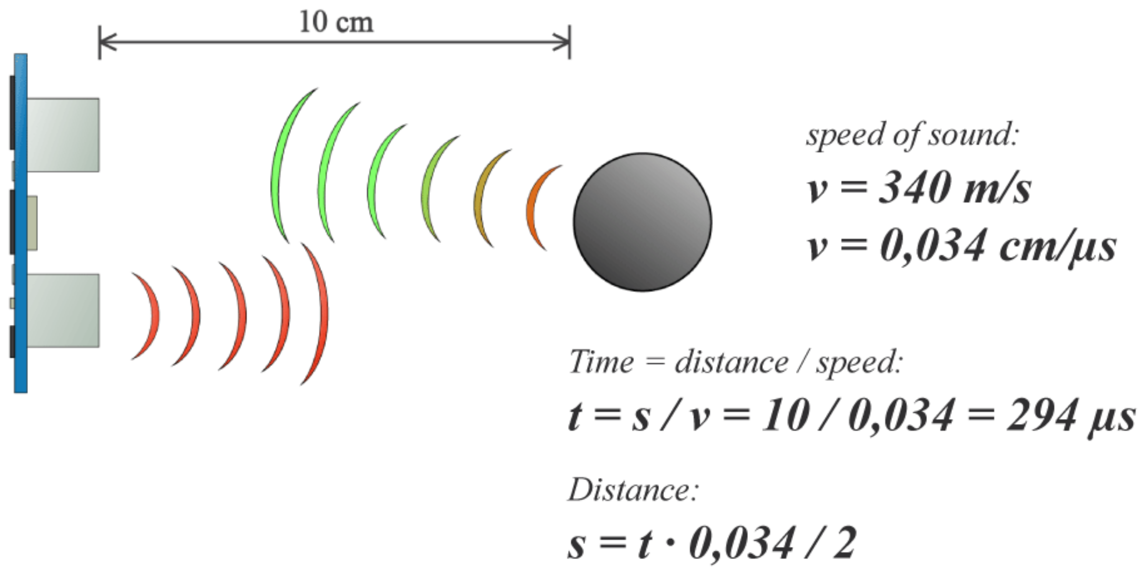
It emits an ultrasound at 40 000 Hz which travels through the air and if there is an object or obstacle on its path It will bounce back to the module. Considering the travel time and the speed of the sound you can calculate the distance.

The HC-SR04 Ultrasonic Module has 4 pins, Ground, VCC, Trig and Echo. The Ground and the VCC pins of the module needs to be connected to the Ground and the 5 volts pins on the Arduino Board respectively and the trig and echo pins to any Digital I/O pin on the Arduino Board.

In order to generate the ultrasound you need to set the Trig on a High State for 10 µs. That will send out an 8 cycle sonic burst which will travel at the speed sound and it will be received in the Echo pin. The Echo pin will output the time in microseconds the sound wave traveled.



For example, if the object is 10 cm away from the sensor, and the speed of the sound is 340 m/s or 0.034 cm/µs the sound wave will need to travel about 294 u seconds. But what you will get from the Echo pin will be double that number because the sound wave needs to travel forward and bounce backward. So in order to get the distance in cm we need to multiply the received travel time value from the echo pin by 0.034 and divide it by 2.



Bolt Wi-Fi Module

Bolt is the unique IoT platform which is capable of connecting and controlling devices over a network easily. The main heart of bolt hardware is the ESP8266 Wi-Fi module manufactured by Espressif. Espressif Systems' Smart Connectivity Platform (ESCP) is a set of high performance, high integration wireless SOCs, designed for space and power constrained mobile platform designers. It provides unsurpassed ability to embed Wi-Fi capabilities within other systems, or to function as a standalone application and minimal space requirement.

ESP8266 on the Bolt

ESP8266 handles the following functionalities:

- Connecting to Wi-Fi: The ESP8266 has a Station mode in which it can connect to most Wi-Fi networks and almost all home and office Wi-Fi networks(2.4 GHz)
- Acting as a Wi-Fi router: When it is not connected to any Wi-Fi network it will host its own Wi-Fi hotspot to which users can connect easily.
- Command Handler: All commands sent over web are first processed by the firmware running on ESP8266 module and then executed as per need.

The CPU inside runs at a frequency of 80MHz.

The ESP has an operating voltage of 3.3V and hence requires an on-board regulator for operation (the three pin IC seen below). For power ratings of ESP please refer to datasheet mentioned above.

The ESP controls all Input/output functionality of Bolt as well as UART communications.

Bolt module GPIO

The GPIO pins present on Bolt are directly controlled by the ESP present on-board and thus have the following features: - Digital Pins: Pins 1, 2, 3, 4 (on the left) along with 0 (next to A0 on the right).

Note: All work on 3v3-volt logic level i.e. HIGH = 3.3v = Binary 1 and LOW = 0v = Binary 0

In Output mode, each pin can source a maximum current of 12mA but it is to be used at a max current sink or source at 10mA.

While using these pins as input (reading voltage values) an easy way to think of thresholds is that any voltage below 1.5 Volts is considered as LOW and any Input above 1.5 Volts is considered as HIGH.

Bolt PWM Pins

Pins 0, 1, 2, 3, 4 act as software PWM (Pulse Width Modulation) pins. You can give an output anywhere between 0-3.3V on these pins in steps of (3.3/256) = 0.012V approximately.

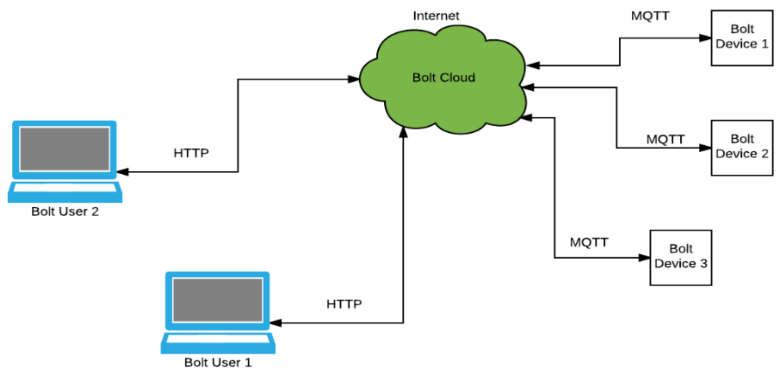
Analog Read Pin

A0 is the only Analog Read Pin. It can read any voltage between 0-1V with a least count of ~0.0009V. Thus, this is a 10 bit ADC (Analog to digital converter) with the resolution of 1/1023 = 0.0009

Understanding the Bolt Cloud Architecture

In the previous lesson, we created our account on Bolt Cloud. Before we link our Bolt device to our account, let us understand the Bolt Cloud Architecture, how it works, and why it is important.

The Bolt Cloud is one of the major component in providing the IoT capabilities to the Bolt device. All the Bolt devices connect to the Bolt Cloud out of the box. The Bolt devices are shipped with a firmware that helps it understand how to connect to the Bolt Cloud over the Internet.



The communication of Bolt devices with Bolt Cloud happens over the MQTT communication protocol. MQTT stands for Message Queue Telemetry Transport. But why do we need to have a protocol such as MQTT for communication when HTTP & HTTPS protocols are so popular and widely used for communication? Although these protocols are popular, the amount of overhead data that is sent over the Internet for managing the communication is quite a lot. Overhead data is the data which is sent along with the actual message/data which conveys the extra information required to understand the message/data sent. The overhead data varies from protocol to protocol. This is fine in case of systems such as mobile phones, laptops, desktop computers that have the hardware capabilities and the network capabilities to send the extra overhead data.

Most IoT devices and sensors contain limited processing capabilities and constrained Internet bandwidth. Due to these limitations, they send data over the Internet only when required and the data sent is very low in terms of bandwidth usage. Hence using protocols such as HTTP, HTTPS does not become feasible where the overhead data is more than the actual data itself. MQTT contains very low overhead and hence becomes ideal for IoT communication.

MQTT is a pub-sub messaging protocol. Pub refers to publishing and sub refers to subscribing. There is a central entity, in our case it is the Bolt Cloud that. All the Bolt devices connect to the Bolt Cloud and send the data to various channels by publishing the data on their unique channels. The Bolt devise also subscribe to channels so that they can receive commands coming from the Bolt users.

The Bolt Cloud users i.e. people like us communicate with the Bolt Cloud using the HTTPS communication protocol. We can use the Bolt Cloud dashboard to control and monitor our Bolt devices or use the Bolt Cloud APIs in case we want to by-pass the dashboard and access our Bolt devices via programs that we have written. We shall learn more about this in the next modules.

Bolt Cloud receives all the commands to control or request for sensor data from Bolt devices, and sends the commands to the Bolt device. The Bolt device executes the commands, and sends a response back to the Bolt Cloud which in turn forwards it to the user who initiated the command.

Software Setup

The Arduino Code linked in below measures the distance using the Ultrasonic sensor and thereafter sends it to the Bolt Wi-Fi module over serial communication.

A Python Script (running on a server or your PC, for instance) queries the Bolt Cloud for this distance value using the Bolt Python Library, which in turn is based on the Bolt open APIs for Serial Read.

The Python script then checks if the distance is less than a preset threshold (basically if the last banana peel is too high in the trash pile). In case the bin is full, an SMS alert is sent out using the Twillio SMS service.

Code

Trash Check Distance – Arduino
This is the code to send data from sensor over serial to Bolt.

```
#include <Ultrasonic.h>

Ultrasonic ultrasonic(12, 13);
int trigPin = 12;
int echoPin = 13;
long duration, dis_cms;

void setup()
{
        Serial.begin (9600);          //Serial Port begin
        pinMode(trigPin, OUTPUT);     //Define inputs and outputs
        pinMode(echoPin, INPUT);
}

void loop()
{
        digitalWrite(trigPin, LOW);

        delayMicroseconds(2);     // Give a short LOW pulse beforehand to ensure a clean HIGH pulse

        digitalWrite(trigPin, HIGH);

        delayMicroseconds(10);   // The sensor is triggered by a HIGH pulse of 10 or more ms.

        digitalWrite(trigPin, LOW);


        // Read the signal from the sensor: a HIGH pulse whose

        // duration is the time (in microseconds) from the sending

        // of the ping to the reception of its echo off of an object.

        pinMode(echoPin, INPUT);

        duration = pulseIn(echoPin, HIGH);


        dis_cms = (duration/2) / 29.1;  // convert the time into a distance

        Serial.print(dis_cms);

        Serial.println();

        delay(800);

}
```

Trash Alert – Python

This is the Python script you run on your PC to check the distance against a threshold and send SMS alerts.

Conf.py

```
SSID='AC68b9162b6eb2b03ba0f5355a83c357f3'
AUTH_TOKEN='d7575b911cc0e4472d22e5d3fe6f93f7'
FROM_NUMBER='+19044471677'
TO_NUMBER='+917001664146'
API_KEY='28de1d6c-70f5-4b76-8ae5-de0a1699b118'
Device_id='BOLT3730565'
access_key='965332396152270848-Swf6ygra13Zdsc53J6qTn9FR3SackmD'
access_secret='dBtaInvCBJ27ZsmgQe3mKDwZlaY9C8617wFYQqBJJZsG2'
consumer_key='sTQWsGj33P5Z03skHxrMpumIE'
consumer_secret='BwV96qpYFdai3Db7O7FaPaIssEBYVvHrefp2Eha2dfzmeUoI40'
```

Trash.py

```
import conf
from boltiot import Bolt, Sms                              #Import Sms and Bolt class from boltiot library
import json, time

garbage_full_limit = 5                        # the distance between device and garbage in dustbin in cm

API_KEY = conf. API_KEY
DEVICE_ID  = "conf. Device_id

SID = conf. SSID                                           # Credentials required to send SMS
AUTH_TOKEN = conf. AUTH_TOKEN
FROM_NUMBER = conf. FROM_NUMBER
TO_NUMBER = conf. TO_NUMBER

mybolt = Bolt(API_KEY, DEVICE_ID)                                    #Create object to fetch data
sms = Sms(SID, AUTH_TOKEN, TO_NUMBER, FROM_NUMBER)                   #Create object to send SMS
response = mybolt.serialRead('10')
print response
while True:
        response = mybolt.serialRead('10')  #Fetching the value from Arduino
        data = json.loads(response)
        garbage_value = data['value'].rstrip()
        print "Garbage level is", garbage_value
        if int(garbage_value) < garbage_full_limit:
             response = sms.send_sms('Hello Sayan, I am full- Trash Talker')
        time.sleep(200)
```

Twilio

Suppose your parent has allotted a task to you to check the water level in the plant daily and water them if required. Isn't it very boring and tiresome? What if you get an SMS on your mobile telling that water level in your plant is very less and need water immediately. Sounds Interesting right!! But how can it be done?

This can be achieved with the help of Bolt and a third-party company named 'Twilio'.

Twilio is a third-party SMS functionality provider. It is a cloud communications platform as a service (PaaS) company. Twilio allows software developers to programmatically make and receive phone calls and also send and receive text messages using its web service APIs.

Model