Tut-Tut, The IoT Rainfall Detector Progress Report

Team 21

CS 461 - Fall 2018

Jonathan Rohr, Shreyans Khunteta, Michael Gillett

February 21st, 2019

Abstract

Tut-Tut, the IoT Raindrop Detector, is a project under the domain of the OPEnS (Openly Published Environment Sensing) Lab at Oregon State University. Tut-Tut will be able to detect a drop of any size, get an idea of the raindrop mass from the intensity at which it falls, and detect the frequency of raindrops. After Tut-Tut gathers this data, it will be able to wirelessly transmit the information to a central server and store it in a database. This document will explore multiple possible technologies that can be used to gather, transmit, and store our data.

1 Project Purposes and Goals

The purpose of Tut-Tut is to provide metrics for understanding the intensity of rainfall as it is happening. We plan on using our sensor to measure how forceful the impact of the raindrops are, and the microcontroller's built-in timer to record the duration between those drops.
With these two parameters, we will place each recorded drop on a scale between 0 and 100, 0 being no rain at all and 100 being torrential downpour, and provide a key with ranges describing the rain in more familiar terms. For example: 0-15 would represent a mist, 15-30 would represent a light rain (jacket recommended), 30-45 would represent a medium rain (umbrella recommended), all the way up to cataclysmic conditions. A rating of 100 should not be reasonably attainable on our scale, so that we can represent any recorded value somewhere on the spectrum. We were hoping to introduce a browser UI system to represent this data in a colorful map or Twitter feed, but this will likely be cut out due to time constraints. That browser UI will be left to a future Capstone team. Such a browser UI could potentially have appealing visuals along with the OSU logo that will render the information much easier to read. However, at the moment, we will just publish the results to a Google Sheets using PushingBox.
The ultimate end goal use of Tut-Tut will be to, for example, put in farmer's fields. Farmers will be able to see how much rain their crops are getting in specific areas and thus know exactly how much water they should give their crops. This will greatly aid in water conservation which has become a moral and practical necessity as the effects of climate change worsen.

2 Current Progress

We have chosen the MEMS sensor due to how sensitive it is relative to the accelerometer and the piezoelectric transducer. In addition we were able to build a quite functional rig that can detect raindrops and then have them neatly slide off the disk. Our MEMS sensor can detect the force of the impacts and we have code that gives us the average amplitude for a specific duration of time. That same duration is used to calculate the frequency that raindrops impact the sensing area. Our microcontroller enters low-power mode when the sensor hasn't detected a drop in one minute. A drop hitting the sensing area will trigger an interrupt that will wake up the microcontroller so data can be gathered again. Our data gets sent to a Google Sheet every two minutes using a Pushing Box API Scenario. Essentially, the data is used as URL arguments for a get request to Google Sheets. A script in our Google Sheet will recognize the format of the request and will add the data to the sheet.

We made some 3D prints of disks and our casing for the electronic components by using a software called Autodesk Fusion360 after an unsuccessful attempt in using AutoCAD Architecture. After realizing that AutoCAD Architecture was a bit of a nightmare to use, we switched over to Autodesk Fusion360 on the advice of Professor Chet Udell. We learned how to use Autodesk Netfabb to check the mesh to make sure there were no errors in it, and then we used Cura Lulzbot to compile our meshes into a file, which we save to an SD card. This SD card is then put in the OPEnS Lab's TAZ 3D printer, where we select it and begin printing our designs.

3 What is Left To Do

We still have a few tasks ahead of us before Tut-Tut will function as intended. First, we plan to maybe 3D print another casing that will hold all the components we need- breadboard, microcontroller, battery, sensor, and all connecting wires. This casing would have a sensing area slanted at a 35-45 degree angle. We want to try the angled model to see if the sensing capability is different because in our lab tests we had a problem with water collecting on the surface of the level sensor. After we test both of our casings, we will choose the one that records data most accurately. Next, we need to finish setting up our microcontroller to send its collected data to our designated spreadsheet using the PushingBox API. This tool allows us to upload data to the specified document in real-time, so we can record and view the results as they are happening. We have the base code done, but rigorous testing needs to be completed to ensure that it will send data while in the field. We also need to connect our microcontroller to a battery and perform tests to make sure that all of our code functions properly when it is disconnected from a laptop. Once we have chosen a model, we will make adjustments as necessary to waterproof the casing and make the microUSB input accessible for charging the battery and making changes to the code. If we find ourselves with extra time, we could attempt to make a very simple UI for conveying the collected data. However, time has been one of our biggest constraints this term, so this task remains a stretch goal.

On a non-technical note, we still have the entirety of the Spring term left, and the most important part of that term will be appearing at Oregon State's Engineering Expo to show our finished product.

4 Problems with Progress

The main problem with our progress has simply been time. We're all students with several other classes as well as jobs. In addition, our advisor, Professor Chet Udell, is a very busy man managing several other Capstone projects, so we can't necessarily meet with him every week. In addition we had some difficulties scheduling our TA meetings earlier in the term. We would meet in the OPEnS Lab and get so immersed in building our rig that we would simply forget to go to our meetings at 1 pm on Mondays. We have moved our meeting time now to 12:45 pm on Wednesdays and have been much more consistent about being able to attend them. Another factor that has impeded our progress has been our inexperience with electrical hardware components and using a 3D printer. A good portion of our time this term has been spent doing self-research on those topics, and asking for help in the OPEnS lab if that proves ineffective. We have learned a great amount of skills that could definitely help us later on, such as creating mesh models and wiring a microcontroller and a sensor to a breadboard.

5 Code

```
1 // Config has to be first has it hold all user specified options
2 #include "config.h"
3
4 // Preamble includes any relevant subroutine files based
5 // on options specified in the above config
6 #include "loom_preamble.h"
7
8 #include "arduino_secrets.h"
9 #include <Adafruit_SleepyDog.h>
10
11 const int PIEZO_PIN = A0; // Piezo output
12 const int ledPIN = 13; // LED connected to digital pin 13
13
14 //float sensorVals[4000];     // create sensor values storage array
15 //float sensorAvgs[100];
16 //float sensorNumDrops[100];
17 //float sensorFreq[100];
18 float duration = 5;
19 int i = 0;
20 int avg = 0;
21 int n = 0;
22 //int f = 0;
23 int t = 0;
24 float average;
25 float frequency;
26 String intensity;
27 boolean loopReset = true;
28 unsigned long referenceTime = 0;
29 int ledState = LOW; // variable used to store the last LED status, to toggle the
       light
30 boolean raining = true;
```

First we include the config.h and loom_preamble.h files to make sure our code is connected to the large loom library. This will give us access to hundreds of functions for use in our code. We then define our global variables that we will use throughout our functions. Variables of note are: PIEZO_PIN which is the pin connected to our sensor, duration which is the specified time in seconds that we want our timer to run for, and the average, frequency, and intensity variables that will hold our data.

```
void setup()
{
  // LOOM_begin calls any relevant (based on config) LOOM device setup functions
  Loom_begin();
  //WiFi.setPins(8,7,4,2);
  pinMode(ledPIN, OUTPUT); // declare the ledPin as the OUTPUT
  digitalWrite(ledPIN, HIGH); // Show device is awake
  Serial.begin(9600);
  //attachInterrupt(PIEZO_PIN, wakeUpNow, LOW);
}
```

Next we have our setup function which calls all relevant setup functions through the Loom_begin function. For example, our board is connected to WIFI here.

```
void loop()
{
  OSCBundle send_bndl;
  measure_sensors();
  package_data(&send_bndl);

  gather_data();

  append_to_bundle_key_value(&send_bndl, "Average", average);
  append_to_bundle_key_value(&send_bndl, "Frequency", frequency);
  append_to_bundle_key_value(&send_bndl, "Intensity", intensity);

  print_bundle(&send_bndl);
  log_bundle(&send_bndl, PUSHINGBOX);
  additional_loop_checks();
}
```

Our Main Loop sets up the Bundle we will use to send data to the Pushing Box functions. It then calls the gather_data function which contains all of the data collection code. That data is then appended to the bundle and the bundle is sent to the Pushing Box functions which constructs a URL using the data key-values and send it as a get request to our Google Sheet. A simple script in the Google Sheet interpolates the URL arguments and adds the data to the Spreadsheet.

```
void gather_data(){
  float sensorVals[4000];
  if(raining == false){
    digitalWrite(ledPIN, LOW); // Show device is asleep
    attachInterrupt(PIEZO_PIN, wakeUpNow, RISING);
    int sleepMS = Watchdog.sleep();
    detachInterrupt(A0);
  }
  else if(raining == true){
    if(loopReset){
      referenceTime = millis()/1000;
    }
    do{
      if(call_timer(0, duration) == true){
        // Read Piezo ADC value in, and convert it to a voltage
        int piezoADC = analogRead(PIEZO_PIN);
        float piezoV = piezoADC / 1023.0 * 5.0;
        if(piezoV > .1) {
          Serial.println(piezoV); // Print the voltage.
```

```
20            sensorVals[i] = piezoV;
21            i++;
22            delay(100);
23          }
24          loopReset = false;
25        }
26        else if(call_timer(0, duration) == false){
27          float temp = sensorVals[0];
28          for(int j = 0; j < i; j++){
29            temp = temp + sensorVals[j+1];
30          }
31          average = temp / (i + 1);
32          frequency = ((i+1)/duration);
33          switch(average){
34            case (0 < average && average <= 35): intensity = String("Light");
35            case (35 < average && average <= 50): intensity = String("Medium");
36            case (50 < average && average <= 100): intensity = String("Heavy");
37          }
38          //sensorAvgs[avg] = average;
39          //sensorNumDrops[n] = (i+1);
40          //sensorFreq[f] = ((i+1)/duration);
41          //f++;
42          //n++;
43          //avg++;
44          i = 0;
45          loopReset = true;
46        }
47      }while(loopReset == false);
48  //    if(avg == 10){
49  //      for(int a = 0; a < 10; a++){
50  //        Serial.println(sensorAvgs[a]);
51  //      }
52  //      Serial.println("----------------------");
53  //      for(int b = 0; b < 10; b++){
54  //        Serial.println(sensorNumDrops[b]);
55  //      }
56  //      Serial.println("----------------------");
57  //      for(int c = 0; c < 10; c++){
58  //        Serial.println(sensorFreq[c]);
59  //      }
60  //      avg = 0;
61  //    }
62    }
63  }
```

Our gather_data function contains our data collection code. We start by checking to see if it is raining. If it is not, then the microcontroller will go to sleep. It will awaken when the sensor triggers an interrupt (from a rain drop). If it is raining, then the rest of our code is run. Our sleep code is not fully operational and requires further testing.

We continue by setting the timer if loopReset = true. We then call the call_timer function to see if the timer is running or if it has ended. If it is still going, then we read the output from PIEZO_PIN and store that value into an array. The delay is to prevent excess readings from being recorded.

If the timer has stopped, then we add together all of the data values and divide that by the total number of values to get the average. We take the total number of values and divide it by the duration to get the frequency. To find the intensity, we compare the average to set ranges to determine how hard the rain was falling.

```
boolean call_timer(int start, int duration){
  unsigned long timer = millis()/1000;
  unsigned long startTime = referenceTime + start;
  unsigned long endTime = startTime + duration;
  if(timer >= startTime && timer <= endTime){
    return true;
  }
  else{
    return false;
  }
}
```

This is our call_timer function where we check to see if the timer has ended. Two integers are passed into the function: start and duration. The start value is usually 0 if we only have one timer set, but if we had additional timers then we would tell the function when the start of those timers would be. For example, if timer 1 has a start of 0 and a duration of 60, then timer 2 would have a start of 61 so there is no overlap. The duration value is how long in seconds that we want the timer to be active for. In our case, we will have the duration set to 120 because we want the data collection intervals to be two minutes long. If the timer value (think of it as current time elapsed) is in the range of the startTime and endTime values, then the timer is still active and it returns true. If it is out of that range, then the timer has ended and it returns false.

6 Images

Figure 1: The testing setup for our sensor and prototype sensing area.

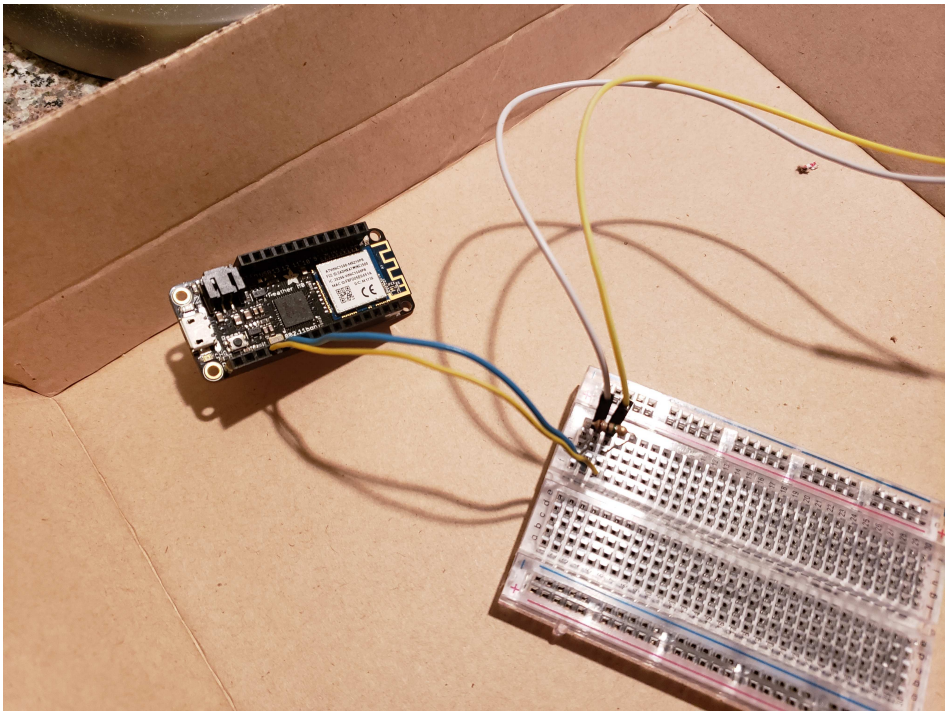

Figure 2: The wiring of our sensor to our microcontroller.

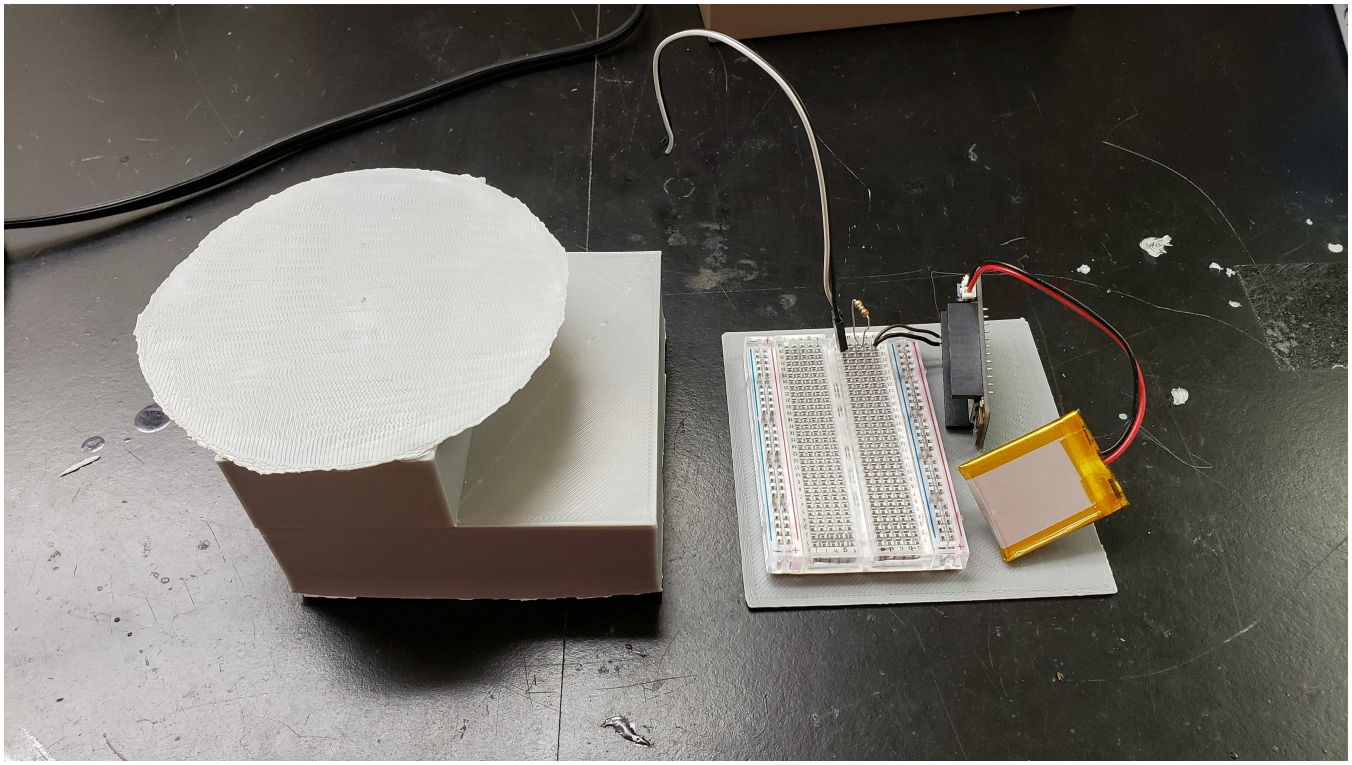

Figure 3: 3D printed enclosure and sensor disk (right) and our Adafruit Feather M0, battery, and breadboard (left).