Introduction

The Qualmetrics Weathertronics 6011-A tipping bucket gain gauge is used to measure rainfall volume and rate. The tipping bucket gauge will be modified to use a Hall effect sensor instead of a magnetic reed switch. Magnetic reed switches eventually fail due to corrosion or mechanical failure. A hall effect sensor has no moving parts should provide a reliable non-mechanical momentary switching option.

Rain water enters the gauge through an opening in the top and is fed through a funnel into one of two buckets. When one bucket contains 0.01" of water it will tip to the side and water will begin to fill the second bucket. The tipping motion actuates a momentary reed switch closing a circuit. The closing or opening of the circuit can be read and recorded as a digital signal. Each signal represents 0.01" of rain. The signals are summed hourly and daily to give the total amount of rainfall during those times.

Figure 1: Tipping bucket rain gauge.

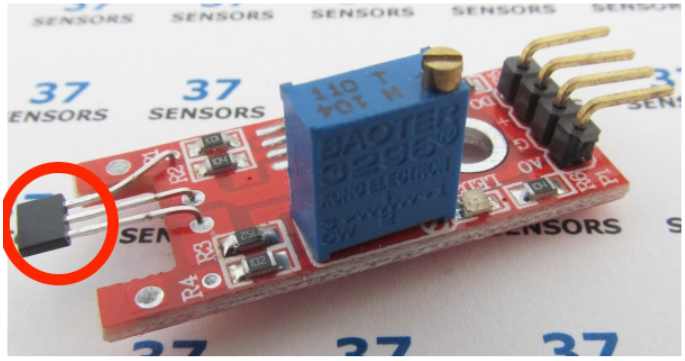

Figure 2: Hall effect sensor module with Honeywell SS49E Hall sensor circled in red.

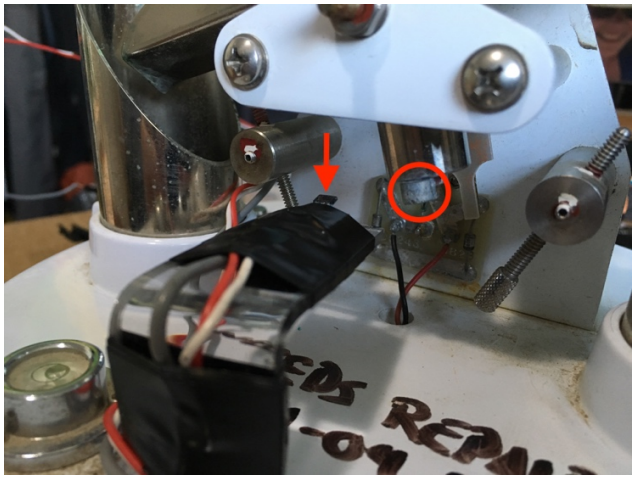

Figure 3: Modified rain gauge. Red arrow points to the Hall sensor. Red circle highlights added magnet.

Table of contents (Place holders)

Specifications [1,2,4]

Qualmetrics Weathertronics 6011-A Tipping Bucket Rain Gauge:
• Mechanical Sensor type: Tipping bucket
• Switch: Form A reed – modified to use Honeywell SS49E Hall sensor
• Sensitivity: 0.01” per count
• Accuracy: ± 0.5% at 0.5"/hr
• Collector orifice: 8.214" diameter (208 mm)
• Size: 8.25" dia x 17.5" H (210 mm x 445
• Hall Sensor Temperature range: -40°C to 85 °C
• Current draw ~ 90mA
• Hall Sensor Response time: 3uS
• Hall sensor magnetic range: min +/-650, typical +/- 1000 Gauss
• Input voltage: 7 – 12 VDC

Absolute maximum ratings:

Arduino [3]
• Maximum input voltage from external 2.1mm center-positive plug: 20VDC
• Use only center positive 2.1mm center-positive plug for external power
• Min/max temperature range: -40°C to 85 °C

Typical performance characteristics

The rain gauge will create a log entry every time the bucket tips, every hour, and every 24 hours at midnight. Each log entry is approximately 44 to 50 bytes. When necessary data is flushed to the SD card every 5 seconds by default. The 5 second interval is adjustable in the firmware by setting the desired value of the flushDelay variable. Adjusting the variable may save power by not writing to the SD card as often.

Theory of operation

Rain water enters the gauge through an opening in the top and is fed through a funnel into one of two buckets. When one bucket contains 0.01" of water it will tip to the side and water will begin to fill the second bucket. The tipping motion actuates Hall sensor closing or opening the circuit. The closed or open circuit is read on digital pin 3 of the Arduino and assigned a value of 0 or 1. Each time this value changes clock time and a bucket tip is recorded. Each recorded tip represents 0.01” of rain.

A hall sensor works by detecting a magnetic field with a minimum of +/-650. When in the presence of a strong enough magnetic field the sensor develops a voltage. The voltage is proportional to the strength of the magnetic field. Rather than reading a voltage, this rain gauge Hall sensor is configured as a switch and is pulled to 5v when a field of sufficient strength is detected and to 0V otherwise.

A switch debouncing library is used in the firmware to prevent false switch state changes with the Hall sensor module. A switch bounce is illustrated in figure 4. The debouncing library uses a delay or smoothing algorithm to avoid reading the non-stable interval when the logic state changes from high to low or vice versa.

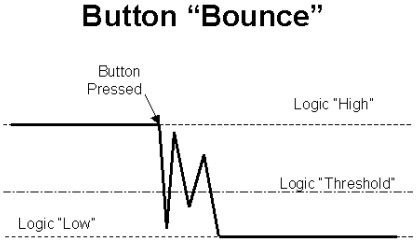

Figure 4: An illustration mechanical switch “bounce”. A similar bounce was seen when using the Keyes KY 024 Hall Sensor module. Illustration credit (6).

A DS3231 AT24C32 I2C real time clock (RTC) is used to produce a time stamp for each log entry. Additionally, the RTC will maintain the current date and time when the Arduino is not powered by using a CR2032 3V battery. (7)

The rain gauge will create a log entry every time the bucket tips, every hour, and every 24 hours at midnight. Each log entry is approximately 44 to 50 bytes. When necessary data is

flushed to the SD card every 5 seconds by default. The 5 second interval is adjustable in the firmware by setting the desired value of the flushDelay variable. Adjusting the variable may save power by not writing to the SD card as often.

Schematic

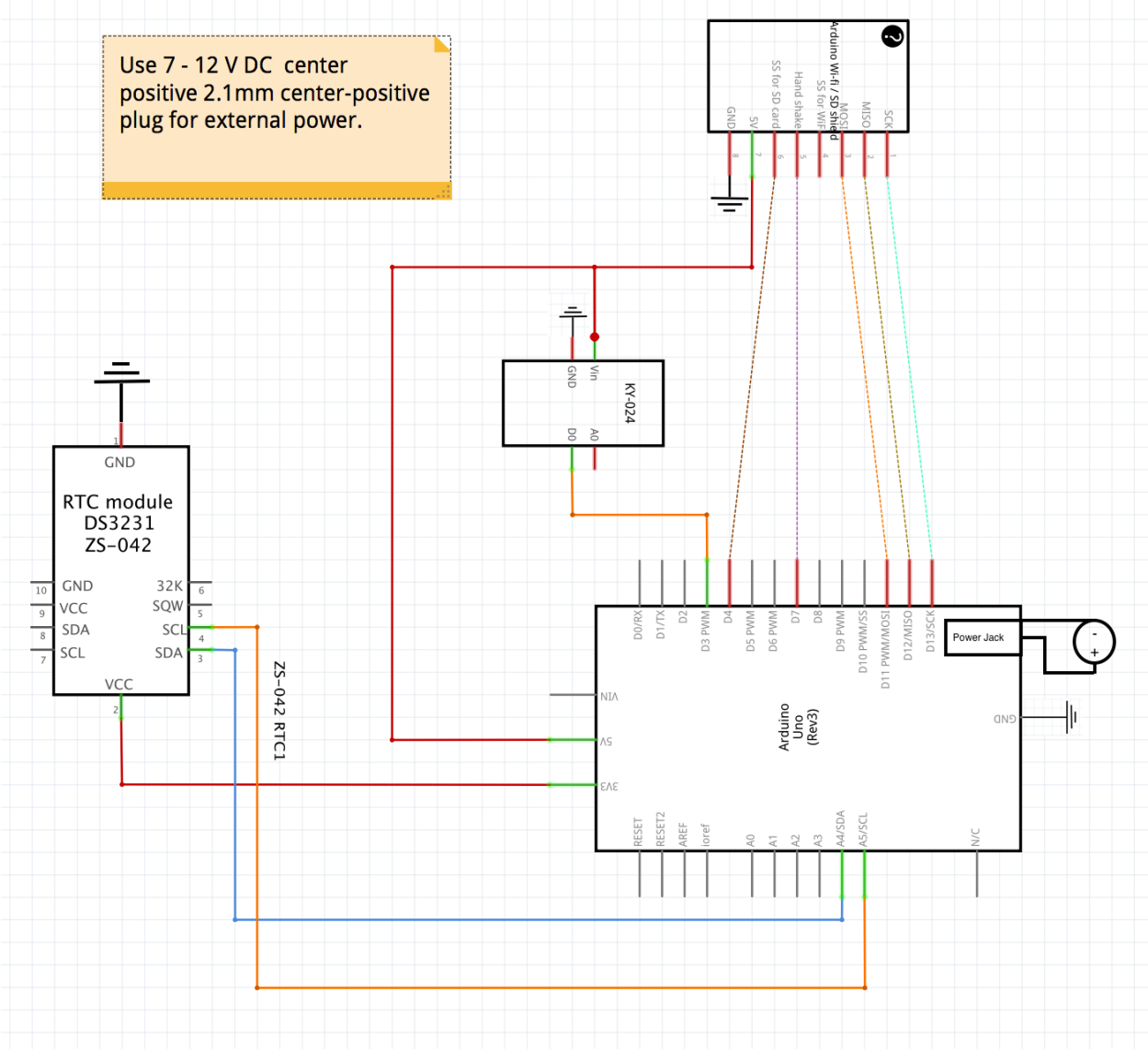

Figure 5: Schematic showing Arduino UNO R3, KY-024 Hall sensor module, Arduino Wi-fi / SD card shield, and DS3231 RTC.

Wiring diagram:

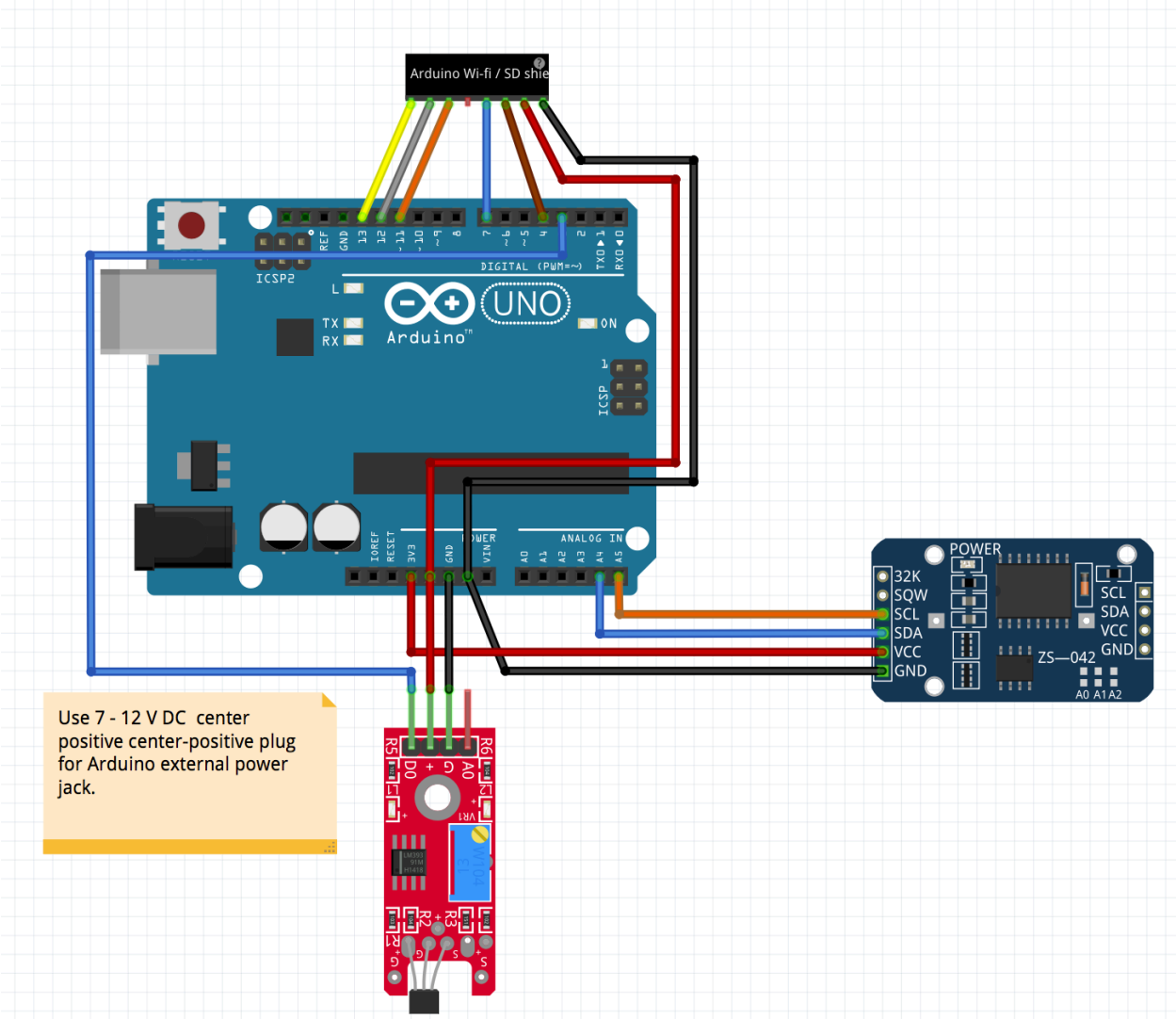

Figure 6: Wiring diagram. During assembly the hall sensor is removed from the module and soldered to longer wire leads. This allows the Hall sensor to be installed in the rain gauge and module can be placed in waterproof container with the Arduino.

Firmware code:

```
// Tipping bucket with Hall Sensor
/*
  This script will read a switch in a tipping bucket rain gauge
  and log each bucket tip. It will record real time bucket tips
  as well the total bucket tips per hour and total bucket tips per
  day. This script expects the Hall Sensor (or other switch) to be
  closed while the bucket is tipped to one side and open with
  tipped to the other side.

  See the manual for wiring and Hall Sensor placement and
  calibration instructions.

  In the field you can use serial plotter to view the sensor or switch
  state. The plot should change from 0 to 1 as the bucket is moved from
  side to side. Attempt to adjust (move) the Hall Sensor so that the
  state changes just as the bucket assembly moves past the center point.
```

```
  It is NOT critical that it is exactly in the center so long as you see a
  reliable state (0 or 1) change when the buckets are resetting at full
  swing on either side.
*/

#include <SPI.h>
#include <SD.h>
#include <Bounce2.h> // Library is used to debounce switches to prevent false switch state interpretation.
#include <DS3231_Simple.h> // RTC library

//#include <avr/power.h>  // Can be used with CLKPR (clock divider) to save power.

#define HallSensorPin 3
#define LED_pin 13 // this is the onboard LED (13) and is not required, it just can be used for setup/calibratiion.
#define CS 4 // for adafruit wifi shield
//#define CS 10 // for other shields

byte HallValue = 0;      // variable to represent Hall Sensor state; low = 0, high = 1
byte LastValue = 0;      // Last HallValue reading, compare to current reading to see if switch state changed.
unsigned long tipCount = 0;           // the total number of bucket tips since startup
unsigned long hourTipCount = 0;   // the total number of bucket tips for the hour
unsigned long dayTipCount = 0;    // the total number of bucket tips for the day
unsigned long flushDelay = 5000;  // flush log to file every 5 seconds - might save a litte power
unsigned long flushTimer = 0;         // Temporary variable for saving the last time the data was written to the SD card

File LogFile;                 // declare the SD file name
DS3231_Simple Clock; // declare the RTC

Bounce debouncer = Bounce();  // Instantiate a Bounce object

void setup() {
  Serial.begin(9600);
 //pinMode (LED_pin, OUTPUT);  // this will light the onboard LED (13) when the Hall state switch state is HIGH. It is not
                               // required.
  pinMode (HallSensorPin, INPUT);

  // The lines below should theoretically save some power by reducing the system clock speed by reducing it from 16MHz to
  // 1 MHz. However, I did not have time to test it.
  //CLKPR = 0x80;    // Tell the AtMega we want to change the system clock.
  //CLKPR = 0x04;    // divide clock peed to get 1 MHz.

  Clock.begin();                // Start the clock
  Clock.disableAlarms();     // Turn off any existing alarms before creating new ones.
  //DateTime CurrentTime;  // Create a variable to hold the clock time

  DateTime alarmTime = Clock.read();      // Get a timestamp then add alarm times
  alarmTime.Second   = 59;  // Sets the seconds to trigger alarm 1
  alarmTime.Minute   = 59;  // Sets the minutes to trigger alarm 1 (end of the hour)
  alarmTime.Hour   = 23;    // Sets the hour to trigger alarm 2 (just before midnight)

  // Only two RTC alarms are available on the RTC. Use one each hour and one each day (midnight).
  Clock.setAlarm(alarmTime, DS3231_Simple::ALARM_MATCH_SECOND_MINUTE);   // This sets the hourly alarm 1
(timer).
  Clock.setAlarm(alarmTime, DS3231_Simple::ALARM_DAILY); // This sets the daily (24 hour) alarm 2 (timer).

  DateTime CurrentTime = Clock.read();      // Get an initial timestamp
  // Set the time and date is the RTC battery has been replaced or removed!
  // Comment out the lines below unless setting the clock
  /*
    CurrentTime.Day    = 28;
    CurrentTime.Month  = 03;
    CurrentTime.Year   = 19; // two digits only
    CurrentTime.Hour   = 15;
    CurrentTime.Minute = 35;
    CurrentTime.Second = 00;
    Clock.write(CurrentTime);   // Update the clock with the time above.
    Serial.print("The time has been set to: ");
    Clock.printTo(Serial);
    Serial.println();
  */
```

```
  // Remeber to comment out the lines above, re-compile and upload the
  // sketch again once the time is set!

  debouncer.attach(HallSensorPin);   // After setting up the button, setup the Bounce instance :
  debouncer.interval(25);            // debounce interval in ms. The amount of time to wait for the switch state to stabilize.

  // start the SD card connection
  while (!SD.begin(CS)) {
    Serial.println("Failing to connect to SD card");
    delay(2000);
  }
  Serial.println("SD card connected");

  LogFile = SD.open("DATAFILE.TXT", O_RDWR | O_APPEND); // Try to open existing Logfile and append data to it.
  while (!LogFile) {
    Serial.println("Failing to open data file");
    LogFile = SD.open("DATAFILE.TXT", O_CREAT | O_WRITE | O_TRUNC); // Create a new file if file does not exist.

    // Write data description to file - type is "start", "hourTotal","dayTotal", and "tip"
    LogFile.println("TimeStamp,Year,Month,Day,Hour,Minute,Second,tipcount,type");
    delay(2000);
  }
  Serial.println("Data file opened");

  // ###################################
  // Log the startup time.
  CurrentTime = Clock.read(); // read the time stamp from the RTC
  logtime();                  // call function to print time to Logfile
  LogFile.print(tipCount);    // print the total number of bucket tips so far to the file (zero so far)
  LogFile.print(",");
  LogFile.print("start");     // log this as a startup type event.
  LogFile.println();
  LogFile.flush();
  //  End logging the startup time.
  // ###################################

  // ###################################
  // Get the initial Hall Sensor state
  // This is done to prevent the posibility of 1 extra
  // tip bucket count on startup.
  debouncer.update();         // Read the digital pin attached to the Hall sensor.
  int HSvalue = debouncer.read(); // Get the updated value, i.e. is the state HIGH or LOW
  if ( HSvalue == LOW ) {
    HallValue = 1;            // equate LOW to a numerical value of 1
  }
  else {
    HallValue = 0;            // equate HIGH to a numerical value of 0
  }
  LastValue = HallValue;
  // End the initial Hall Sensor state
  // ###################################
}

void loop() {
  debouncer.update();         // Read the digital pin attached to the Hall sensor.
  int HSvalue = debouncer.read(); // Get the Hall Sensor Value (HIGH or LOW) :

  // Set a value and turn on or off the LED as determined by the state. Leave LED lines
  // commented out for lower power consumption.
  if ( HSvalue == LOW ) {
    //digitalWrite(LED_pin, HIGH );
    HallValue = 1;
    // Serial.println("Switch state HIGH");
  }
  else {
    //digitalWrite(LED_pin, LOW );
    // Serial.println("Switch state LOW");
    HallValue = 0;
  }
  Serial.println(HallValue);
```

```
  if (HallValue != LastValue ) {     // if the Hall sensor value has changed, the buckets have tipped.
    tipCount++;                      // add one to the total tip count
    hourTipCount++;                  // add one to the hourly tip count
    dayTipCount++;                   // add one to the daily tip count
    logtime(); // call function to print time to Logfile
    LogFile.print(tipCount);  //print the total number of bucket tips so far to the file.
    LogFile.print(",");
    LogFile.println("tip");
  }
  LastValue = HallValue;
  uint8_t AlarmsFired = Clock.checkAlarms();

  // hourly timer, record bucket tip count for last hour
  // Check alarm 1 state by using a "bitwise and"
  if (AlarmsFired & 1) {
    logtime();
    LogFile.print(hourTipCount);  //print the total number of bucket tips from the last hour the file
    LogFile.print(",");
    LogFile.println("hourTotal");
    LogFile.flush();
    hourTipCount = 0; // reset the hour count
    Clock.printTo(Serial);
    Serial.println("Alarm 1 (hourly"); //once per hour
  }

  // 24 hour timer, record bucket tip count for last 24 hours
  // Check alarm 2 state
  if (AlarmsFired & 2) {
    logtime();          // call function to print time to Logfile
    LogFile.print(dayTipCount);  //print the total number of bucket tips from the last 24 hours to the file
    LogFile.print(",");
    LogFile.println("dayTotal");
    LogFile.flush();
    dayTipCount = 0; // reset the day count
    Clock.printTo(Serial);
    Serial.println("Alarm 2 (daily"); //once per day
  }

  if (millis() - flushTimer > flushDelay ) { // Save some power by only writing to the file in flushDelay ms.
    LogFile.flush();        // save the file
    flushTimer = millis();  // reset the flush timer to current millis
  }
}

// Function to log a time stamp to the data file.
void logtime(void) {
  DateTime CurrentTime;              // Create a variable to hold the clock time
  CurrentTime = Clock.read();        // Read the realtime clock.
  Clock.printTo(LogFile);            LogFile.print(",");
  LogFile.print("20");  //prepend 20 to the begining of the year so that we have a 4 digit year.
  LogFile.print(CurrentTime.Year);    LogFile.print(",");
  LogFile.print(CurrentTime.Month);   LogFile.print(",");
  LogFile.print(CurrentTime.Day);     LogFile.print(",");
  LogFile.print(CurrentTime.Hour);    LogFile.print(",");
  LogFile.print(CurrentTime.Minute);  LogFile.print(",");
  LogFile.print(CurrentTime.Second);  LogFile.print(",");

}
```

Instructions

The hall sensor must be oriented flat side up (figure 7) and close enough to the magnet to "turn on" when the bucket is tipped to one side and "turn off" when the bucket is tipped to the opposite side. This requires the sensor to be positioned both vertically and horizontally. Vertical adjustment is performed by bending a mounting bracket to the correct height (I used a piece of Lexan (plexiglass) bent with a heat gun as shown in figures 7 and 8. A magnet, circled in red (figure 8), must be affixed to the tipping bucket's counter weight. The added weight of the magnet may require calibration of the tipping buckets.

Left to right adjustment is shown in figure 8. The Arduino IDE serial plotter can be used to see when the sensor is activated. If using the Keyes KY Hall Sensor module, you should also see an LED light when the sensor is activated.

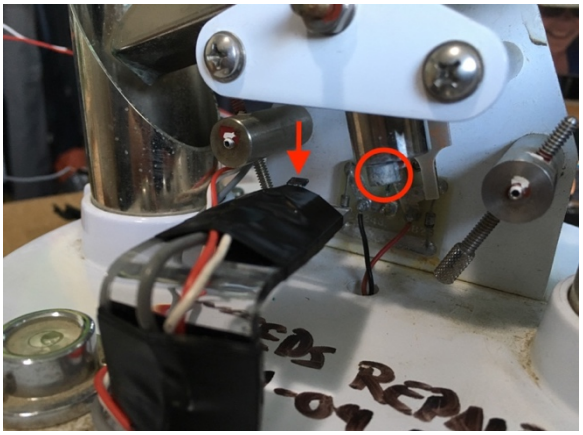

Figure 7: Modified rain gauge. Red arrow points to the Hall sensor.

Figure 8: Left to right adjustment of the hall sensor.

When setting up the tipping buck gauge refer to the Weathertronics 6011-A tipping bucket rain gauge user's manual for placement and additional calibration instructions. [5]

Each line of data recorded to the SD card will have the following format: Clock time stamp, Year, Month, Day, Hour, Minute, Second, tip count, and type. Tip count is the total number of bucket tips recorded or the total for the hour or day. The 'type' field indicates the type of data for the line; startup (when the Arduino is powered on with an SD card inserted), a real time bucket tip, or an hourly or daily total.

Example data and code

Sample data:

Clock Time Stamp	Year	Month	Day	Hour	Minute	Second	tipcount	type
2019-03-28T08:13:17	2019	3	28	8	13	17	0	start
2019-03-28T22:35:53	2019	3	28	22	35	53	1	tip
2019-03-28T22:43:12	2019	3	28	22	43	12	2	tip
2019-03-28T22:59:59	2019	3	28	22	59	59	2	hourTotal
2019-03-28T23:15:53	2019	3	28	23	15	53	3	tip
2019-03-28T23:59:00	2019	3	28	23	59	59	1	hourTotal
2019-03-28T23:59:00	2019	3	28	23	59	0	3	dayTotal

Python code:

```
# This Python 3 script is designed to analyze
#  data from a tipping bucket rain gauge.
# It will plot hourly rainfall for a given day,
# daily rainfall for a month, and the total cumulative
# rainfall measured in the data from the SD card.
# Expected data format:
# TimeStamp,Year,Month,Day,Hour,Minute,Second,tipcount,type
# The type field indicates the type of event being logged:
#  "start", "hourTotal","dayTotal", or "tip"

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

fileName = '~/RainGauge/data/DATAFILE.TXT'

# Load the data file into a pandas dataframe
# pandas is used so that the text headers and text type fields are maintained.
df = pd.read_csv(fileName)

# Create an array containg the data from the pandas data frame
rainArray = df.values

# The next three lines create arrays of of all the bucket tips,
# the hourly totals, and the daily totals.
allTips = rainArray[rainArray[:,8] == 'tip']
hourTotals = rainArray[rainArray[:,8] == 'hourTotal']
dayTotals = rainArray[rainArray[:,8] == 'dayTotal']

# Calculate the total amount of rainfall recorded in the
# datafile. The depth in inches is equal to the the
# total number of tips that occured times 0.01 inches.
totalRainfall = np.size(allTips,0) * 0.01

# prompt to enter a month and day to plot. If the month or day does
# not exist in the data an error will occur.
month = input('What month do you want to plot for? [1-12] ')
day = input('What day of the month do you want to plot hourly rainfall for? [1-31] ')

plotDay = hourTotals[hourTotals[:,3] == (int(day) + 1 )] # Create array containing data for day
plotMonth = dayTotals[dayTotals[:,2] == int(month)]      # Create array containing data for month

# Plot the rainfall for the month
# Create list of hours for plot
hour = plotDay[:,4]
plt.figure(figsize=(8, 4))
plt.bar(hour, (plotDay[:,7]*0.01))
plt.xlabel('Hour')
plt.ylabel('Rain (inches)')
plt.title('Hourly Rainfall for ' +str(month) + '/' + str(day) + '/' + str(round(plotDay[0,1])))
# Set the ticks on the x-axis
plt.xticks(np.arange(np.min(hour),(np.max(hour)+1),1))
plt.tight_layout()
plt.show()
tempVar1 = dayTotals[dayTotals[:,3] == int(day)]  // Select day totals
tempVar1 = tempVar1[0,7]*0.01                     // Multiply number of tips by 0.01” for rain depth
print ('Total rain for ' + str(month) + '/' + str(day) + '/' + str(round(plotDay[0,1])) + ' was '+ str(tempVar1)+' inches.')

# Plot the rainfall for the month
# List of days for plot
days = (plotMonth[:,3] - 1)
plt.figure(figsize=(8, 4))
plt.bar(days, (dayTotals[:,7]*0.01))
plt.xlabel('Day of the Month')
plt.ylabel('Rain (inches)')
plt.title('Daily Rainfall for the month '+ str(month) + '/' + str(round(plotDay[0,1])))
# Set the ticks on the x-axis
plt.xticks(np.arange(np.min(days),(np.max(days)+1),1))
```

```
plt.tight_layout()
plt.show()
tempVar1 = dayTotals[dayTotals[:,2] == int(month)]
tempVar2 =np.sum(tempVar1, axis=0)
tempVar1 = tempVar1[0,7]*0.01
print ('Total rain for the month of ' + str(month) + '/' + str(round(plotDay[0,1])) + ' was '+ str(tempVar2[7]*0.01)+'
inches.')

# Plot of cumulative rain recorded on SD card.
# This plot includes all tip counts in data file.
plt.figure(figsize=(20, 6))
plt.plot(allTips[:,0],(allTips[:,7] * 0.01))
plt.xlabel('date')
plt.ylabel('Rain (inches)')
plt.title('All rain recorded in datafile')
plt.xticks( rotation=90 )
plt.xticks(fontsize=7)
plt.show()
```

References

1. Keyes KY 024 Hall Sensor module datasheet: https://github.com/R2D2-2017/R2D2-2017/wiki/Keyes-KY-024-Hall-Sensor (This is the closest thing I found to an actual datasheet for the Hall sensor module.)
2. Honeywell SS49E Hall sensor datasheet: https://sensing.honeywell.com/ss39et_ss49e_ss59et_50000353_issue-2_final_20dec12.pdf, published by Honeywell.
3. Arduino Uno Datasheet: https://www.farnell.com/datasheets/1682209.pdf, published by Farnell distributors.
4. Weathertronics 6011-A tipping bucket rain gauge datasheet: https://novalynx.com/brochures/260-6011.pdf
5. Weathertronics 6011-A tipping bucket rain gauge user's manual: http://www.allweatherinc.com/wp-content/uploads/6011-0011.pdf, published by Allweather inc.
6. https://students.cs.byu.edu/~clement/cs224/references/HowTos/HowTo_Switches.php
7. Maxim Integrated DS3231 datasheet: https://datasheets.maximintegrated.com/en/ds/DS3231.pdf

Addendum

The Hall sensor needs to be adequately water proofed. Silicon conforming compound is often used to waterproof circuit boards and is a good option. When I attached the wires to the hall sensor, I used a copper veroboard and did not adequately waterproof it. As a result the Hall sensor stopped working after about 6 days use with precipitation. This should be easily avoided with proper waterproofing.

It should be possible to use a Hall sensor and a 100k resistor instead of using the whole module. It should also be possible directly replace the reed switch in its original position with a more sensitive Hall sensor such as a Uxcell a14072500ux0221. The Arduino Uno R3 and SD card shield cab be replaced with less expensive options such as an Arduino Pro Mini and a small dedicated SD card writer/reader. I found the Arduino Pro Mini, SD card writer/reader, and a DS3231 real time clock. For a total of $11 on eBay. The Arduino Pro Mini also uses less power than an UNO R3. However, the Arduino Pro Mini requires an FTDI programmer module which costs about $5. Only one FTDI module is required as it is not part of the sensor and is only used to upload firmware.

BOM:

Arduino Pro Mini:
https://www.ebay.com/itm/Arduino-Pro-Mini-Board-Free-with-Headers-ATMEGA328P-16MHz-5V-ATMEGA328/382557228840

SD card module:
https://www.ebay.com/itm/2PCS-Read-And-Write-For-Arduino-ARM-MCU-SD-Card-Module-Slot-Socket-Reader-N150/382580269850

RTC:
https://www.ebay.com/itm/DS3231-AT24C32-IIC-Precision-Real-Time-Clock-RTC-Memory-Module-for-Arduino/382564275397

FTDI programmer:
https://www.ebay.com/itm/FT232RL-FTDI-USB-3-3V-5-5V-to-TTL-Serial-Adapter-Module-for-Arduino-Mini-Ports/382704226692