POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Comparative characterization of long-range capacitive sensor frontends

Supervisors

Prof. MIHAI LAZARESCU

Prof. LUCIANO LAVAGNO

Candidate

WENLU CUI

June 2021

Summary

At present, indoor person and object positioning has a key impact on people's daily production and life. Sensors are one of the key enablers, and sensors based on different technologies have become the focus of attention of researchers. They range from imaging techniques (visible and infrared camera), capacitive sensing, microwave (RFID, Wi-Fi), radar (LIDAR, ultrasound), inertial sensors dead reckoning (gyroscope and accelerometer). However, besides advantages, different sensors have their own unavoidable limitations.

This thesis focuses on the characterization of capacitive sensors front-end characteristics for long range sensing. One of the main reasons for using capacitive sensors is low cost, low power consumption, and effective sensing. Capacitive sensors are widely used (such as smartphones, wearable devices, the automotive industry and smart objects, etc.).

Currently, most research is focused on short-distance applications of capacitive sensors. The purpose of this thesis work is to characterize their long-range characteristics.

The front-end used in this research is a period modulator based on a 555 timer IC produced by TI, used as astable oscillator with the period depending on the capacitance of the sensor plate with the environment. The output frequency varies based on externally connected resistors and a capacitor. When the values of the resistors are fixed, the output frequency is only related to the value of the capacitor C. In my case, the capacitor C is the total capacitance of the sensor plate, including the mutual capacitance with the environment and the human body. When the distance between the human body and the metal plate changes, the mutual capacitance will change accordingly and the output frequency of the oscillator will also vary. Thereby, a relationship is established between the distance to the object of interest and the front-end oscillation frequency. By analyzing this relationship, we can find the distance-oscillation frequency characteristic of the capacitive sensor.

To accurately analyze this relationship, we use a complete data processing system. The system is based on the ATmega328P microcontroller on the Arduino UNO board, which collects and processes data. We use also XBee PRO 802.15.4 RF modules to configure the sensor and transfer the data. Finally, the Matlab

toolbox is used to plot the sensor data for analysis. The Arduino UNO board has low cost and sufficient interfaces. A Wireless SD shield allows an Arduino board to communicate wirelessly using a wireless XBee module, suitable for this project. On the data processing side, we focus on accurate frequency measurement on-board the sensor node. We measure the duration of multiple periods of the sensor front-end oscillation to reduce the measurement quantization error and get an accurate frequency value.

In the second chapter, I describe the hardware structure of the system in detail and introduce the performance and application of the main modules. Including the schematic of the 555-based oscillator circuit, the initialization and configuration of the ATmega328P microcontroller, the installation and configuration of XBee PRO modules, and the connection between the Arduino UNO board, the XBee module, and the 555 oscillator circuit. At the same time, the main frequency measurement algorithm is explained in detail, and the Matlab script for plotting processed data is briefly described.

The front-end capacitance measurement technique is however very susceptible to environmental interferences, both electromagnetic and from other objects. To characterize these dependencies, I compared during the tests the measurements using plates of different sizes in different environments. In indoor conditions, with no interferences in the room and a 45 cm $\times$ 16 cm metal plate, the oscillation frequency-distance to human body dependency has good stability up about to 2 m, while the sensing distance can exceed 3 m. In outdoor conditions, using a 45 cm $\times$ 50 cm metal plate, the sensing distance of human body can reach 20 m, but the stability is relatively poor due to environmental and device constraints. I then check the characteristics to a large-scale reference object, a car in two conditions: to the rear and to the side. The car (van) rear surface area is 1.76 m $\times$ 1.74 m, while the side surface area is 4.16 m $\times$ 1.74 m. As expected, the results showed that the larger the surface area of the reference object, the stronger the sensing and the better the stability. For the rear of the car, the sampling stability is greatly improved and the sample changes within 15 m are apparent. In contrast, the sensitivity difference on the side of the car is not much different, and the sampling stability is better than that of the rear.

The outdoor tests show some strong variations as the distance changes. They can be due to underground structures, such as sewage, and to interference between the sensor and the ground itself, which is much closer to the sensor than the object at long distances.

As expected, the experimental data shows that the size of the capacitive sensor plate is strongly correlated with the sensing distance. The larger the plate, the farther the sensing distance. At the same time, the larger the target surface area, the better the sensing and lower the environmental interference. However, the performance of the capacitive sensors is susceptible to environmental noise. Metal objects, human bodies and even underground sewage near the test site will reduce the sensing distance and stability.

For future research, we can explore the main environmental factors that affect the stability and sensitivity of capacitive sensors outdoors, as well as the reasons for unexpected but unavoidable data fluctuations. However, as far as the current test system is concerned, its performance can also be improved by some methods. The simplest is to choose a better test site that avoids interference as much as possible. In addition, the height of the metal plate can be increased, and the coupling capacitance between the sensor and the ground can be reduced. Also, some feasible filtering techniques can be applied to remove some noise in current circuits. The most effective method should be to use more powerful microprocessors and receiving and sending modules, which can fundamentally improve the performance of the test system.

Acknowledgements

First of all, I would like to thank Professor Mihai Lazarescu and Professor Luciano Lavagno for their guidance and full support for my thesis.

I would also like to thank Osama Bin Tariq and Junnan Shan for sharing their knowledge and help when encountering difficulties.

Also, I would like to thank my friend Chen Ze for sharing my happiness and worries during this period.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Capacitive sensing

Capacitive sensing is a technology based on capacitive coupling, which is currently widely used in smartphones, touch screens, laptops and wearable devices.

As shown in Figure 1.1 from [1], capacitance exists between the human body, their equipment and the environment. Through measurement and analysis, capacitive sensors can infer some required information. But the inherent coupling capacitance between objects will also reduce the sensitivity and accuracy of the capacitive sensor.

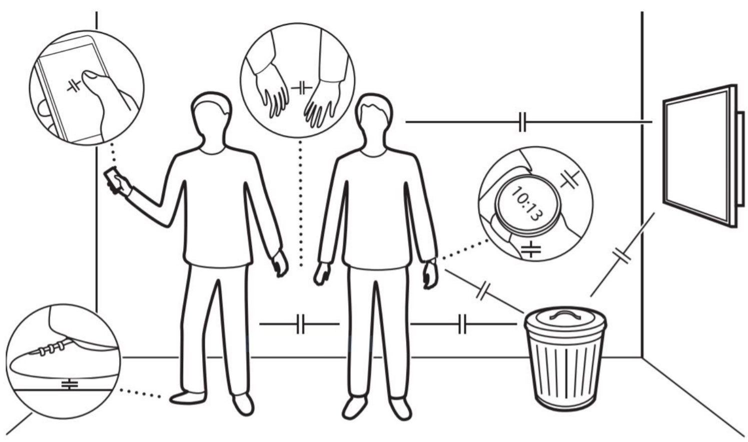


Figure 1.1: Coupling capacitances between body and environment from [1]

Referring to Figure 1.2 from [2], The overall capacitance of a single plate capacitive sensor depends on 4 types of capacitances related to the environment.

1. the capacitance between the sensor and ground (C_{sg})
2. the capacitance between the human body and ground (C_{bg})
3. the capacitance between the sensor and the environment (C_{se})
4. the capacitance between the sensor and the human body (C_{sb})

The last depends on the distance, d, between the sensor and the human body [2].

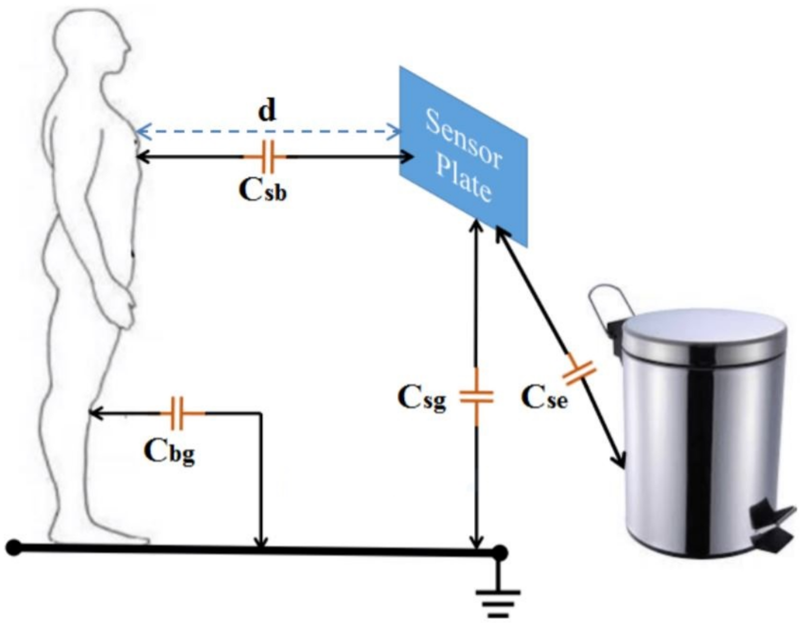


Figure 1.2: Capacitance of a single plate capacitive sensor from [2]

1.2 Operating mode

Capacitive sensing technology can be divided into four working modes: loading, shunting, transmitting and receiving [1].

Loading mode is the simplest and most common type of capacitive sensing, and its transmitting and receiving electrodes are the same. The body capacitively loads the electrodes and causes a displacement current to flow through the body to the ground. As the body gets closer to the electrode, the capacitive coupling increases and the displacement current also increases [1].

In the shunt mode, the transmitting and receiving electrodes are different, so there is some capacitive coupling between them. When the human body is close to the electrode, it will capacitively couple with the transmitting electrode and the receiving electrode in roughly the same order of magnitude as the coupling between the electrode. This will cause the displacement current to flow through the body to the ground, thereby reducing the displacement current flowing from the transmitting electrode to the receiving electrode [1].

Transmit mode is similar to shunt mode, except that the body is very close to the transmitter. In this mode, the coupling between the body and the transmitter is much greater than the coupling between the body and the receiver or between the transmitter and the receiver. The body acts as an extension of the emitter electrode. When the body approaches the receiving electrode, the displacement current entering the receiving electrode increases [1].

Receive mode is the inverse of transmit mode, that the body is very close to the receiving electrode. In this case, the body becomes an extension of the receiving electrode, and the operating principle is the same as the transmitting mode [1].

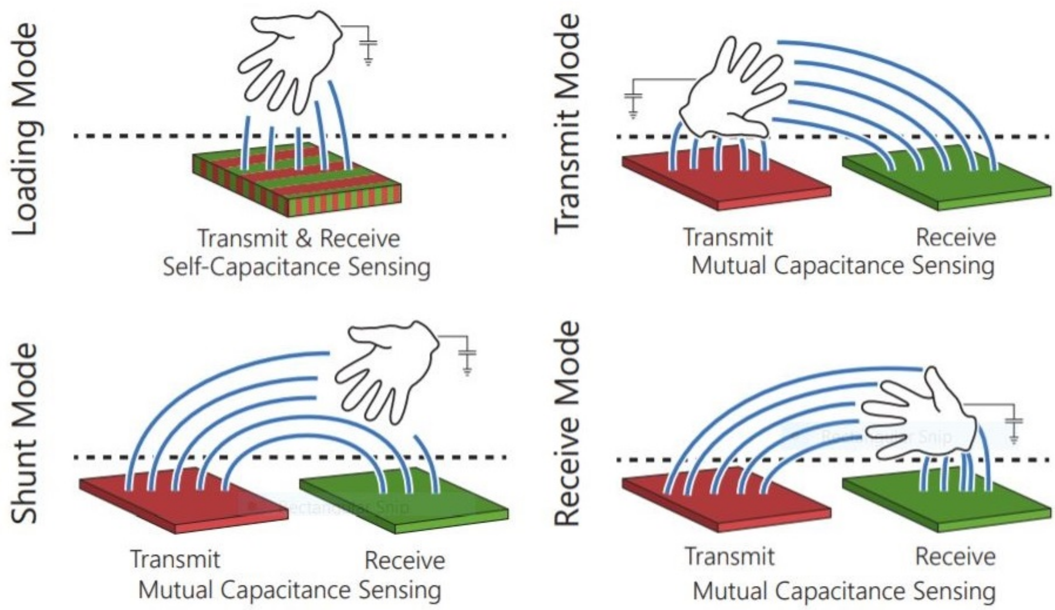


Figure 1.3: Capacitive sensing operating mode from [1]

Chapter 2

Research project

2.1 Previous work

In [3],the authors introduce an indoor person localization system based on capacitive sensing and localization algorithms that can determine the location with less error in a $3\,\mathrm{m} \times 3\,\mathrm{m}$ room.

the operation of the system consists of:

1. indirect measurement of the capacitance of the transducer by measuring the frequency of a relaxation oscillator whose electrical and timing characteristics depend on transducer capacitance;
2. use of base-band digital filters to attenuate the noise captured by the sensor;
3. use of localization algorithms to infer the position of the person using the data from several load-mode capacitive sensors within one or several rooms.

The oscillator based on a 555 timer integrated circuit (IC) configured in astable mode.

The frequency is independent of the supply voltage. The duration of one full timing cycle is equal to the sum of the two individual times that the capacitor charges and discharges added together and is given as:

$$\mathrm{T} = \mathrm{t}_1 + \mathrm{t}_2 = 0.7(\mathrm{R}_1 + 2\mathrm{R}_2)\mathrm{C} \tag{2.1}$$

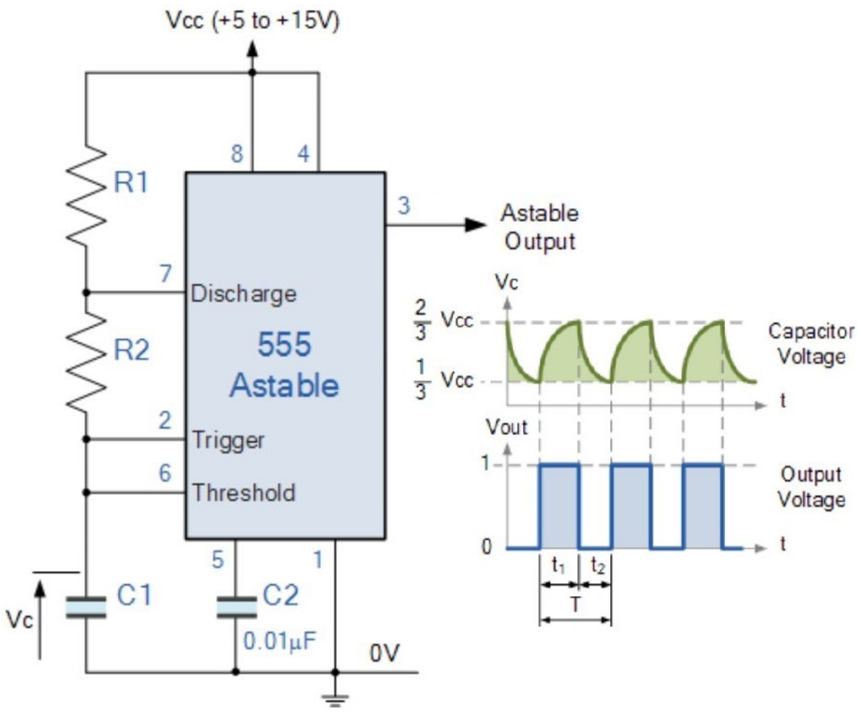


Figure 2.1: Basic astable 555 Oscillator circuit from [4]

the output frequency of an Astable 555 Oscillator as:

$$\mathrm{Frequency} = \frac{1}{\mathrm{T}} = \frac{1}{0.7\,(\mathrm{R}_1 + 2\mathrm{R}_2)\,\mathrm{C}} \tag{2.2}$$

In their case, they replaced the fixed value capacitor C1 with a metal plate, as shown in Figure 2.2, and set $\mathrm{R}_1 = 200\,\mathrm{k}\Omega$ and $\mathrm{R}_2 = 560\,\mathrm{k}\Omega$.

It can be seen from (2.2) that because the resistances values are fixed, the output frequency is only related to the capacitance C, while the capacitance C is only related to the distance between the human body and the metal plate. Therefore, the change in the output frequency reflects the change in the relative position of the human body and the plate.

They used the full-period measurement method to get a more accurate frequency change. Through the data communication between the two XBee modules, the measurement data is uploaded from the sensor to the PC for processing. Applying different sizes of metal plates ($4\,\mathrm{cm} \times 4\,\mathrm{cm}$, $8\,\mathrm{cm} \times 8\,\mathrm{cm}$, $16\,\mathrm{cm} \times 16\,\mathrm{cm}$) and different algorithms (1-NN algorithm, 56-NN algorithm, Naïve Bayes (NB)

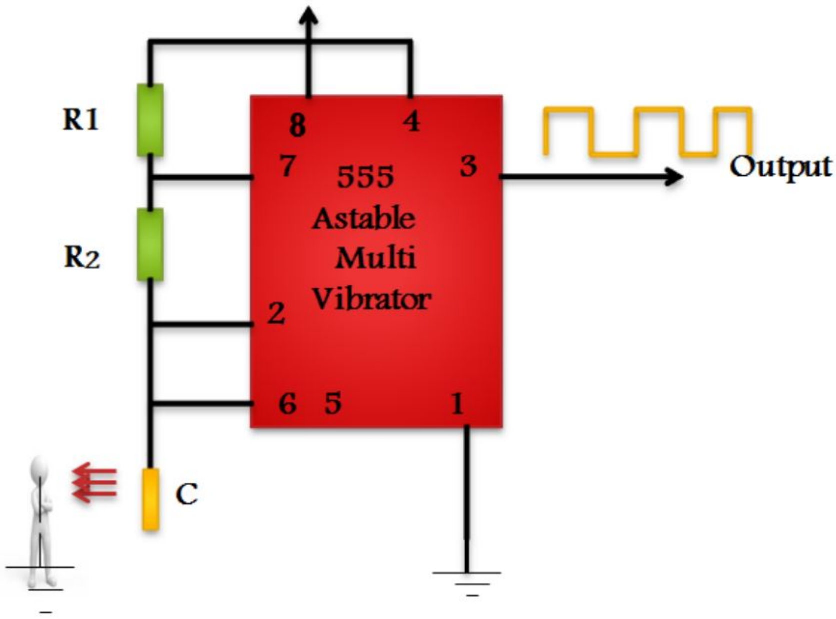


Figure 2.2: 555 astable multivibrator circuit set up from [3]

algorithm, Support Vector Machines (SVM) algorithm), the results show that the localization of the NB algorithm using a 16 cm × 16 cm metal plate is very accurate.

In [5],[6],[7],[8], they describe the characteristics and applications of capacitive sensors from different aspects, and its content and related technologies have greatly inspired my thesis.

This research will continue to use the 555 astable oscillation circuit to explore the factors that affect the stability and sensitivity of the capacitive sensor and to characterize the long-range front-end characteristics of the capacitive sensor.

2.2 Project description

In my research, I use the 555 astable oscillation circuit to present the long-range front-end characteristics of the capacitive sensor according to the relationship between the distance between the human body and the plate and the output frequency in this circuit. The key points: Data collection and communication, Data processing, Noise rejection.

For data collection and communication,the data sampling system is based on the ATMEGA328P microcontroller [9] on the ARDUINO UNO board and using the Digi XBee-PRO module [10] for communication.The ARDUINO UNO board has low cost and sufficient interfaces, and there is a Wireless SD shield [11] that allows an Arduino board to communicate wirelessly using a wireless XBee module, which is very suitable for this project. In order to achieve data collection and communication, I use two sets of ARDUINO UNO boards and XBee modules (one set is used for data collecting and data transmitting, the other set is used for data receiving).

For data processing,in order to get the accuracy of the research results, accurate data processing is essential, which involves two processing procedures: frequency measurement and clear figure. I use the full-period measurement method [12] to ensure the accuracy of the measurement frequency, and use the Matlab tool to clearly reflect the received data in the figure for later comparison and analysis.

For noise rejection, since the noise in this research mainly comes from the coupling capacitance generated by the interference objects (metal objects, human body, etc.) in the environment. In order to suppress the environmental noise, my method is to reduce the interference objects around the test place and increase the effective coupling capacitance, which is coupling between the body and metal plate(such as replacing the human body with a vehicle, as the test subject).

2.3 Implementation

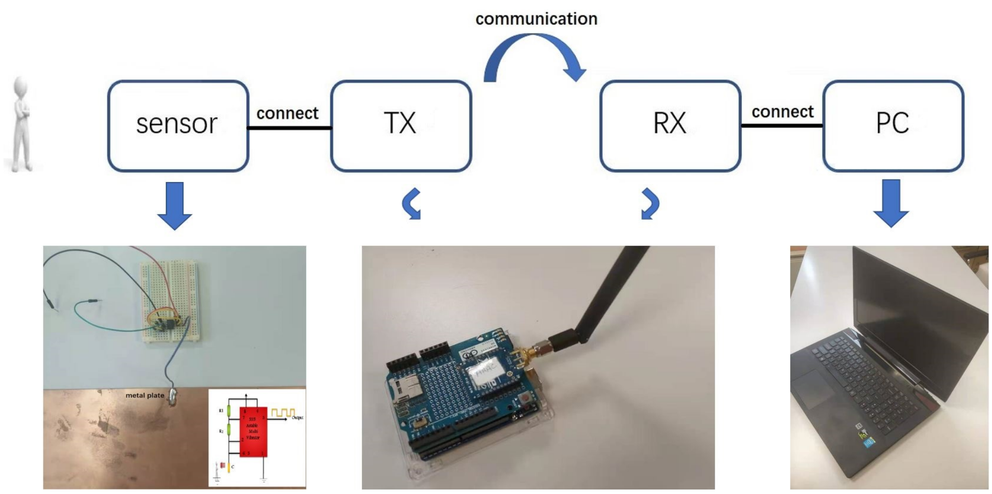


Figure 2.3: The framework of the system

Figure 2.3 presents the overall framework of the system. The hardware part involves the construction of the 555 astable oscillation circuit and the installation and configuration of the XBee modules; the software part includes frequency measurement and data visualization.

2.3.1 Hardware Implementation

555 astable multivibrator circuit set up

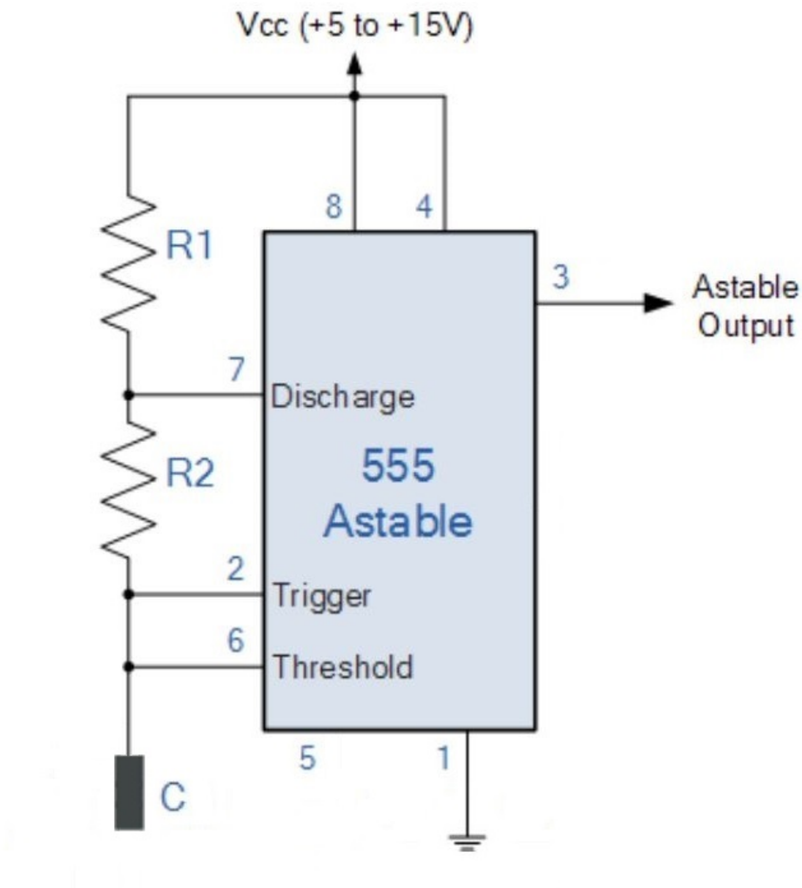


Figure 2.4: 555 astable circuit from [4]

The commonly used NE555 chip is selected, and connected in astable mode. In the circuit above, pin 2 and pin 6 are connected together allowing the circuit to re-trigger itself on each and every cycle allowing it to operate as a free running oscillator. During each cycle capacitance, C, which is the coupling between the human body and the plate in my case, charges up through both timing resistors, R_1 and R_2 but discharges itself only through resistor R_2, as the other side of R_2 is connected to the discharge terminal, pin 7.

From the capacitance formula (2.3):

$$C = \frac{\varepsilon A}{d} \tag{2.3}$$

where A is the area of the conductive plate; ε is the permittivity dielectric; d is the distance.

Therefore, the coupling capacitance decreases as the distance between the human body and the plate increases, the output period T decreases, and the frequency f increases from (2.1), (2.2). Due to this relationship, by observing and analyzing the frequency change of the output waveform, the influence of distance and size of the plate on the characteristics of the capacitance sensor can be obtained.

Configuration and Installation of XBee module

XBee module is a remote low-power wireless module. It is a ZigBee module product of Digi Company in the United States. XBee is the name of the series. XBee module functions both transmitting and receiving data, are widely used in the field of intelligent home, remote controls, sensors, wireless detection. At the same time, Digi's corresponding platform and adapter provide a great convenience for module configuration and debugging.

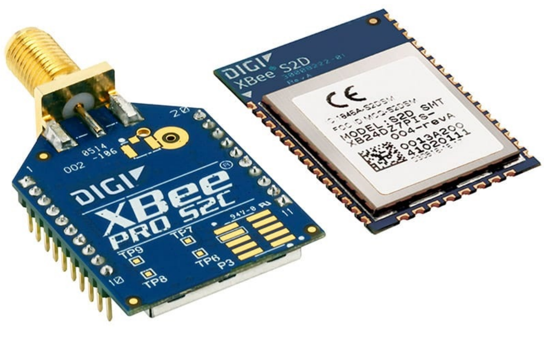


Figure 2.5: XBee module from [10]

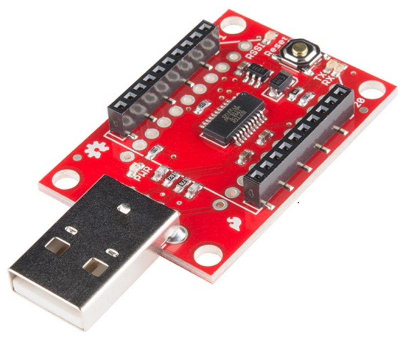


Figure 2.6: XBee Explorer from [10]

In my research, I use two XBee modules for the remote transmitter and receiver respectively. The XBee Explorer was used for configuring the modules on the XCTU platform provided by Digi.

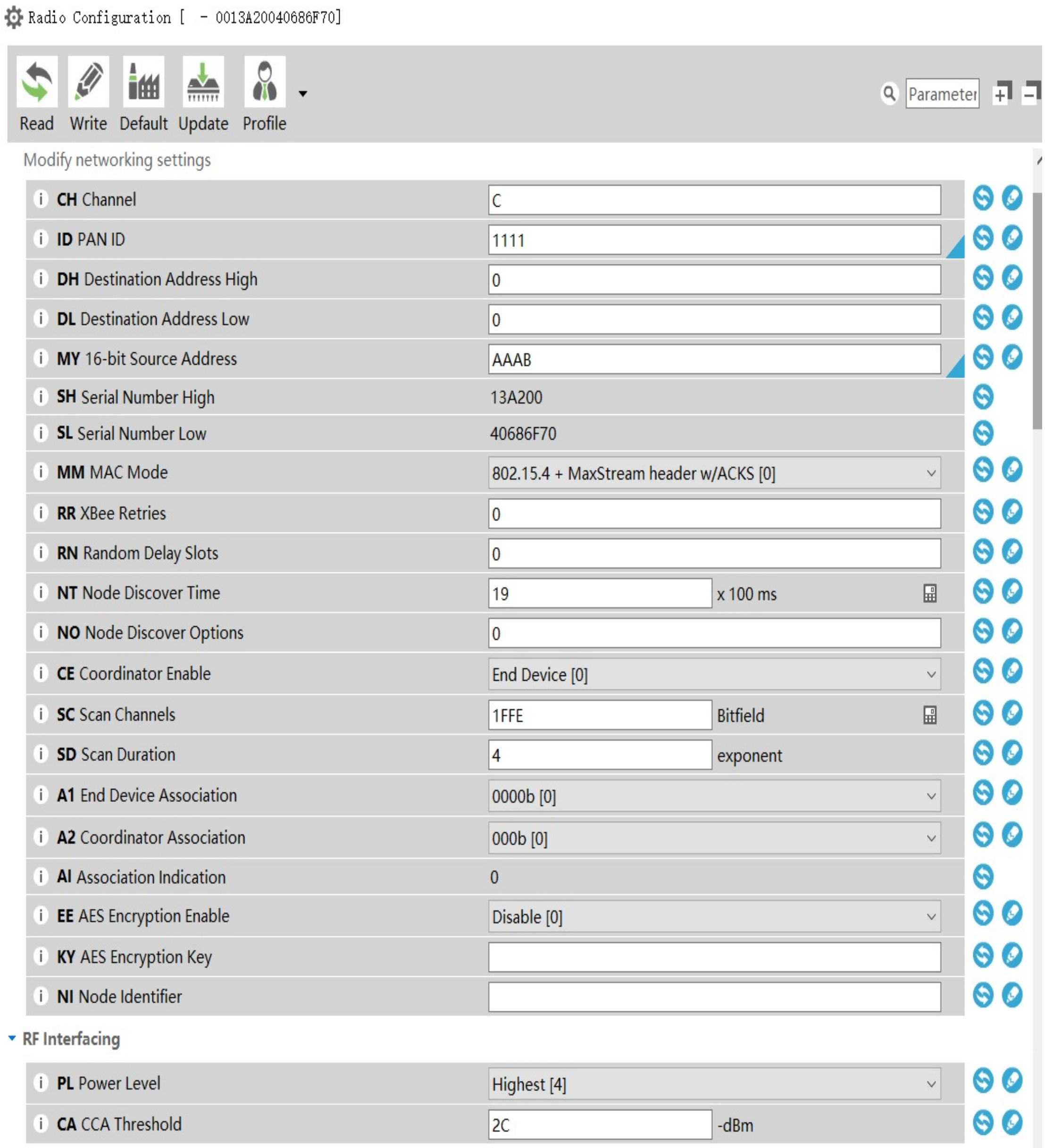


Figure 2.7: XBee module configuration

After the configuration, connecting the microcontroller and XBee module through the Wireless SD shield and then debugging.

There is a very important point that I must mention is that when installing the XBee module, you must pay attention to the correct direction, otherwise, the module will be burned.

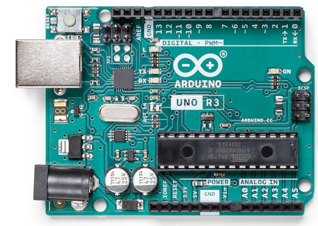


Figure 2.8: ARDUINO UNO from [11]

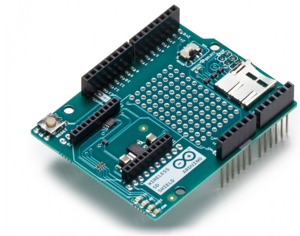


Figure 2.9: Wireless SD shield from [11]

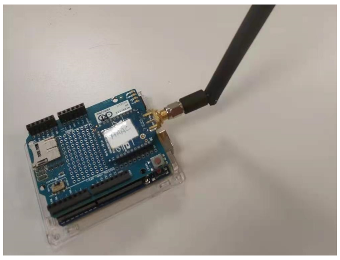


Figure 2.10: Transmitter/Receiver module

2.3.2 Software Implementation

Frequency measurement

As mentioned above, I analyze the characteristics of capacitive sensors by analyzing the changes in output frequency, so accurate frequency measurement is very important. There are two measurement techniques with their own advantages and disadvantages: measuring the number of cycles in a fixed period of time and measuring the time of one cycle.

There is a simple comparison is shown in the Table 2.1:

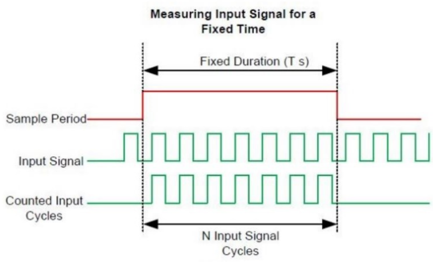
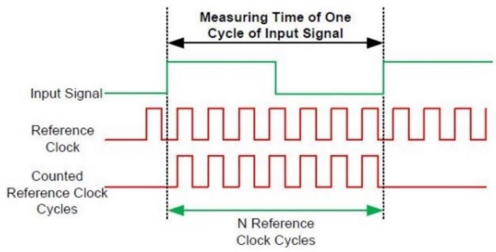
Measure the number of cycles in a fixed period of time	Measure the time of one cycle
	
$\mathrm{f} = \frac{\mathrm{N}}{\mathrm{T}}$	$\mathrm{f} = \frac{\mathrm{f_{clock}}}{\mathrm{T}}$
Good for measuring low frequency; At high frequency error increases.	Better for long time measuring; High error for short time especially for low frequency measuring.

Table 2.1: Frequency measurement methods from [12]

While there is a better method I used which is the full-period measurement from [12].Its principle is shown in Figure 2.11.

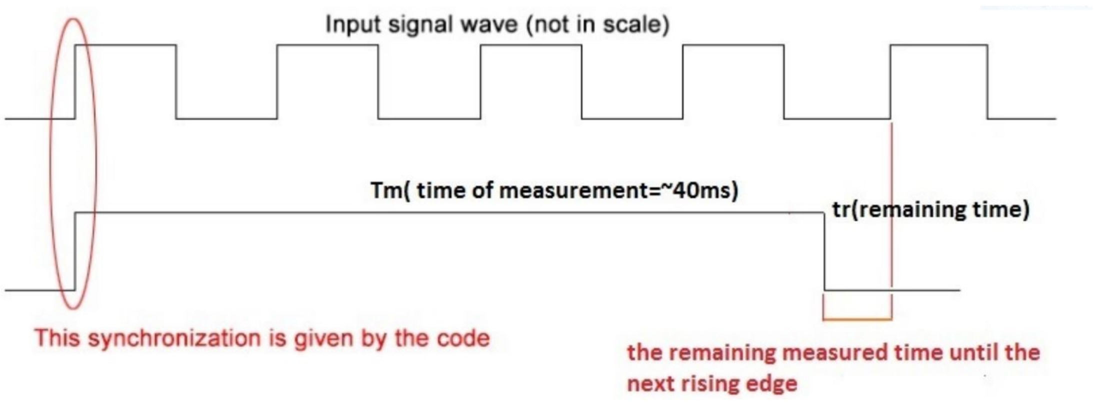


Figure 2.11: Full-period Frequency measurement from [12]

The frequency of the signal for this case is given by (2.4):

$$\text{Frequency} = \frac{\mathrm{N}}{\mathrm{T_m} + \mathrm{T_r}} \tag{2.4}$$

Where N is the number of full periods of the measured signal, T_m is the initial measurement time, and T_r is the remaining time after the measurement time (T_m) elapses until the next rising edge of the measured signal.

In order to achieve full-period measurement, I need to use two timers provided by the ATmega328p microcontroller, one to count the period of the signal to be measured, and one to time the measurement time (T_m).

Next, I will briefly describe some codes of timers configuration and interrupt handle for frequency measurement.

As shown in Figure 2.12, idle timer0 and timer1 by initializing the control register to zero, and clear the count value. At the same time, set PD4/PCINT20 as input through DDRD(Port D Data Direction Register), and turn on port d interrupts, which can be done by PCICR(Pin Change Interrupt Control Register).

```
TCCR0A = 0;                 // reset timer/counter0 control register A
TCCR0B = 0;                 // reset timer/counter0 control register A
TCNT0 = 0;                  // counter value = 0

// timer1 setup / is used for frequency measurement gate time generation with 16MHZ/no prescaling
TCCR1A = 0;
TCCR1B = 0;

f_measurement_running = 0;

//pinchange interrupt
DDRD &= ~(1 << DDD4);     // Set pd4/pcint20 as input
PORTD |= (1 << PORTD4); /* Activate PULL UP resistor */
//pin PD4 is now an input with pull-up enabled

PCICR |= 0b00000100;       // turn on port d interrupts (PCINT[23:16])
```

Figure 2.12: Timers initialization and configuration

In Figure 2.13, two independent overflow interrupt routines are used to count the overflow times of the two timers. At the end of the measurement time, the pin change interrupt is enabled to wait for the last rising edge.

```
// if Timer/Counter0 overflow flag
ISR(TIMER0_OVF_vect)
{
    count_ovf_t0++;             // count number of Counter0 overflows the external signal
}

// if Timer/Counter1 overflow flag
ISR(TIMER1_OVF_vect)
{
    count_ovf_t1++;

    if (count_ovf_t1 >= measurement_window_expected_duration)
        PCMSK2 |= (1 << PCINT20); // Look for next input signal rising edge (end measurement).

}
```

Figure 2.13: Overflow handles

The detail of the operations of the pin-change interrupt handle is shown in Figure 2.14, which completes most tasks of the measurement process. In the beginning, synchronizing the two counters, stop the counters at the end of measuring time, and extract the values to be used for final computation.

At the start, when the code detects the rising edge of the input, it resets the two counters and their overflow counts, then starts counting, enables the overflow interrupt handler, sets the measurement end flag, and disables pin-change interrupt until the end of the measurement. When the measurement is over, stop the counter, disable overflow interruption, extract the count value, reset the measurement end flag, and prepare for the next measurement.

```
ISR(PCINT2_vect)                // Port d, PCINt20
{
    char statepin = 0;
    // No raising edge of input signal
    statepin = PIND;
    if ((statepin & 0x10) != 0x10)
        return;
    // Raising edge of input signal.
    // Start a new measurement.
    if (f_measurement_running == 0) {
        // Disable PCint until the end of measurement window.
        PCMSK2 &= ~(1 << PCINT20);
        // Reset counters and their overflow counters.
        TCNT1 = 0;
        TCNT0 = 0;
        count_ovf_t0 = 0;
        count_ovf_t1 = 0;
        // Start measuring the time.
        TCCR1B |= (0 << CS12) | (0 << CS11) | (1 << CS10); //no prescaling internal clock source
        // Connect input signal as Timer0 clock (rising edge of external signal, no prescaling)
        TCCR0B |= (1 << CS02) | (1 << CS01) | (1 << CS00);
        // Enable counter overlow interrupts.
        TIMSK1 |= (1 << TOIE1);
        TIMSK0 |= (1 << TOIE0);
        f_measurement_running = 1;
    }
    // End of the measurement.
    else {
        // Disable pin change interrupts.
        PCMSK2 &= ~(1 << PCINT20);
        // Stop counters.
        TCCR1B = TCCR1B & ~7;
        TCCR0B = TCCR0B & ~7;
        // Disable counter overflow interrupts.
        TIMSK1 &= ~(1 << TOIE1);
        TIMSK0 &= ~(1 << TOIE0);
        // Get remainders from counters.
        remaineder_clocks = TCNT1;
        remaineder_input_periods = TCNT0;
        f_measurement_running = 0;
    }
}
```

Figure 2.14: Pin-change interrupt handle

Graphical results

The code of Matlab drawing is much simpler than the code of frequency measurement. The flow chart is shown in Figure 2.15.

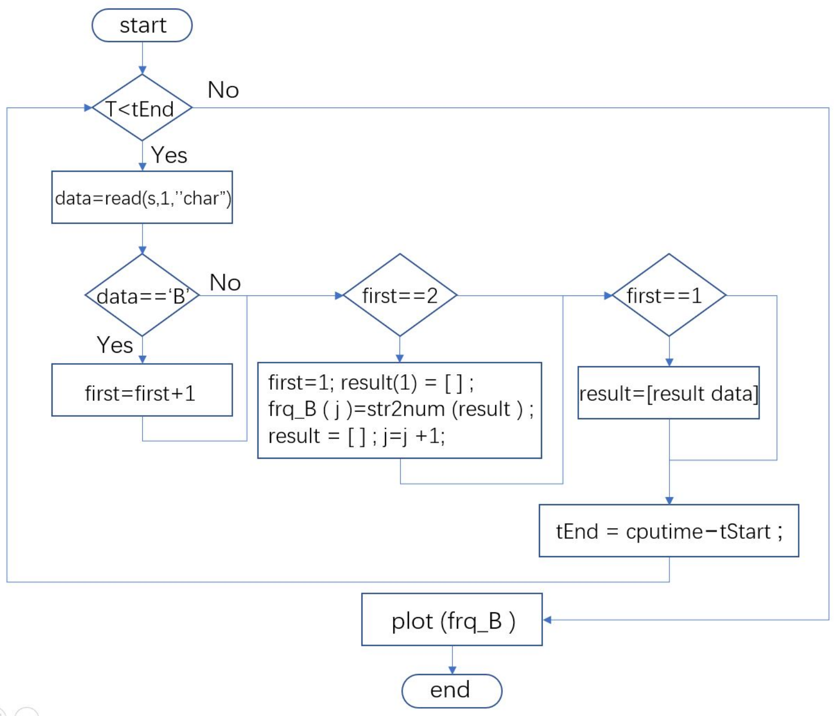


Figure 2.15: The flow chart of Matlab code

The main idea is to extract intact digital characters, that is, the measured value of frequency, from character strings such as "xxxBxxxxxBxxxxxBxx" (where 'x' refers to digital characters,'B' refers to the label of the transmitter), and then plot the extracted data for analysis. The detail of the implementation code is shown in Figure 2.16.

```
%set start and end time
tStart = cputime;
tEnd=0;

while(tEnd<360)
    data=read(s,1,"char");

    % the format of transmitted data is sameting like "xxxBxxxxxBxxxxxxBxxxx..."
    if data=='B'        % transmittor label,the data from transmittor 'B'
        first=first+1;  %  the flag to record the times 'B' comes
    end

    if first==1         % after the first B comes,record next serveral data which is the frequency value.
       result=[result data];
    end

    if first==2         % the second time B comes means the second frequency value comes, save result and set flag
        first=1;
        if result(1)=='B'
            result(1)=[];
            frq_B(j)=str2num(result);
        end
        result=[];
        j=j+1;
    end

    tEnd = cputime-tStart;
end
save('frequency_B', 'frq_B');
```

Figure 2.16: Matlab code segment

Chapter 3

Experimental results

The purpose of this research is to explore the long-distance characteristics of the capacitive sensor, so the test in the process is concentrated on the long-distance test of the large metal plate.

I also performed two small-plate short-range tests, in conditions of almost no interference in the room: interval test and continuous test. The purpose is to show the sampling stability and sensitivity of the capacitive sensor in a relatively good environment. The results of these tests can be compared to analyze the size of the capacitor plate impact on the sampling distance and sampling stability of the capacitance sensor.

The equipment used for long-range testing is the same as that of the small plate; only the size of the metal plate is increased. However, the test site requires a large enough space and reduce the surrounding interference. Therefore, I need to test under different environments and characterize its performance and characteristics by comparing and analyzing the results.

The test system is shown in Figure 3.1 and Figure 3.2. These two figures are to visually present the size of the two capacitive sensing plates. I use sensor plates with dimensions of $45\,\text{cm} \times 16\,\text{cm}$ and $50\,\text{cm} \times 45\,\text{cm}$ to be mounted on a bracket composed of boxes. The circuit board is placed on top of the box to reduce noise. Digi International's XBee PRO 802.15.4 RF modules are used as the transmitter and receiver. And use MATLAB to receive the data on the PC, and then plot the data values.

Figure 3.1: sensor plate: 45 cm × 16 cm

Figure 3.2: sensor plate: 50 cm × 45 cm

Figure 3.3: The test environment of small plate

I conducted small plate short-range testing by two methods in a relatively good environment and compare it with other environments. In the environment shown in Figure 3.3, there is almost no interference within the device radius of 2.4 m. The tests are interval sampling and continuous sampling and the results are shown into two graphs, the frequency sample graph presents the sampling situation of the experiment process and the frequency distance graph reflects the sensing distance of the sensor.

For the Interval sampling, there are 9 sampling points, each point interval is 20 cm, and each sampling point is sampled for 10 s. The distance from 0.5 m to 2.3 m. We can see from Figure 3.4 that the sensor maintains stable sampling at each sampling point. Only when the human body moves, the sampling will produce large fluctuations. This is because the change of the coupling capacitance between the human body and the sensor during movement takes a while to stabilize. Figure 3.5 also shows that the sensing distance exceeds 2.1 m.

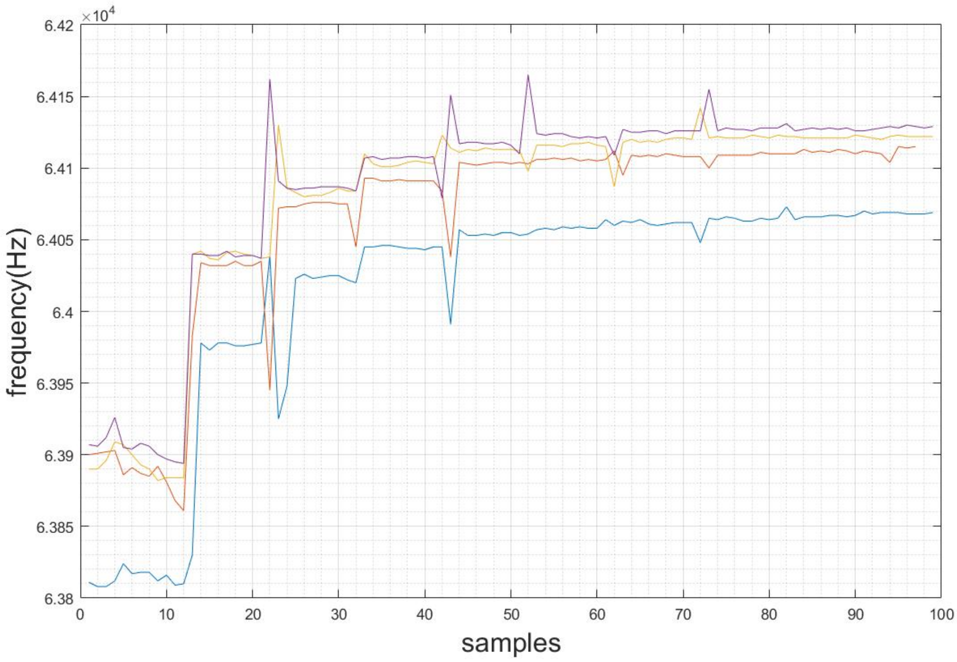


Figure 3.4: Frequency-sample curve: small plate interval test

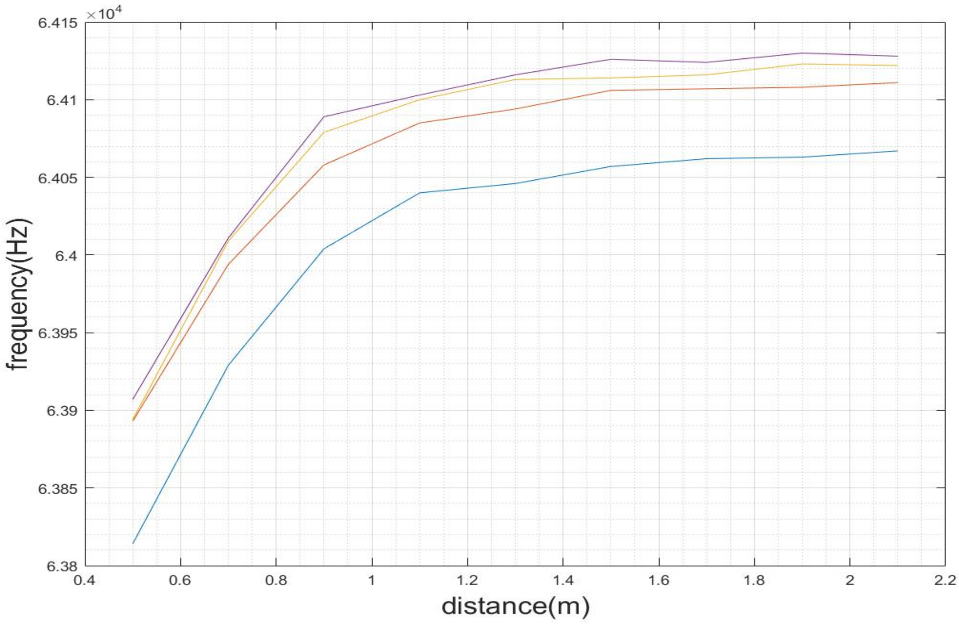


Figure 3.5: Frequency-distance curve: small plate interval test

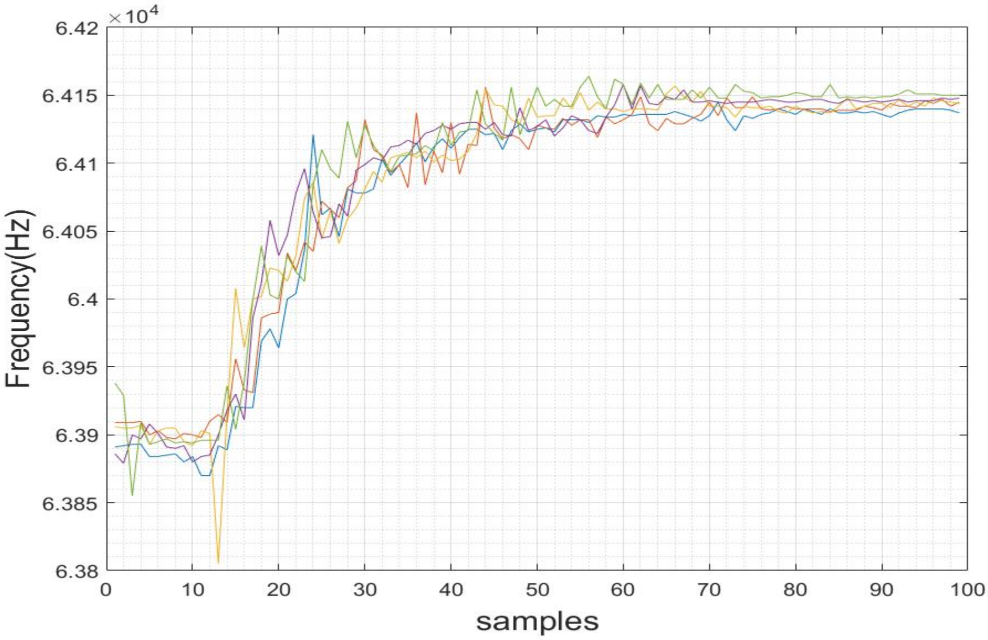


Figure 3.6: Frequency-sample curve: small plate continuous test

Continuous sampling is a process of uniform and continuous sampling while maintaining the same time and distance as interval sampling, which reflects the sensitivity of capacitive sensors. It can be seen from Figure 3.6 that the samples swing a lot because the coupling capacitance has been changing during the movement, but the resulting curve still shows that the sensor can perceive small changes in distance.

Next, we will focus on the long-distance test of the large plate. Figure 3.7 shows the outdoor environment for the test, which is a long terrace to test the stability and sensitivity of outdoor sensing using the plate. Because the test site is relatively narrow, the metal railings and window frames on both sides influence the measurements, but we can evaluate the maximum sensing distance.

Figure 3.7: The first test environment of large plate

The large plate (50 cm × 45 cm) was used to test in the environment of the long and narrow corridor, as shown in Figure 3.7. The results are shown in Figures 3.8 and 3.9.

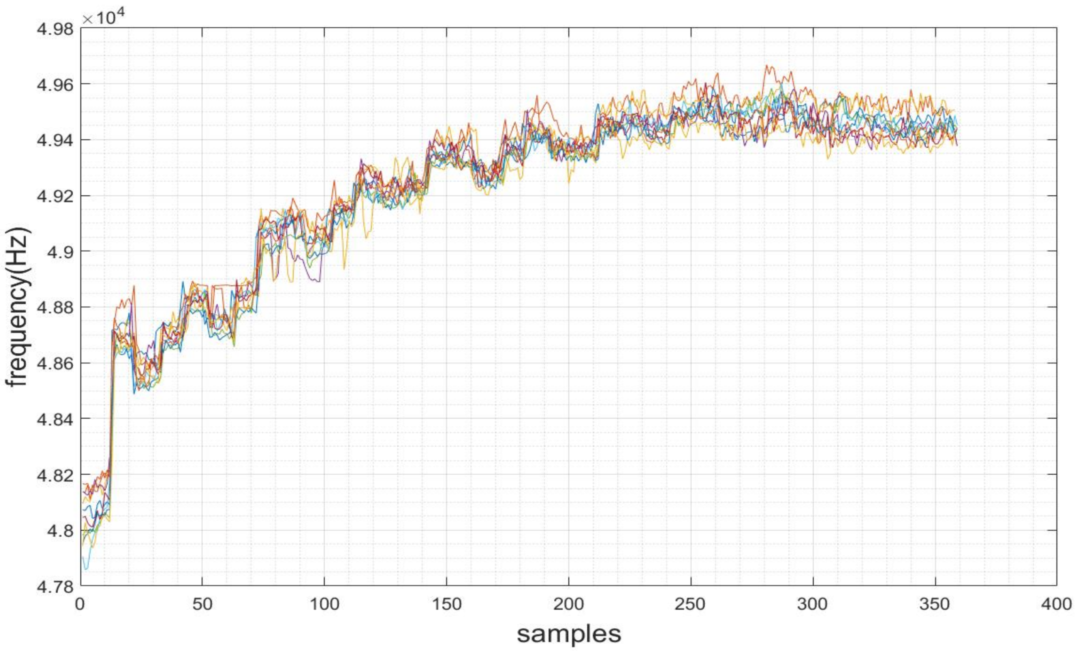


Figure 3.8: Frequency-sample curve: large plate interval test

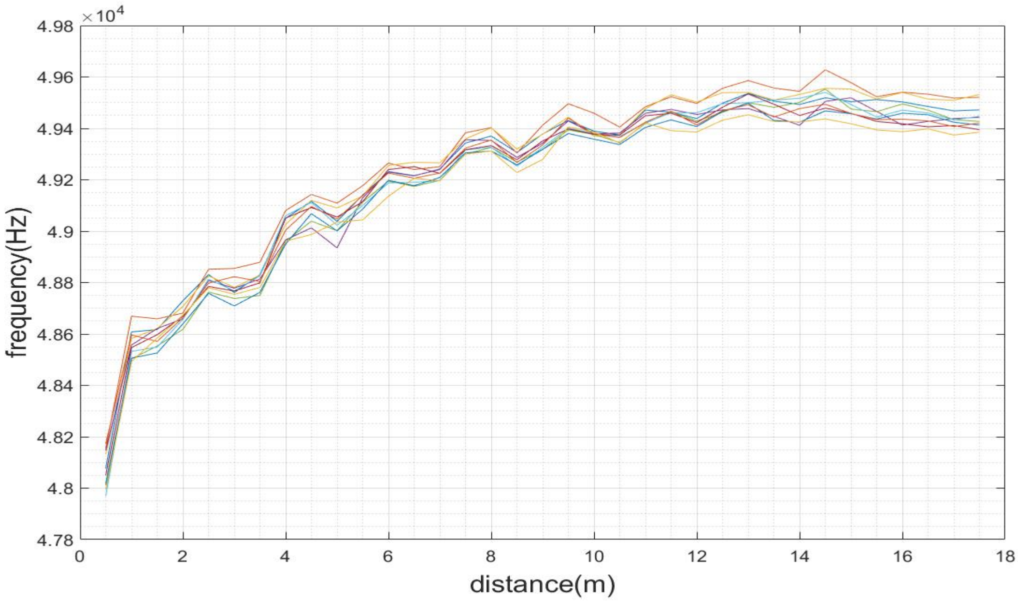


Figure 3.9: Frequency-distance curve: large plate interval test

The interval sampling results look good, especially since the sensing distance can exceed 14 m. However, compared with the previous small-board indoor test, the sampling stability is obviously lacking. We can see that even at each fixed sampling point, the frequency samples still have certain fluctuations, and the fluctuations are greater during movements. Because the sampling stability of interval testing is so, continuous testing is almost meaningless, so in this environment, I did not conduct continuous sampling testing. There are many factors that cause this situation, such as nearby metal fences, metal windows, and wind. In summary, the size of the sensor board will affect the sensing distance to a large extent, and metal objects in the environment will reduce the sampling stability of the sensor.

Figure 3.10 shows another outdoor test site for the large plate test. There are no objects in vicinity, except the sewage manhole cover at 4 m from the sensors. The purpose of this test and test site is to eliminate the influence of nearby metal objects on the test results, but its underground structure may still interfere with the test results.

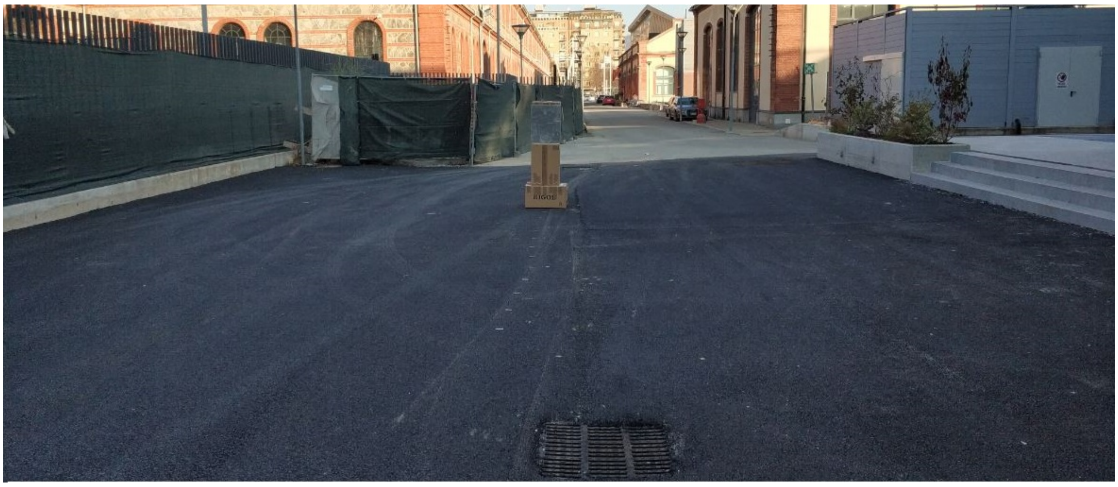


Figure 3.10: The second test environment of large plate

The test result shown in Figure 3.11 have many strong fluctuations, attenuating as the person is moving away from the sensor. They may be due to interference with sewage pipes or other structures under the test path.

Then I conducted some tests in the parking lot using a much larger object, a car (van), as shown in the Figure 3.12. This test environment avoids the interference of the previous test place. The surrounding area is open and presumably no underground structures on the test path (yet still there are manholes in vicinity). Its purpose is to test the performance of the large plate with a large surface area object (car) in a relatively good environment. This test is an interval test, the interval is 1 m, the maximum distance is 15 m, and each test point is sampled 30

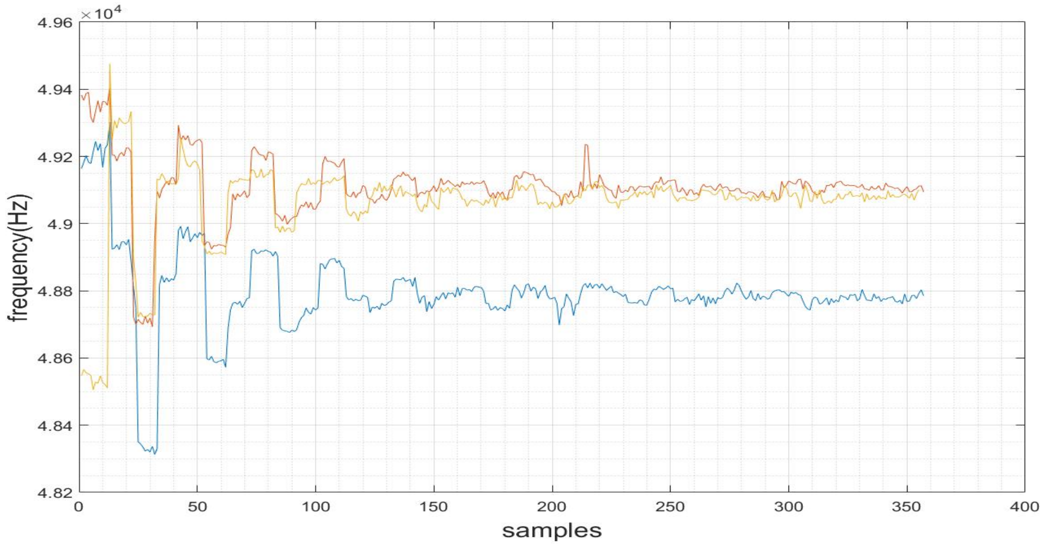


Figure 3.11: The result of the second test location interval test

times.

Figure 3.12: The environment of the parking lot

The test is divided into two groups. One group takes the rear of the car as a reference, with a surface area of 1.76 m × 1.74 m; the other group takes the side of the car as a reference, with a surface area of 4.16 m × 1.74 m. The result is shown in the Figure 3.13 and 3.15. We can see that the fluctuations with the distance to the object (car) are still present, but are lower than the variation of the sensor output with the sensing distance to the object. The reason may be the underground structure of the parking lot and environmental factors such as wind or the limitation of the test equipment itself.

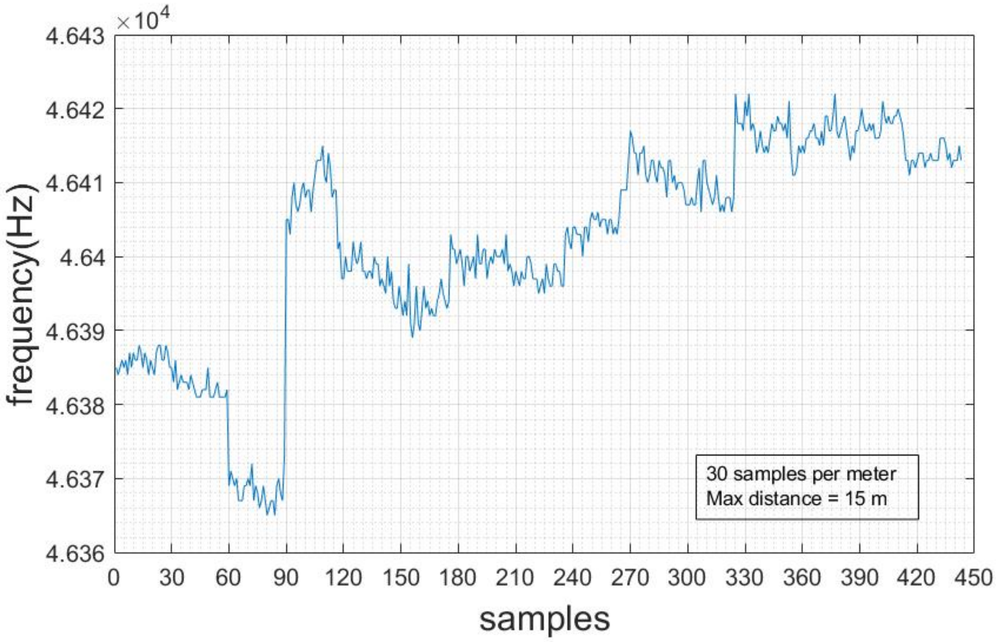


Figure 3.13: The sample of the rear of the car

To reduce the noise and increase the readability, I removed outlier samples. The main idea is to remove sudden and large jitters based on the data curve and finally retain 25 samples for each sampling point. The processed data is shown in Figure 3.14 and 3.16.

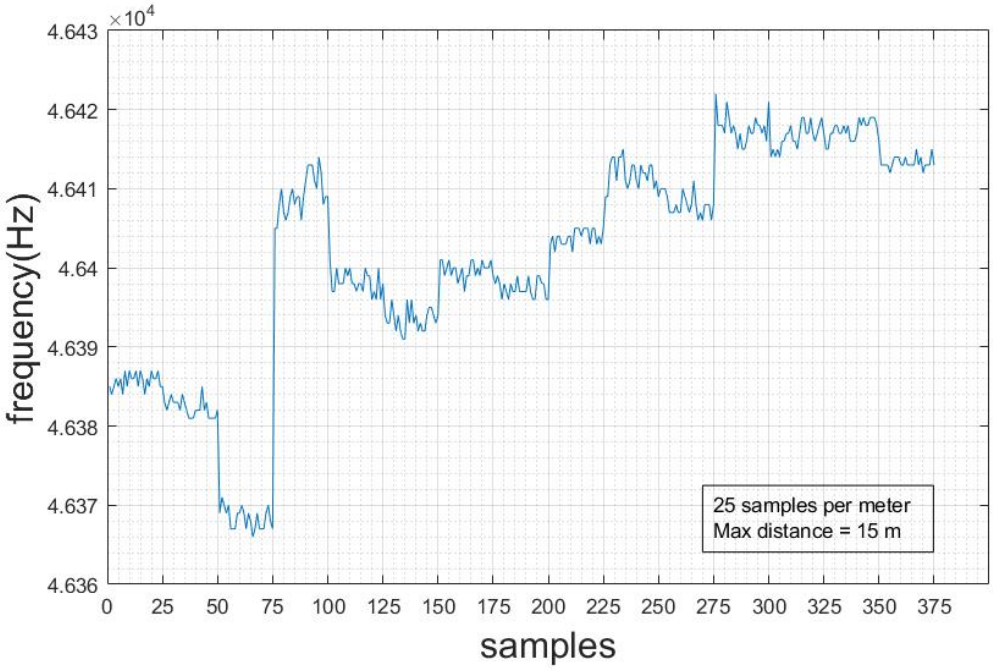


Figure 3.14: The processed samples of the rear of the car

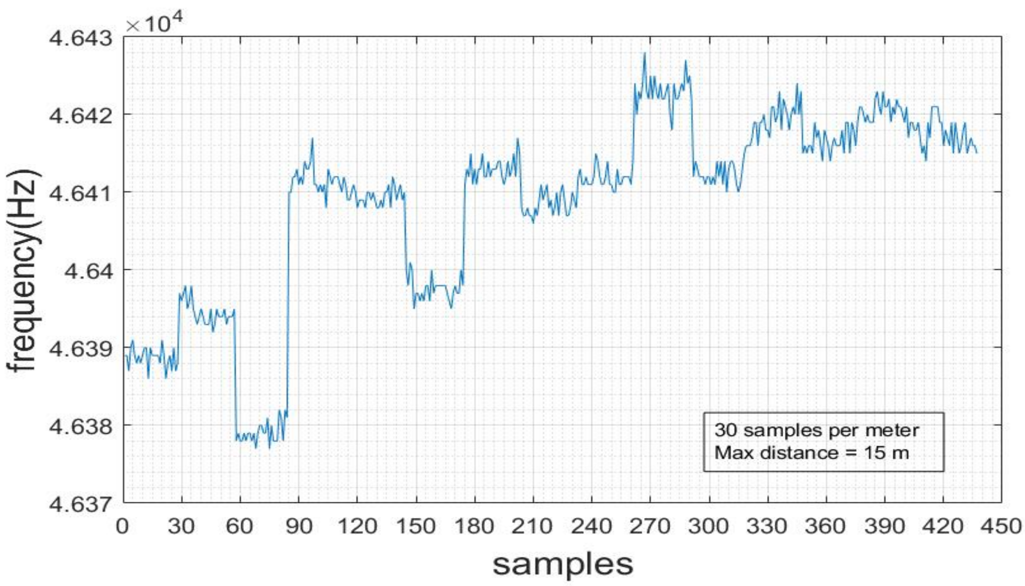


Figure 3.15: The sample of the side of the car

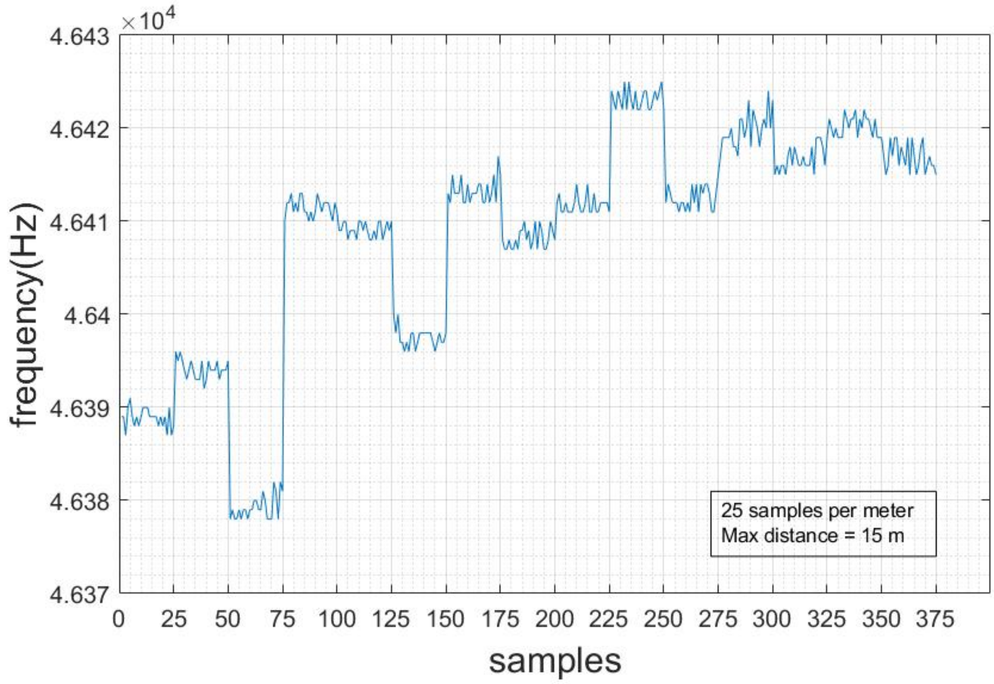


Figure 3.16: The processed samples of the side of the car

Comparing the sampling diagrams before and after processing, it is obvious that the processed data can more clearly reflect the sample collection status. At the same time, comparing Figure 3.8, Figure 3.14 and Figure 3.16, we can see that when the human body is used as a reference object, the sampling stability is the worst, and the sensitivity can reach about 10 m; when the rear of the car is used as a reference object, the sampling stability is greatly improved and the sample changes within 15 m are clear; while the results on the side of the car show that its sensitivity is not much different from that of the rear, but the sampling stability is better than that of the rear.

It can be concluded that the larger the surface area of the target test object, the stronger its anti-interference ability, and the higher the sampling stability and sensitivity. Moreover, the sensitivity of the sensor reaches more than 20 $\times$ its diagonal in all outdoor testing cases with large objects. It has a lower sensitivity to smaller objects, like a human body, because of the shielding effect of the extended patch of ground in close proximity between the sensor and the object of interest.

Chapter 4

Conclusion and future work

The experimental data shows that the size of the capacitive sensor plate plays a decisive role in the sensing distance. The larger the plate, the farther the sensing distance. At the same time, the larger the target surface area, the better the perception performance and the stronger the anti-interference ability. However, the performance of capacitive sensors is susceptible to environmental noise. Metal objects, human bodies and even underground sewage near the test site will reduce the sensing distance and sampling stability.

Finally, I put forward several improvement plans and suggestions for future research based on my experimental process.

First of all, control the experimental environment, where possible, minimize the interference in the environment (metal objects, etc.). At the same time, pay attention to the influence of invisible structures such as underground and walls on the test results.

Secondly, add a filter circuit on the basis of the circuit system of this project, which can increase the anti-interference ability of the system, thereby improving the overall performance.

Then in some applications, the distance between the capacitive sensor and the ground can be appropriately increased to reduce the coupling capacitance between the ground and the sensor.

The last, if possible a more superior microcontroller and sending/receiving module can be used. According to application requirements, weighing cost and performance, and selecting appropriate components may greatly improve the performance of capacitive sensors.

Appendix A

Frequency measurement code

```
#include <avr/io.h>
#include <stdlib.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#include "XBee.h"
#include "XBee_reception.h"

// Constants
//

#define F_CPU 16000000UL

// Serial port setup
//#define BAUDRATE 111111
#define BAUDRATE 9600
#define BAUD_PRESCALLER (((F_CPU / (BAUDRATE * 16UL))) - 1)

// Server protocol
#define SERVER_REQUEST_OFFSET 8
#define SERVER_REQUEST_START_MEASUREMENT 49
#define SERVER_ADDRESS 0xABCD

// Local variables
//
struct payload_s {
```

```
    uint32_t total_clocks_during_measurement;
    uint32_t total_input_periods;
    //uint32_t packet_counter;
} payload;

uint8_t *TX_payload = &payload;
uint32_t total_clocks_during_measurement = 0;
uint32_t total_input_periods = 0;
uint32_t pkt_counter_temp = 0;
//float input_signal_frequency = 0.0;

volatile unsigned char f_measurement_running = 0;
volatile unsigned char count_ovf_t0 = 0;
volatile unsigned char count_ovf_t1 = 0;
uint8_t measurement_window_expected_duration = 0;

volatile uint8_t remaineder_input_periods = 0;
volatile uint16_t remaineder_clocks = 0;

// Local function prototypes
//

static long intToStr(long x, char str[], int d);
static void ftoa(float n, char *res, int afterpoint);
static void reverse(char *str, int len);


#if !defined (__AVR_Ameasuring_windowega168__) || !defined (__AVR_Ameasuring_windowega48__) || !defined (__AVR_Ameasuring_windowega88__) || !defined (__AVR_Ameasuring_windowega328P__) || !(__AVR_Ameasuring_windowega1280__)
//#error "Unsupported architecture."
#endif

// if Timer/Counter0 overflow flag
ISR(TIMER0_OVF_vect)
{
    count_ovf_t0++;            // count number of Counter0 overflows the external signal
}

// if Timer/Counter0 overflow flag
ISR(TIMER1_OVF_vect)
{
    count_ovf_t1++;

    if (count_ovf_t1 >= measurement_window_expected_duration)
```

```
        PCMSK2 |= (1 << PCINT20); // Look for next input signal
    rising edge (end measurement).

}

/*
 * This interrupt is enabled ONLY when we are looking for input
 * signal edges, either:
 * - to start a new measurement
 * - to stop an ongoing measurement.
 *
 * NOTE: during a measurement, this interrupt is ALWAYS disabled.
 */
ISR(PCINT2_vect)                    // Port d, PCINt20
{
    char statepin = 0;

    // No raising edge of input signal
    statepin = PIND;
    if ((statepin & 0x10) != 0x10)
        return;

    // Raising edge of input signal.
    //

    // Start a new measurement.
    if (f_measurement_running == 0) {

        // Disable PCint until the end of measurement window.
        PCMSK2 &= ~(1 << PCINT20);

        // Reset counters and their overflow counters.
        TCNT1 = 0;
        TCNT0 = 0;
        count_ovf_t0 = 0;
        count_ovf_t1 = 0;

        // Start measuring the time.
        TCCR1B |= (0 << CS12) | (0 << CS11) | (1 << CS10); //no
    prescaling internal clock source

        // Connect input signal as Timer0 clock (rising edge of
    external signal, no prescaling)
        TCCR0B |= (1 << CS02) | (1 << CS01) | (1 << CS00);

        // Enable counter overflow interrupts.
        TIMSK1 |= (1 << TOIE1);
        TIMSK0 |= (1 << TOIE0);

```

```
        f_measurement_running = 1;
    }
    // End of the measurement.
    else {

        // Disable pin change interrupts.
        PCMSK2 &= ~(1 << PCINT20);

        // Stop counters.
        TCCR1B = TCCR1B & ~7;
        TCCR0B = TCCR0B & ~7;

        // Disable counter overflow interrupts.
        TIMSK1 &= ~(1 << TOIE1);
        TIMSK0 &= ~(1 << TOIE0);

        // Get remainders from counters.
        remaineder_clocks = TCNT1;
        remaineder_input_periods = TCNT0;

        f_measurement_running = 0;
    }
}

int
main(void)
{
    //uint8_t *packet_from_server;

    cli();

    // Measurement window in terms of 16 bit counter overflows (
    TIMER1)
    measurement_window_expected_duration = 20;

    TCCR0A = 0;                    // reset timer/counter0 control
    register A
    TCCR0B = 0;                    // reset timer/counter0 control
    register A
    TCNT0 = 0;                     // counter value = 0

    // timer1 setup / is used for frequency measurement gate time
    generation with 16MHZ/no prescaling
    TCCR1A = 0;
    TCCR1B = 0;

    f_measurement_running = 0;

    //pinchange interrupt
```

```
    DDRD &= ~(1 << DDD4);       // Set pd4/pcint20 as input
    PORTD |= (1 << PORTD4); /* Activate PULL UP resistor */
    //pin PD4 is now an input with pull-up enabled

    PCICR |= 0b00000100;        // turn on port d interrupts (PCINT
    [23:16])

    XBeeUSART_init();

    sei();

    while (1) {

        //correct packet received
        //pkt_counter_temp = packet_from_server[4] << 24 |
    packet_from_server[3] << 16 | packet_from_server[2] << 8 |
    packet_from_server[1];



        // Make a new measurement.
        //
        // Input signal rising edge will start the new measurement.
        //

        f_measurement_running = 0;
        PCMSK2 |= (1 << PCINT20);
        // Wait for the actual START of the new measurement.
        //
        while (f_measurement_running == 0) ;

        // Wait for the actual END of the new measurement.
        //
        while (f_measurement_running == 1) ;
        // Measurement completed.
        //
        // Calculate input signal frequency.
        //input_signal_frequency = (float)total_input_periods / ((
    float)F_CPU * (float)total_clocks_during_measurement);

        // Total time for all input signal periods.
        total_clocks_during_measurement = count_ovf_t1 * 65536 +
    remaineder_clocks;

        // Full periods of input signal.
        total_input_periods = count_ovf_t0 * 256 +
    remaineder_input_periods;

```

```
        // f555 = 1 / ((total_clocks_during_measurement / 16 MHz) / total_input_periods)
        //
        // f555 = total_input_periods * 16 MHz / total_clocks_during_measurement

        remaineder_clocks = 0;
        remaineder_input_periods = 0;

        /////// This section is for XBee communication ////////
        // TODO: check for endianess ATmega vs server.
        payload.total_clocks_during_measurement = total_clocks_during_measurement;
        payload.total_input_periods = total_input_periods;
        //payload.packet_counter = pkt_counter_temp;

        _delay_ms(150);
        //add new IN to buffer
        XBee_TX_Request(SERVER_ADDRESS, TX_payload, sizeof(struct payload_s));
        //packet sent

        /*
        //used for debugging
        _delay_ms(200);

        DDRB |= (1<<DDB3); //setting port direction to output
        PORTB |= (1<<PORTB3); //setting output high RED

        DDRB |= (1<<DDB4); //setting port direction to output
        PORTB |= (1<<PORTB4); //setting output high YELLOW

        DDRB |= (1<<DDB5); //setting port direction to output GREEN
        PORTB |= (1<<PORTB5); //setting output high



        PORTB &= ~(1<<PORTB3); //setting output low
        PORTB &= ~(1<<PORTB4); //setting output low
        PORTB &= ~(1<<PORTB5); //setting output low
        */
    }
}

static void
reverse(char *str, int len)
{
    int i = 0, j = len - 1, temp;
    while (i < j) {
```

```
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
        i++;
        j--;
    }
}

// Converts a given integer x to string str[].  d is the number
// of digits required in output. If d is more than the number
// of digits in x, then 0s are added at the beginning.
static long
intToStr(long x, char str[], int d)
{
    int i = 0;
    while (x) {
        str[i++] = (x % 10) + '0';
        x = x / 10;
    }

    // If number of digits required is more, then
    // add 0s at the beginning
    while (i < d)
        str[i++] = '0';

    reverse(str, i);
    str[i] = '\0';
    return i;
}

// Converts a floating point number to string.
static void
ftoa(float n, char *res, int afterpoint)
{
    // Extract integer part
    long ipart = (long)n;

    // Extract floating part
    float fpart = n - (float)ipart;

    // convert integer part to string
    long i = intToStr(ipart, res, 0);

    // check for display option after point
    if (afterpoint != 0) {
        res[i] = '.';           // add dot

        // Get the value of fraction part upto given no.
        // of points after dot. The third parameter is needed
```

```
        // to handle cases like 233.007
        fpart = fpart * pow(10, afterpoint);

        intToStr((long)fpart, res + i + 1, afterpoint);
    }
}
```

Appendix B

Receiver module code

```
#define highWord(w) ((w) >> 16)
#define lowWord(w) ((w) & 0xffff)
#define makeLong(hi, low) (((long) hi) << 16 | (low))
#define Frq_CPU  16000000UL

#include <XBee.h>

XBee xbee = XBee();
XBeeResponse response = XBeeResponse();
//response objects for responses we expect to handle
Rx16Response rx16 = Rx16Response();


uint8_t total_clocks_during_measurement[4];
uint8_t total_input_periods[4];
uint16_t add;
uint8_t check_error;

word r_hiword_tcm, r_loword_tcm, r_hiword_tip, r_loword_tip;
unsigned long r_full_tcm, r_full_tip;
double frq;
unsigned long Int_frq;

String add_str;
String address_frq;


void setup() {

  // start serial
```

```
  Serial.begin(9600);
  xbee.setSerial(Serial);


}

// continuously reads packets, looking for RX16 or RX64
void loop() {

    xbee.readPacket();

    if (xbee.getResponse().isAvailable()) {
      // got something

        if (xbee.getResponse().getApiId() == RX_16_RESPONSE) {
                xbee.getResponse().getRx16Response(rx16);
                check_error = rx16.getErrorCode();
                if (check_error == NO_ERROR) {

                add = rx16.getRemoteAddress16();
                String add_str = String(add, HEX);


                total_clocks_during_measurement[0] = rx16.getData(0);
                total_clocks_during_measurement[1] = rx16.getData(1);
                total_clocks_during_measurement[2] = rx16.getData(2);
                total_clocks_during_measurement[3] = rx16.getData(3);
                    total_input_periods[0] = rx16.getData(4);
                    total_input_periods[1] = rx16.getData(5);
                    total_input_periods[2] = rx16.getData(6);
                    total_input_periods[3] = rx16.getData(7);


                r_hiword_tcm =word(total_clocks_during_measurement[3], total_clocks_during_measurement[2]);

                r_loword_tcm = word(total_clocks_during_measurement[1], total_clocks_during_measurement[0]);

                r_full_tcm = makeLong(r_hiword_tcm, r_loword_tcm);

                r_hiword_tip = word(total_input_periods[3], total_input_periods[2]);

                r_loword_tip = word(total_input_periods[1], total_input_periods[0]);

                r_full_tip = makeLong(r_hiword_tip, r_loword_tip);

```

```

                    frq =(double)r_full_tip * ((double)Frq_CPU / (double)
    r_full_tcm ) ;
                    Int_frq = (long)frq;

                        //String str_frq = String(r_full_frq);
                        String str_frq = String(Int_frq);
                        //String out = String(rec_data[1], HEX);

                        if (add_str == "aaaa"){
                          address_frq = String("A"+ str_frq);
                        }

                         if (add_str == "aaab"){
                          address_frq = String("B"+ str_frq);
                        }

                        if (add_str == "aaac"){
                          address_frq = String("C"+ str_frq);
                        }

                        if (add_str == "aaad"){
                          address_frq = String("D"+ str_frq);
                        }

                        Serial.println(address_frq);
                        delay(1000);

                    }
            }
        }
    }
```

Appendix C

Matlab code

```
clear;
clc;

%open serial port
s = serialport ( "COM11" , 9600);

first=0;
result=[];
j=1;

%set start and end time
tStart = cputime;
tEnd=0;

while(tEnd<120)
    data=read(s,1,"char");

    % the format of transmitted data is sameting like "
    xxxBxxxxxBxxxxxxBxxxx..."
    if data=='B'        % transmittor label,the data from transmittor
     'B'
        first=first+1;  %  the flag to record the times 'B' comes
    end

    if first==2         % the second time B comes means the second
    frequency value comes, save result and set flag
        first=1;
        if result(1)=='B'
            result(1)=[];
            frq_B(j)=str2num(result);
```

```
            end
            result=[];
            j=j+1;
    end

    if first==1        % after the first B comes,record next serveral data which is the frequency value.
        result=[result data];
    end

    tEnd = cputime-tStart;
end
save('frequency_B', 'frq_B');

for i=1:floor(length(frq_B)/10)
    average=0;
    for j=1:10
        average=average+frq_B((i-1)*10+j);
    end
    frq_avg_B(i)= round(average/10);
end
save('frequency_B_average', 'frq_avg_B');

x=1:length(frq_B);
x_avg=0.5:0.5:length(frq_avg_B)/2;

plot(x,frq_B);
xlabel('Samples')
ylabel('Frequency(Hz)')
saveas(gcf,'sample_frq.fig')

plot(x_avg,frq_avg_B);
xlabel('Distance(m)')
ylabel('Frequency(Hz)')
saveas(gcf,'distance_frq.fig')
```

Bibliography

[1] Tobias Grosse-Puppendahl, Christian Holz, Gabe Cohn, Raphael Wimmer, Oskar Bechtold, Steve Hodges, Matthew S Reynolds, and Joshua R Smith. «Finding common ground: A survey of capacitive sensing in human-computer interaction». In: *Proceedings of the 2017 CHI conference on human factors in computing systems*. 2017, pp. 3293–3315 (cit. on pp. 1–3).

[2] Javed Iqbal, Mihai Teodor Lazarescu, Osama Bin Tariq, and Luciano Lavagno. «Long range, high sensitivity, low noise capacitive sensor for tagless indoor human localization». In: *2017 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*. IEEE. 2017, pp. 189–194 (cit. on p. 2).

[3] Alireza Ramezani Akhmareh, Mihai Teodor Lazarescu, Osama Bin Tariq, and Luciano Lavagno. «A tagless indoor localization system based on capacitive sensing technology». In: *Sensors* 16.9 (2016), p. 1448 (cit. on pp. 4, 6).

[4] ElectronicTutorials. *555 Oscillator Tutorial*. URL: `https://https://www.electronics-tutorials.ws/waveforms/555_oscillator.html` (cit. on pp. 5, 8).

[5] Pellegrino D'Angelillo. «Characterisation of capacitive front-ends for indoor person localization». PhD thesis. Politecnico di Torino, 2020 (cit. on p. 6).

[6] Chenjie Cao. «Capacitive Sensor Front-end Using Carrier Demodulation». PhD thesis. Politecnico di Torino, 2020 (cit. on p. 6).

[7] Pellegrino D'Angelillo. «Characterisation of capacitive front-ends for indoor person localization». PhD thesis. Politecnico di Torino, 2020 (cit. on p. 6).

[8] Mihai Teodor Lazarescu, Luciano Lavagno, and Fotso Hondjie Christian Ulrich. «Front-end for long range capacitive sensor». In: () (cit. on p. 6).

[9] Atmel. *ATmega328P datasheet*. URL: `http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf` (cit. on p. 7).

[10] DIGI. *Digi Xbee*. URL: `https://www.digi.com/xbee` (cit. on pp. 7, 9).

[11] ARDUINO. *ARDUINO WIRELESS SD SHIELD*. URL: `https://store.arduino.cc/arduino-wireless-sd-shield` (cit. on pp. 7, 11).

[12] Kuku Tena Nigatu. «tagless long/distance capacitive sensors for indoor human localization». PhD thesis. Politecnico di Torino, 2018 (cit. on pp. 7, 11, 12).