# Design and implementation of a Digital Synchronous Detector for Microwave Radiometer

A technical report for 2022 CISESS summer internship

Begin Date: 06/13/2022 End Date: 08/26/2022

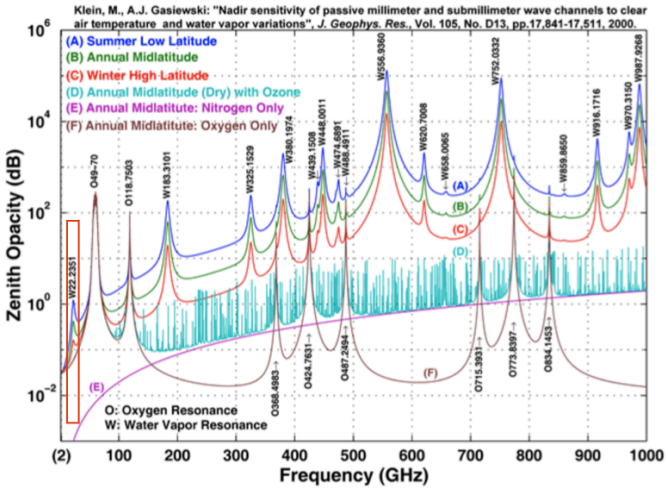
**Mentor**: Dr. Hu Yang
**Phone**: 301 405 4240
**Email**: huyang@umd.edu

**Student**: Chao-Wei Tu
**Phone**: 240 659 8664
**Email**: ctu1@terpmail.umd.edu

cisess
Cooperative Institute for Satellite Earth System Studies

## 1. Introduction: A description of the scope and background of the research project

Microwave signals in the atmosphere contain important parameters and are widely used to predict weather conditions. Our main goal is to build a microwave radiometer to collect water vapor information in the atmosphere. Since 22GHz is the frequency most reactive to water vapor changes, we design our instrument to collect this frequency.




## 2. Instrument Description:

### 2.1 Block Diagram of the Microwave Radiometer

Figure 2 demonstrates the basic working principle of our microwave detector. Water vapor signal is first detected by the Feed Horn Antenna, goes through switch control and a Low Noise Block module, and is finally detected by a highly sensitive power detector before being received by the microcontroller ATmega328P. The Feed Horn Antenna used on the radiometer is a K band (18~26GHz) antenna. This range is perfect for detecting water vapor signals because it has a central frequency of 22GHz, which is the absorption frequency of water vapor. The switch control, which turns on/off our defection, is the key element in making digital synchronous demodulation possible. It is controlled by a pulse width modulation signal generated from ATmega328P. The Low Boise Block module

down-converts the 22GHz signal to a lower frequency because ATmega328P cannot operate at such a high frequency.

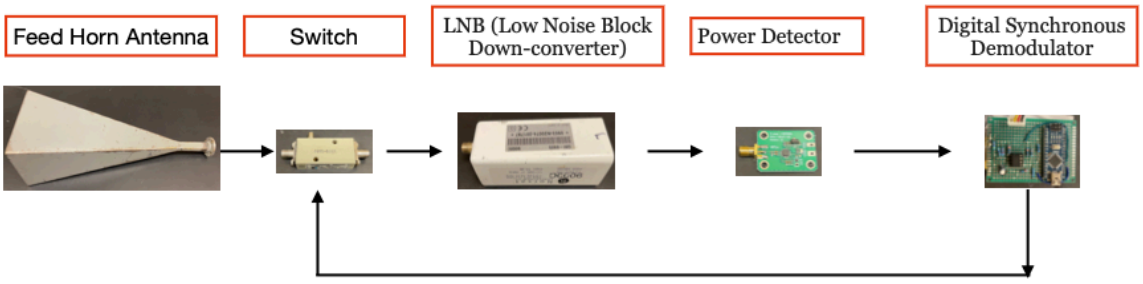



*Figure 2. Diagram of the Digital Synchronous Detector for Microwave Radiometer*

## 3. The Back-end Processing System

### 3.1 Principle of Digital Synchronous Detector

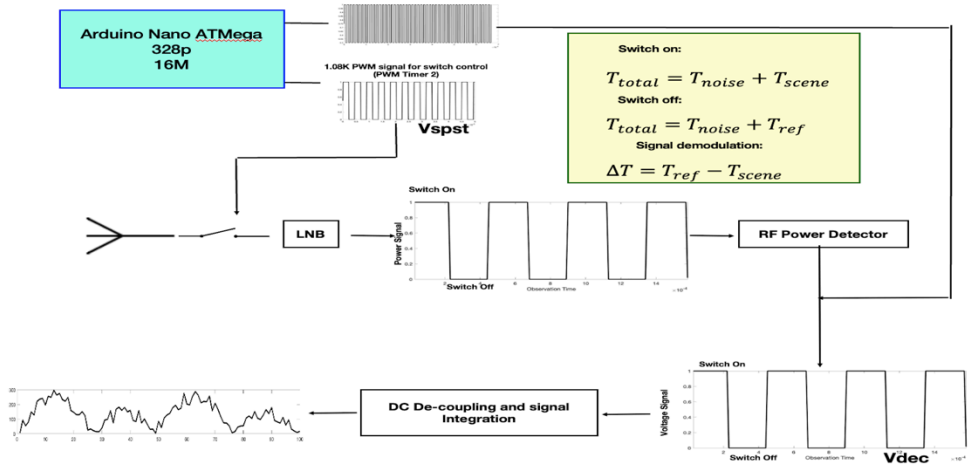



*Figure 3. Diagram of basic working principle of the Digital Synchronous Detector for Microwave Radiometer*

Figure 3 summarized the basic working principle of the Digital Synchronous Detector. The signal is collected by the antenna and turned on and off by the switch. When the switch is on, the detected signal contains background noise and scene signal. When the switch is off, the detected signal contains background noise and a reference signal. We take the difference between the detected signal when the switch is on and when the switch is off to remove the noise and restore the water vapor information. To perform the digital synchronous demodulation on ATmega328P, it is necessary to determine which part of the signal contains the scene and which part contains the reference, and thus the microcontroller needs to know when the switch is on and when it is off.

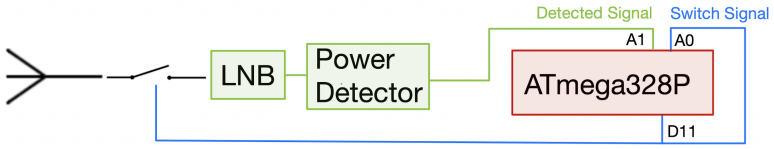



*Figure 4. Detailed diagram explaining how ATmega328P relates to the rest of the module*

Digital pin D11 of the ATmega328P generates a PWM signal that turns on the switch when D11 is high and off the switch when D11 is low. As a result, the detected signal is a square wave that is synched with the PWM generated from D11. Analog pin A0 and A1 are used to read the PWM signal and the detected signal respectively. In ATmega328P, there is an analog-to-digital conversion (ADC) module that makes AD conversion one at a time across all analog pins. To read two pins, the ADC module does conversion alternately between A0 and A1. Figure 5 demonstrates how the ADC module alternates between A0 and A1 and how they correspond to the PWM that turns on and off the signal.

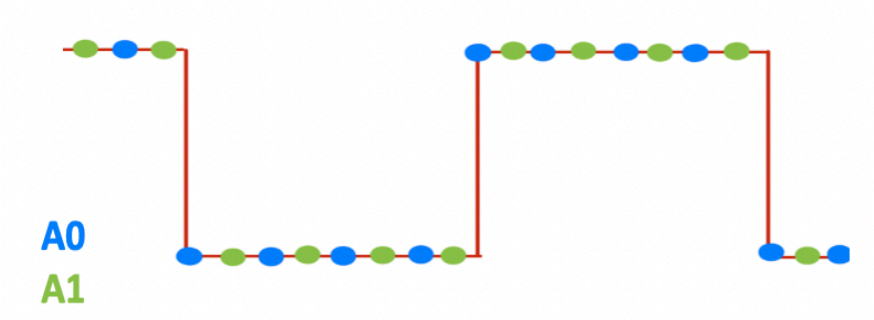



*Figure 5. Diagram of analog read by A0 and A1 pin on the ATmega328P. ADC sampling rate on each pin is 8 time the rate of switch control, so each analog pin record 8 sampling data every cycle of switch control*

Both the frequency of our PWM pulse that controls the switch and the frequency of AD conversion are factors that determine the resolution and cleanliness of the demodulation. To output our demodulated data on the water vapor signal at 100Hz, we found that setting the PWM pulse at 2083Hz and the ADC rate on each pin at 16.67kHz is optimal. The next section will discuss the firmware implementation of the digital synchronous demodulator.

### 3.2 Implementation of Digital Synchronous Detector

The firmware for the digital synchronous demodulator on the ATmega328P is implemented and debugged using the Arduino IDE.

#### 3.2.1 Variable Declaration

We declare several global variables and important numbers that will be used across the code. Note that the ADC rate is set at 33.33kHz because the ADC module on the

ATmega328P can only do one conversion at a time. To have a conversion rate of 16.67kHz on A0 and A1, the ADC rate needs to be 2 times the rate of 16.67kHz.

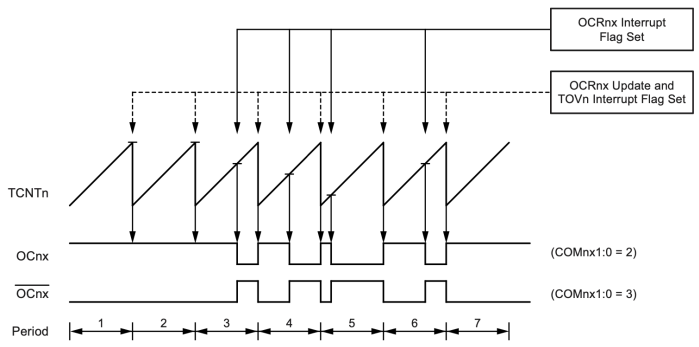
```
#include <Wire.h>                       // I2C
#define F_CPU 16000000UL                // Setting Arduino system clock

// ================================================================================
const uint32_t ADC_INTERVAL_MICROSECONDS = 30; // 33.33 kHz
const uint32_t SWITCH_CONTROL_INTERVAL_MICROSECONDS = 480; // 2.083kHz
const uint32_t TIMER2_PRESCALER = 32; // 1, 8, 32, 64, 128, 256, 1024
const uint32_t CLOCKS_PER_MICROSECOND = F_CPU / 1000000ul; // 16
const float PRESCALED_CLOCKS_PER_MICROSECOND = CLOCKS_PER_MICROSECOND /
TIMER2_PRESCALER / 2;
const uint32_t TIMER1_TOP = ADC_INTERVAL_MICROSECONDS *
CLOCKS_PER_MICROSECOND - 1;
const uint32_t TIMER2_TOP = SWITCH_CONTROL_INTERVAL_MICROSECONDS / 4 - 1;

const uint16_t MAX_RESULTS = 160;
```

```
// array to store ADC value
volatile uint16_t Vdec [MAX_RESULTS]; // A1, detected signal
volatile uint16_t Vspst[MAX_RESULTS]; // A0, pwm signal for switch control
volatile uint16_t spstCounter;
volatile uint16_t decCounter;
```

### 3.2.2 Timers on the ATmega328P

To control the PWM pulse frequency and the ADC frequency on the ATmega328p, we use internal timers given in the chip. The ATmega328P has a system clock speed of 16MGz and has 3 timers given: Timer0, Timer1, and Timer2. Timer0 is an 8-bit timer, which means it counts from 0 to 255 and back to 0. Timer0 is used by utility functions such as delay(), millis(), micros(), and more, so we won't be using Timer0 in this project. Timer1 is a 16-bit timer, which means it counts from 0 to 65535 and back to 0. Timer1 is used to trigger one AD conversion at each cycle. Timer2 is also an 8-bit timer. Timer2 will be used to generate PWM which controls the PWM.

#### 3.2.2.1 Timer2 Settings for Switch Control PWM Control



$$f_{OCnx} = \frac{f_{clk\_I/O}}{2 \times N \times (1 + OCRnx)}$$

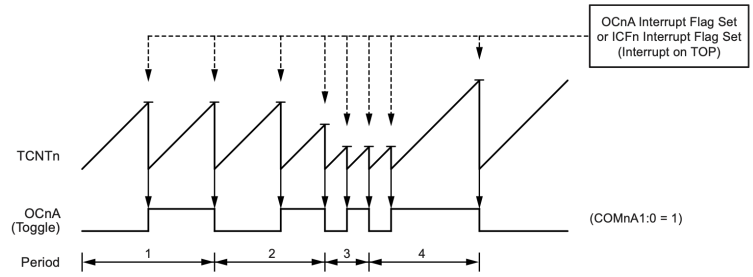
*Figure 6. Fast PWM mode on Timer2 and the equation to calculate the frequency of the PWM based on pre-scaler and OCR2A*

```
void setupTimer2(){
  // set Timer2 fast PWM for switch control
  // OC2A at pin D11

  TCCR2A = 0;
  TCCR2B = 0;

  TCCR2A |= _BV(COM2A0);                  // Toggle OC2A on compare match.

  // Fast PWM, TOP = OCRA
  TCCR2A |= _BV(WGM21) | _BV(WGM20);     // Set to fast pwm
  TCCR2B |=  BV(WGM22);                   // Set OCR2A as TOP
```

```
 TCCR2B |= _BV(CS21) | _BV(CS20);       // set prescalar to 32
 OCR2A = TIMER2_TOP;                    // set TOP
}
```

### 3.2.2.2 Timer1 Settings for ADC sampling rate Control

**Figure 15-6. CTC Mode, Timing Diagram**




$$f_{OCnA} = \frac{f_{clk\_I/O}}{2 \times N \times (1 + \quad ICR1 \quad)}$$

*Figure 7. CTC mode on Time12 and the equation to calculate the frequency of the PWM based on pre-scaler and ICR1*

```
void setupTimer1(){
  // set Timer1 to trigger ADC when overflow

  TCCR1A = 0;
  TCCR1B = 0;
  TIMSK1 = _BV(TOIE1);          // Overflow interrupt enabled
  ICR1 = TIMER1_TOP ;           // Set TOP

  // Set Fast PWM, TOP in ICR1, TOV1 at TOP
  TCCR1A |= _BV(WGM11);
  TCCR1B |= _BV(WGM13) | _BV(WGM12);

  TIFR1 |= _BV(TOV1);           // Clear any pending Timer1 Overflow

  TCCR1B |= _BV(CS10);          // set prescaler to 1
}
```

### 3.2.3 ADC on the ATmega328P

ATmega328P has a 10-bit ADC, which is capable to convert analog data to a value between 0 to 1023. Since only one channel can be converted at a time, the ADC module must do alternating conversions between A0 and A1. In addition, we set the internal 5V as a voltage reference for the conversion.

#### 3.2.3.1 ADC Mode Settings (Trigger Conversion on Timer1 Overflow)

```
void setupADC(){
  // Set up the ADC to start a conversion when Timer1 overflows

  // initialized analog channel to pin A0
  // initialized ADLAR=0 for right adjusted (in ADCH and ADCL, the ADC data
register)
  ADMUX = 0x40;                          // AVCC with external capacitor at
AREF pin

  // set ADC converter register A
  ADCSRA = 0;
  ADCSRA |= _BV(ADEN);                   // enable ADC
  ADCSRA |= _BV(ADATE);                  // ADC auto trigger enable
  ADCSRA |= _BV(ADIF);                   // ADC interrupt flag
  ADCSRA |= _BV(ADIE);                   // ADC interrupt enable
  ADCSRA |= _BV(ADPS2);                  // set prescaler to 16

  // set ADC converter register B
  ADCSRB = 0;
  ADCSRB |= _BV(ADTS2) | _BV(ADTS1);    // Timer/Counter1 overflow as trigger
source
}
```

#### 3.2.3.2 ADC Interrupt After One Conversion

Once a conversion is completed, the program interrupts and begins the ISR function. The conversion results are stored in the ADC register and are saved to Vspst and Vdec array alternately. Once decCounter and spstCounter reach MAX_RESULTS, the program stops ADC and calculates the demodulation.

```
ISR(ADC_vect) {
  TIFR1 |= _BV(TOV1);  // Clear the pending Timer1 Overflow Interrupt

  if (decCounter >= MAX_RESULTS && spstCounter >= MAX_RESULTS) {
    ADCSRA = 0;
  } else {
    switch (ADMUX) { // switching between A0 and A1
      case 0x40:
        Vspst[spstCounter++] = ADC;
        ADMUX = 0x41;
        break;
      case 0x41:
        Vdec[decCounter++] = ADC;
        ADMUX = 0x40;
      default:
        break;
    }
  }
}
```

### 3.2.4 Setup Function

```
void setup() {
  Serial.begin(115200);
  pinMode(11,OUTPUT);  // D11 set to output

  // ---- I2C begin
  Wire.begin();

  sei();

  setupTimer2();
  setupTimer1();
  setupADC();
}
```

### 3.2.5 Loop Function

The loop function will wait until MAX_RESULTS number of data points are collected on each detected signal and PWM signal. Once the condition is met, the program calculates the demodulation by finding the mean of the signal value when the switch is on and the mean of the signal value when the switch is off. To decide what Vdec[i] belongs to, we look at the value of the corresponding Vspst. Once the mean of the detected signal in each state is acquired, we take the difference to get the final demodulated data. This data is the water vapor signal we are looking for.

```
void loop() {
  if (decCounter == MAX_RESULTS && spstCounter == MAX_RESULTS) {
    // All samples have been collected.
    uint32_t Von = 0;
    uint32_t on_cnt = 0;
    uint32_t Voff = 0;
    uint32_t off_cnt = 0;
    int32_t demod = 0;

    for (int i = 0 ; i < MAX_RESULTS ; i ++) {
      if (i != 0 && i != MAX_RESULTS - 1) { // ignore edge values
        if (Vspst[i - 1] < 512 && Vspst[i + 1] < 512) { // 1024/2 = 512
          Von += Vdec[i];
          on_cnt ++;
        }
        else if (Vspst[i - 1] > 512 && Vspst[i + 1] > 512) {
          Voff += Vdec[i];
          off_cnt ++;
        }
      }
    }

    // ---- demodulation calculation
    demod = Voff/off_cnt - Von/on_cnt;

    // ----- I2C
    char x[8];
    dtostrf(demod, 8, 2, x);
    Wire.beginTransmission(8); // transmit to device #8
    Wire.write(x);             // sends the given value
    Wire.endTransmission();    // stop transmitting
    Serial.println(demod);
    Serial.flush();

    delay(1);
    // ---- reset variables and turn on ADC
```

```
    // ----- I2C
    char x[8];
    dtostrf(demod, 8, 2, x);
    Wire.beginTransmission(8); // transmit to device #8
    Wire.write(x);         // sends the given value
    Wire.endTransmission();     // stop transmitting
    Serial.println(demod);
    Serial.flush();

    delay(1);
    // ---- reset variables and turn on ADC
    decCounter = 0; //reset counter
    spstCounter = 0;

      // set ADC converter register A
    ADCSRA = 0;
    ADCSRA |= _BV(ADEN);                    // enable ADC
    ADCSRA |= _BV(ADATE);                   // ADC auto trigger enable
    ADCSRA |= _BV(ADIF);                    // ADC interrupt flag
    ADCSRA |= _BV(ADIE);                    // ADC interrupt enable
    ADCSRA |= _BV(ADPS2);                   // set prescaler to 16
  }
}
```

### 3.3 User Interface

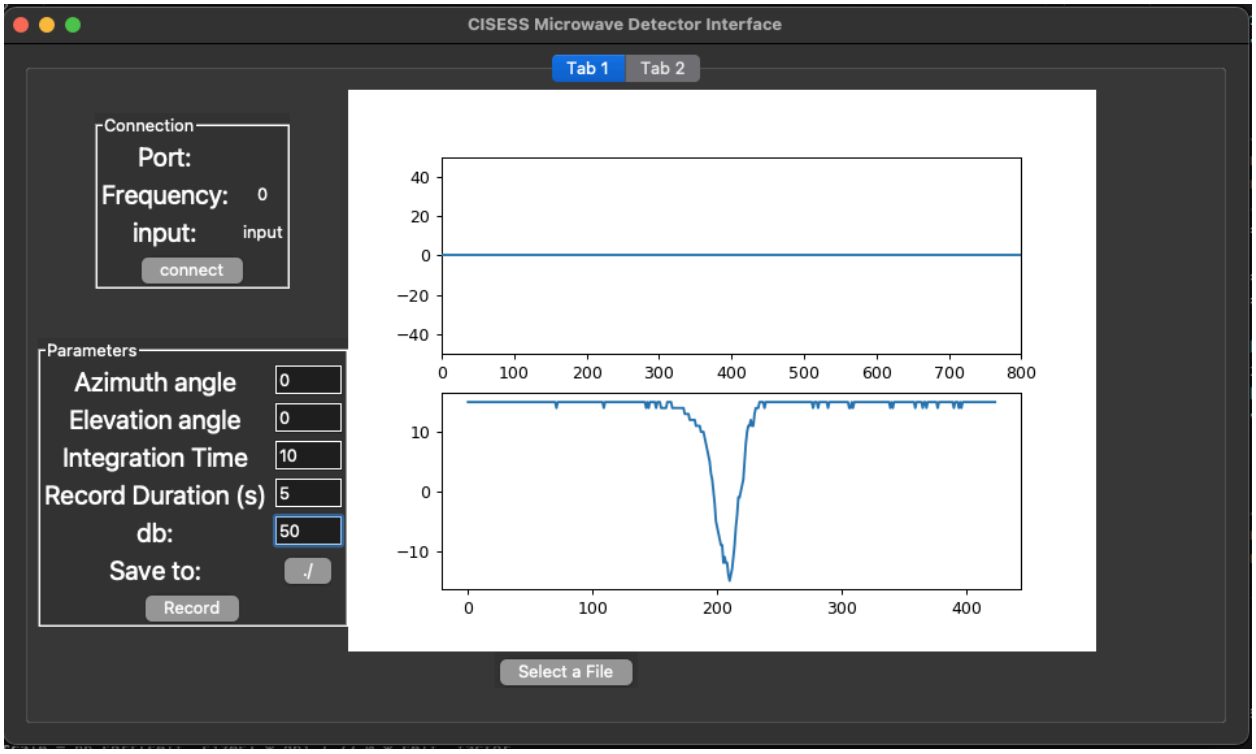



*Figure 8. Python graphical user interface to monitor and record data output from the microwave radiometer*

The data acquisition interface monitors and displays real-time water vapor signals detected by the microwave radiometer. This interface has a record feature that can record water vapor changes over time and record the azimuth and elevation angle the instrument is pointing at.

## 4. Validation and Results

### 4.1 ADC Readings from A0 and A1 Pin

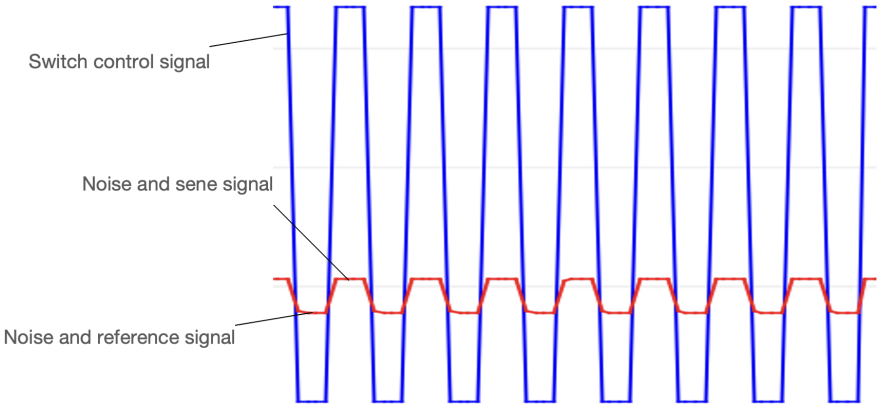
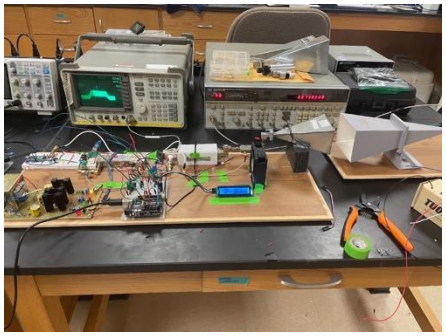



*Figure 9. Readings from A0 and A1 are graphed with serial print in the Arduino IDE*

In figure 9, the blue square wave represents the PWM signal that controls the on/off of the switch and the red square wave represents the detected signal that is detected from the power detector. This graph verifies that the detected square signal is synched with the PWM signal.

### 4.2 Device Calibration

To validate our microwave radiometer, we tested the device against a signal generator generating 22GHz at various intensities. We record and plot the output of the radiometer on the signal intensity from -70dBm to -40dBm. The plot shows a logarithmic relationship. After we convert the intensity to power in millivolts, the curve shows a linear relationship. These graphs validate our microwave radiometer.



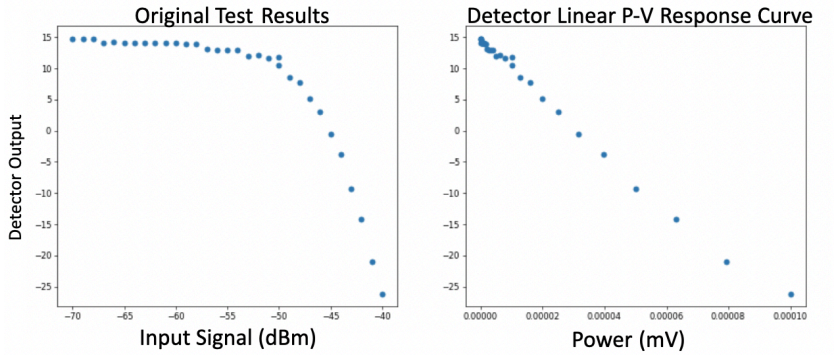



*Figure 10. Device calibration*

### 4.3 Indoor Experiment: 2D Scan on Signal Generator

In the indoor experiment, we mount our microwave radiometer on a telescope tripod, so it points directly towards a fixed signal source from the signal generator. We scan a 2D image of the signal source with an azimuth range of 40 degrees with a 1-degree increment and an elevation range of 25 degrees with a 2.5-degree increment. We adjust the azimuth angle and elevation angle manually and record the device output for 5 seconds. Finally, we plot the output on a 2D map as in Figure 12.

*Figure 11. The microwave radiometer is mounted on a telescope tripod and pointed toward the signal source. Azimuth angle and Elevation angle are adjusted manually.*

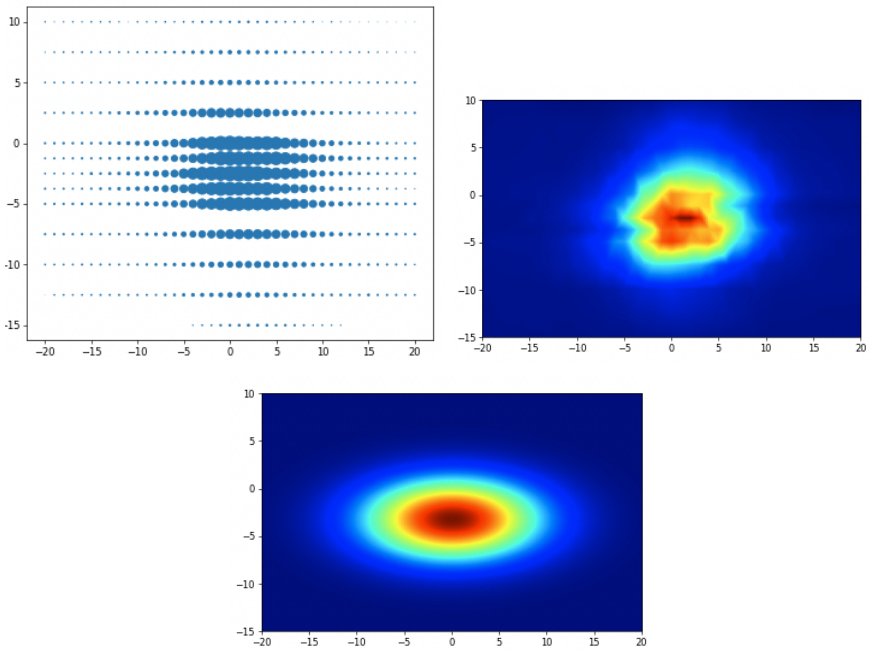


*Figure 12. From left to right: plot of the 2D scan, linear interpolation based on the 2D data, gaussian fit on the interpolated data*

### 4.4 In-situ Experiment: Water Vapor Changes on Rainy Day

*Figure 13. Setups for the in-situ experiment*

For the in-situ experiment, we put our microwave radiometer against a window and pointed it directly to the sky during a rainy day as in Figure 13. We record water vapor changes in the atmosphere for 10 minutes during heaving rain and light rain.

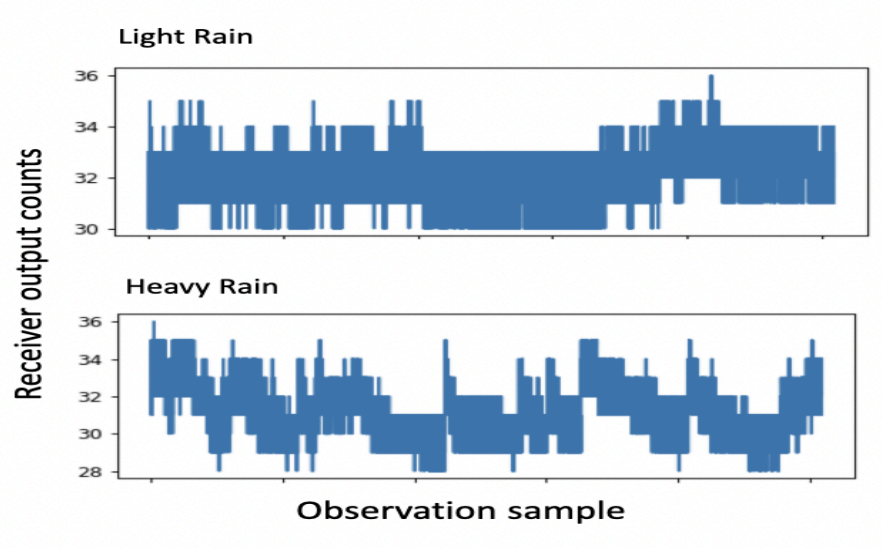



*Figure 14. Results of our in-situ experiment*

The results are shown in Figure 14. The microwave radiometer can observe variations in atmospheric water vapor. Signal collected during light rain has less variation compared to signals collected during heavy rain.

## 5. Conclusion

This project proves the possibility of building a workable low-cost microwave radiometer and shows the possible future application with this device. This device can be used to detect and monitor weather conditions. Furthermore, by replacing the feed horn antenna with another antenna that collects different frequency bands, we can detect other molecules and substances in the atmosphere. Future works include mounting the radiometer on an electronic tripod or on a drone to make data collection easier, as well as output data to an external machine for a more detailed analysis.