IT 18 063
Examensarbete 30 hp
November 2018

Dancing with Theremins

Rahmanu Hermawan

Abstract

Dancing with Theremins

Rahmanu Hermawan

This project is building a wireless sensor network for a room-scale Theremin instrument using ZigBee protocol. The Theremin is an instrument that is played entirely without touching it. The instrument based on electromagnetic fields, which are being "disturbed" by the limbs of the player. Theremin has two antennas, vertical antenna for controlling the pitch and horizontal antenna to control the volume. Unfortunately, the playing of Theremin is limited to the area around the antenna.

This case, however, also means that it will be necessary to redesign the theremin in some aspects. We will try out a couple of ultrasound sensors, which will be used to detect the movements of the player's bodies. We also utilised those ultrasound sensors along with microcontroller, FreeRTOS, and ZigBee devices to create a wireless sensor network.

In the end, after did some evaluations and concluded that to replace the electromagnetic field, we can use the ultrasound sensors. Besides, ultrasound sensors have a decent accuracy to measure an object distance. Our room-scale Theremin (we call it Thereminz) make music using MIDI by converting the detected distance into MIDI note. We configured Thereminz to create MIDI note-on, note-off, and pitch bend to particular movements.

Handledare: Lars Oestreicher
Ämnesgranskare: Pontus Ekberg
Examinator: Justin Pearson
IT 18 063
Tryckt av: Reprocentralen ITC

Contents

List of Figures

1. Introduction

1.1 Background

The Theremin is an instrument that is played entirely without touching it [13], which makes it an almost magic device. It has been intriguing people with its sound and way of playing since it was invented in 1914 by Leon Theremin. The instrument based on electromagnetic fields, which are being "disturbed" by the limbs of the player. Because of electromagnetic usage, unfortunately also means that the areas reach a limited space around the antenna. Making a super-theremin is not just a case of making it bigger and with a more powerful electromagnetic field.

Theremin, as mentioned, is played without touching or even handling the instrument physically. Division of Visual Information and Interaction in Uppsala University, under The MUMIN project, has utilised Theremin to help children with weak muscles to play music even without full control over the arms. On the downside, the original theremin is very difficult to play in a tuned fashion due to its sensitivity. Also, a minimal movement such as finger movement will change the pitch. There is currently a new version of the Theremin, called a Theremini. It has some features such as pitch control, which allows the Theremini's player to play a perfect pitch. Some people think this is a clear example of cheating the original idea, but it does play a significant role in the MUMIN-project.

This thesis project will pick some of the best parts of the Theremin(i) and modify it to produce an instrument that will extend it into a choreographic multi-user instrument. The idea is to develop a "dance floor" where the music is created interactively by the "dancers" through the dance. We called our developed Theremin as Thereminz.

Under MUMIN-project, this project is helping children with weak muscles to play music with minimal movement. Room-Theremin is one solution to help them. For illustration, a child on a wheelchair will be able to play music by moving forward or backwards while facing one of Thereminz' antenna. The Thereminz, hopefully, can also help a dance community to create an aesthetic dance work using Thereminz in the future.

This case, however, also means that it will be necessary to redesign the theremin in some aspects. The electromagnetic field does not have the required reach for the setup. Instead, it will be tried out a couple of ultrasound sensors, which will be used to detect the movements of the bodies. This means that the function will be slightly different from the original theremin, but has the advantage that is possible to get better control over the generated music. In the end, we hope Thereminz can be a digital musical instrument alternative for everyone.

1.2 Research question

As a preliminary study, there are some research questions which can be answered at the end of this project. The questions are:

1. What is the possibility of building a room-Theremin?
2. Can ultrasound sensors be used to build a room-Theremin?
3. How is the room-Theremin's performance compared with the original Theremin?

1.3 Delimitation

We state some limitations of this project. This work will focus on building the hardware and software implementation for the Thereminz and also do some testings. As we only do "proof of concept" research, we will not do optimisation on the system. The user interface in the computer which is a customised synthesiser, will not be covered by this work. Hence, in this project, we used an available open source synthesiser to generate the music.

In the hardware part, we used microcontrollers, wireless communication modules, and ultrasound sensors to detect the movement. We also developed the software in Arduino C-language under FreeRTOS environment. We test our system with some methods. They are distance measurement accuracy, distance to musical-expression capability, and comparing the Thereminz and Theremin performance. Some feedback from testing parts will be used for the future development of the system.

1.4 Thereminz overview

As mentioned earlier, to make a dance-floor, a room-scale theremin is needed. In this project, we call the room-scale theremin as Thereminz. Four of antennae built a Thereminz system which detects the distance and movement of the participants. An additional antenna which detects the acceleration is also utilised to produce drum sounds. All of the antennas send the message to the base station. In the base station, the microcontroller translates the distance-message into MIDI message to the computer. A synthesiser in the computer synthesises MIDI messages to some interesting sounds.

Figure 1.1 illustrates an overview of the Thereminz. Generally, the Thereminz is a wireless sensor network with the star topology. Each node (K, L, M, N,

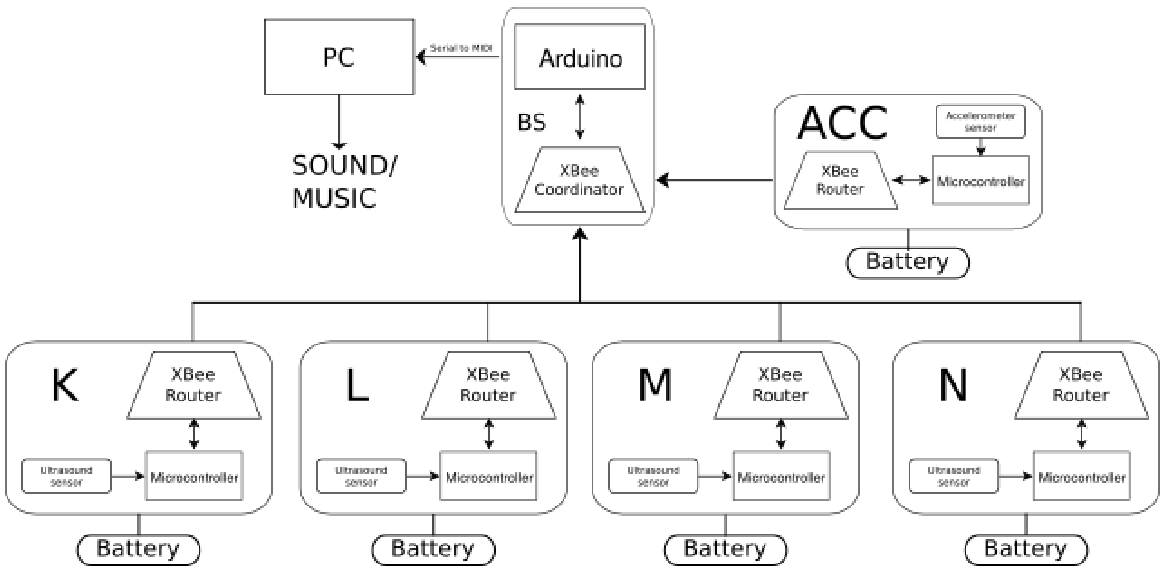


Figure 1.1. Diagram of the whole "Dancing with Theremins" system

and ACC) communicates via radio frequency with the base station (BS) using ZigBee protocol. Inside a node, there are three main hardware components: a microcontroller, ultrasound sensors, and an XBee router. Via a power supply circuit, a 9V battery powers all of them. In this project, we call these nodes antennas.

The base station consists of two main hardware components: an Arduino Uno board and an XBee coordinator. The base station is connected to a PC via USB to produce sound. A synthesiser does the sound production in the PC through the speakers.

Each antenna (except the acceleration antenna ACC) in the Thereminz utilises ultrasound sensors to replace the electromagnetic field of the original Theremin. Many people cannot perform the original Theremin because of the interference explained by Martin et al. [9]. They explain that a thereminist, theremin player, usually gets interference by another player when performing in a stage or small room. On the other hand, Thereminz encourages people to play together.

In term of detection range, compared to original Theremin, Thereminz has more extended detection range. Thereminz can detect a movement up to four meters. By using the potentiometer on each antenna, the user can variate the detection range regarding the room size. The following subsections (ultrasound sensor, FreeRTOS, microcontroller, ZigBee, MIDI) will discuss the respective main components.

1.4.1 Ultrasound sensor

An ultrasound sensor is a sensor which utilises ultrasonic wave to detect an object. It works in ways similar to radar and other sensors which is utilising Doppler theory. In an ultrasound sensor, two transducers transmit and receives the ultrasound wave. Once the ultrasound wave is transmitted, it will hit an object and reflects toward the receiving transducer which converts the ultrasound wave into an electrical signal. Hence, the distance between the object and sensor can be evaluated using Equation 1.1. Where d is the distance, v is speed of sound and t is traveling time duration of ultrasound wave from transmitting transducer to receiving transducer [18].

$$d = \frac{v * t}{2} \tag{1.1}$$

In this project, 16 HC-SR04 ultrasound sensors are used to detect the distance. The specifications of HC-SR04 are listed below [21].

- Power supply: +5 VDC
- Working current: 15 mA
- Ranging distance: 2 - 400 cm
- Resolution 0.3 cm
- Measuring angle: 30 degree
- Trigger input pulse width: 10 uS
- Dimension: 45mm x 20mm x 15mm
- Weight: approx. 10g

Figure 1.2 shows the HC-SR04 ultrasound sensor. This sensor has four pins, VCC, GND, TRIG, and ECHO. All of them have a different function. VCC and GND are for power connection. They need to be connected to +5 VDC and ground respectively. TRIG pin is responsible for sending the ultrasound wave. This pin should be connected to a digital pin in microcontroller and set to HIGH for 10uS. At this point, TRIG pin will send eight cycles of ultrasonic burst at 40 KHz. After the ultrasound wave hits an object, it will be reflected and received by ECHO pin. Using this pin, we can do distance measurement [21].

To calculate the distance to the object, the duration of ECHO pin stays HIGH can be tracked. The ultrasound burst travelling time is the time ECHO pin stays HIGH. Using Equation 1.1, where $v = 343m/s$ (in standard temperature and pressure), the distance can be calculated.

The expected delay, which is the maximum travelling time, for the measurement if the sensor gets maximum distance will be explained as follows. Let us assume the sensor hits an object in 400 cm. Therefore, the travelling time is

$$t = \frac{d}{v * 2} = \frac{0.4}{343 * 2} = 2.33ms \tag{1.2}$$

Because of the longest travelling time is 2.33 ms, the measurement delay must not faster than 2.33 ms otherwise the calculation will be incorrect.

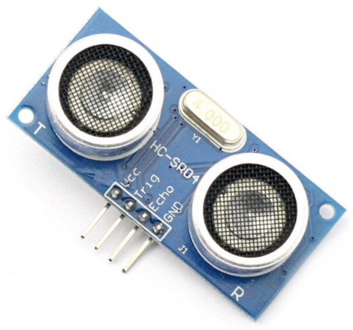


Figure 1.2. HC-SR04 ultrasound sensor has a transmitter (T) and a receiver (R) [3]

1.4.2 Microcontroller

The microcontroller as the central processing unit controls everything from blinking LEDs until wireless communication through the ZigBee network. Generally, any microcontroller is suitable for this project as long as it can cover all of the needs such as digital pins, analogue pins, serial communication hardware, and other peripherals. In this project, we use ATMega328P.

ATMega328P is one of 8-bit AVR microcontroller produced by Atmel. It is one of RISC-based which has 32 kB flash memory, one kB EEPROM, 2kB SRAM, 23 GPIO, three timers with compare modes, interrupt pins, SPI serial port, serial programmable USART, and 6-channel 10-bit ADC [4].

Because of its popularity, there are many references about it. Some of them are Instructable.com and hackster.io, both of them may DIY-project which use a microcontroller like ATMega328P or ATMega168 as the main component. The most popular microcontroller development board, Arduino, also uses ATMega328P on their Arduino Uno product.

A project which uses an ATMega328P or any microcontroller is replicable and reproducible. Once the schematic and the program is well documented, the project can easily be replicated by other interested people. Arduino is an open source hardware device which is well maintained and documented by the company. Also, Arduino publishes their complete schematic and provides a special IDE to program the microcontroller. Because of well documented, there is much Arduino-like board produced by some small companies and even for private usage.

1.4.3 FreeRTOS

FreeRTOS is a professional-grade embedded real-time operating system (RTOS) that is free to use and open source [8]. Real Time Engineers Ltd is the developer, maintainer, and owner of FreeRTOS. This RTOS is suitable for embedded systems using a small microprocessor or microcontroller like ATMega328P. In addition, it has compatibility with a large variety of microcontrollers or microprocessor.

FreeRTOS support both soft and hard real-time applications. Soft real-time system requirements are the system which states a time deadline, but breaking the deadline will not make the system fail. For example is the sound system in a computer or a blinking LED in a portable MP3 player. If the system misses some deadlines, the system still works fine. [10]

On the other hand, hard real-time system requirements are the systems which state a time deadline, and it must not miss the deadline. Once one deadline is missed, the system probably will behave abnormally. For example is the airbag system in a car. If the airbag is not active at the right time, the driver and perhaps the passenger will eventually have a severe injury. [10]

FreeRTOS organises the application as a collection of independent threads of execution. Because most of the microcontroller or small microprocessor has only one core, only one thread can be executed by the processor at one time. In FreeRTOS, a thread is called as a task. The kernel decides the thread/task execution by examining the assigned priority of a task. However the tasks can have some different priorities, the priority assignment is not easy. For example, if the tasks contain a mix of hard and soft real-time system requirements. [10]

As one of real-time kernel scheduler, FreeRTOS also offers some benefits [2]. They are modularity, maintainability, easier testing, code reuse, improved efficiency, idle time, power management, flexible interrupt handling, and mixed processing requirements.

1.4.4 ZigBee

ZigBee is a protocol that leverages IEEE 802.15.4 protocol. IEEE 802.15.4 is a technical standard which specify the low-rate wireless personal area network (LR-WPAN). It defines the physical layer (PHY) and media access control (MAC) for LR-WPAN.

The IEEE 802.15.4 physical layer has a responsibility to manage data transmission and reception using a certain radio channel. There are three operational frequency bands under IEEE 802.15.4 standard, 2.4 GHz, 915 MHz, and 868 MHz. There are 16 channels between 2.4 and 2.4835 GHz, ten channels between 906 and 928 MHz, and one channel between 868 and 868.6 MHz [27].

The IEEE 802.15.4 physical layer is responsible of the following tasks [27].

- Selection of channel frequency.
- Clear channel assessment.
- Radio transceiver activation and deactivation.
- Energy detection.
- Link quality indication (LQI)

There are two operational modes in IEEE 802.15.4 medium access control, non-beacon-enabled and beacon-enabled mode. In a non-beacon-enabled mode, there are no beacon or superframes. An unspotted carrier sense multiple access with collision avoidance (CSMA/CA) mechanism rules the medium access. On the other hand, in beacon-enabled mode, medium access is ruled by slotted CSMA/CA. In addition, beacon-enabled mode also enables guaranteed time slots (GTS) for nodes that are requiring guaranteed bandwidth.

As mentioned earlier, IEEE 802.15.4 is the basis of ZigBee. According to [15], there are two identifications of ZigBee, FFD (Full Function Device) and RFD (Reduced Function Device). An FFD device is usually powered by AC electricity thus it can always actively listening on the network. On the other hand, an RFD device is usually powered by battery or other portable power sources. Thus it makes an RFD device only capable of doing limited tasks because it is not listening to all the time.

ZigBee adopts the IEEE 802.15.4 both FFD and RFD concepts and creates three ZigBee protocol devices. They are ZigBee coordinator which is represented as an FFD device, ZigBee router which is represented as an FFD device, and ZigBee end device which is represented by an FFD and RFD device. [15]

ZigBee coordinator is acting as the sink or the server of the network. It coordinates the other nodes who join the network. It also routes messages between nodes in the network. ZigBee router is acting the routing device in the network. If needed, It can expand the network range and also manage if more nodes want to join the network. ZigBee end device is acting as the sensor interface or executes control function. [15]

In term of network topologies, three common ZigBee networks can be set up as the star, cluster tree, and mesh. Figure 3.3 shows the network topologies. First is the star network. This network configuration involves a ZigBee coordinator and some routers or end devices. A star network the simplest network topology which can be created using ZigBee. This project is implementing star network topology

Second is the cluster tree network. This network configuration involves ZigBee coordinator, ZigBee router, and ZigBee end device. This network topology is a little bit more complicated than the star network. It allows ZigBee routers act as the guard that allows ZigBee device to join the network via it. Each ZigBee end device cannot communicate directly without ZigBee coordinator or ZigBee router. It is possible to create a cluster tree topology by connected some star networks.

Third is the mesh network topology. It is the most complex network because it allows multi-hop communication between the nodes and also between nodes and sink.. In addition, a ZigBee end device may communicate directly with each other as long as it set up to be FFD device.

Concerning configuration mode, ZigBee has two modes. They are transparent mode (AT) and Application Programming Interface (API) mode. In transparent mode, a ZigBee coordinator and ZigBee router can communicate directly without encryption. The router receives the information accurately as it transmitted. In addition, this mode is designed for more human interaction. That is why AT mode is not using any encryption. [16]

On the other hand, the API mode is not human-friendly. The researcher designs it so the nodes in the network can communicate with each other efficiently [16].However, when API mode is used in the application, we as a human need to decrypt the whole received message to get the exact transmitted message.

ZigBee is not a device

The bottom-line about ZigBee, ZigBee is not a device. ZigBee can be implemented manually on to a project which uses a microcontroller or any microprocessor [23]. However, to simplify ZigBee implementation, XBee can be utilised. XBee is one of wireless communication device which has ZigBee stack specification. It is developed by DIGI [1]. There also some xbee libraries which can be used to simplify the implementation.

XBee is a ZigBee device

In this project, we used XBee-pro series 2 which has the specification as follows:

- Long range data transmission, up to 60m
- Transmission power of 10mW
- Source and destination addressing
- Unicast and broadcast communications
- Low power. TX and RX peak current in is 250 mA and 55mA respectively.
- 3.3 VDC operational voltage.
- Supports AT and API mode
- Comes with free X-CTU software which can be used to configure XBee.

The datasheet of XBee Pro Series 2 gives information that XBee has bandwidth 31 250 Byte per second. Therefore we can expect that in one second, the maximum data which can be sent is 31.250 kB. Because each antenna in Thereminz sends 1 Byte distance data to base station, we can assume that the 1 Byte data can be sent in 32 microsecond.

There is also a scanning mechanism in XBee which scans all of the involved nodes in the network at first initialisation. As a default, the duration of the scanning time is 2.95 seconds. When a collision happens, the exponential algorithm will perform a random time delay to retransmit the data. The duration of the random time is between 15.36 ms and 251.65824 seconds at 250 kb/s. Let us assume that when the collision happens, the random time is 15.36 ms. Adding the sending time 32 microseconds and random time 15.36 ms, we can expect that the duration of transmitting one-music-package is 15.392 ms.

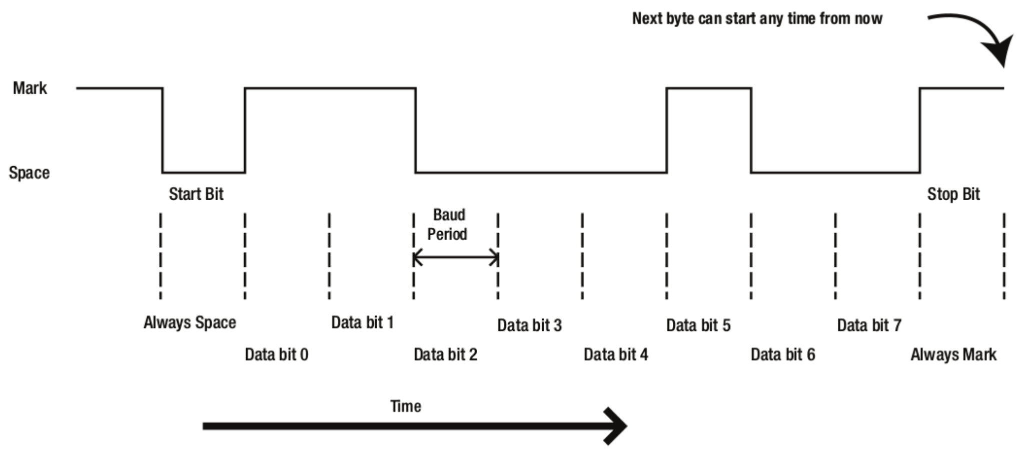


Figure 1.3. The timing diagram of serial data in serial communication [12]

1.4.5 MIDI

Musical Instrument Digital interface or widely called as MIDI is an interface which can standardise any electronic synthesiser that had been developed by some manufacturers [5]. Based on [12], MIDI is not only for controlling musical notes but also capable of controlling some lighting effects or other conventional musical applications.

In technical terms, MIDI uses an asynchronous serial interface to communicate with the computer with the speed of communication of 31 250 symbols per second. The rate seems very odd. However, the Universal Asynchronous Receiver Transmitter (UART) chip need a clock oscillator that is 16 times the required speed. Then for the solution, in the old days, the engineer used 1 MHz crystal to build into an oscillator because it was easy and cheap [14].

The following example is explaining why 16 times required speed be needed. By using one 1 MHz crystal feed into two circuits and then a UART, we will automatically get 31 250 KHz. Let us assume each branch oscillates in 500 000 KHz. Then by divide it in 16 we can easily get a oscillator which oscillates in frequency of 31 250 KHz. [14].

The asynchronous serial data is the data which is sent one bit at a time. There are many variations of it. However, the data has a start bit as the start signal, data bits, and end bit as the stop signal [14]. Therefore, in the serial communication, the computer will acknowledge the data by captured it after the start bit is acknowledged. An example of serial communication in daily life is entering a letter to the computer from the keyboard. Figure 1.3 shows how does serial data behave.

Concerning MIDI message, it is closely related to serial communication. MIDI communication is built on serial communication. Some common MIDI messages are note-on, note-off, control change, and some many more messages. In this project, we use note-on, note-off, and control change.

Each MIDI message affiliates with a MIDI channel. There are 16 channels from 0 to 15 which can be used in MIDI communication [14]. In a MIDI

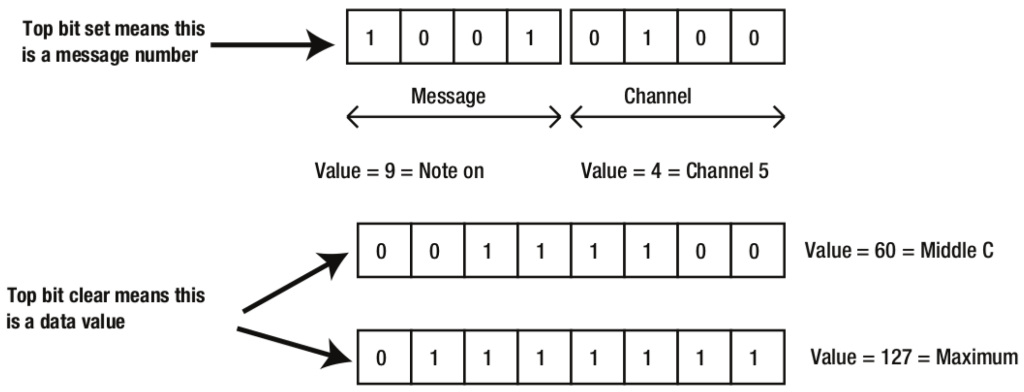


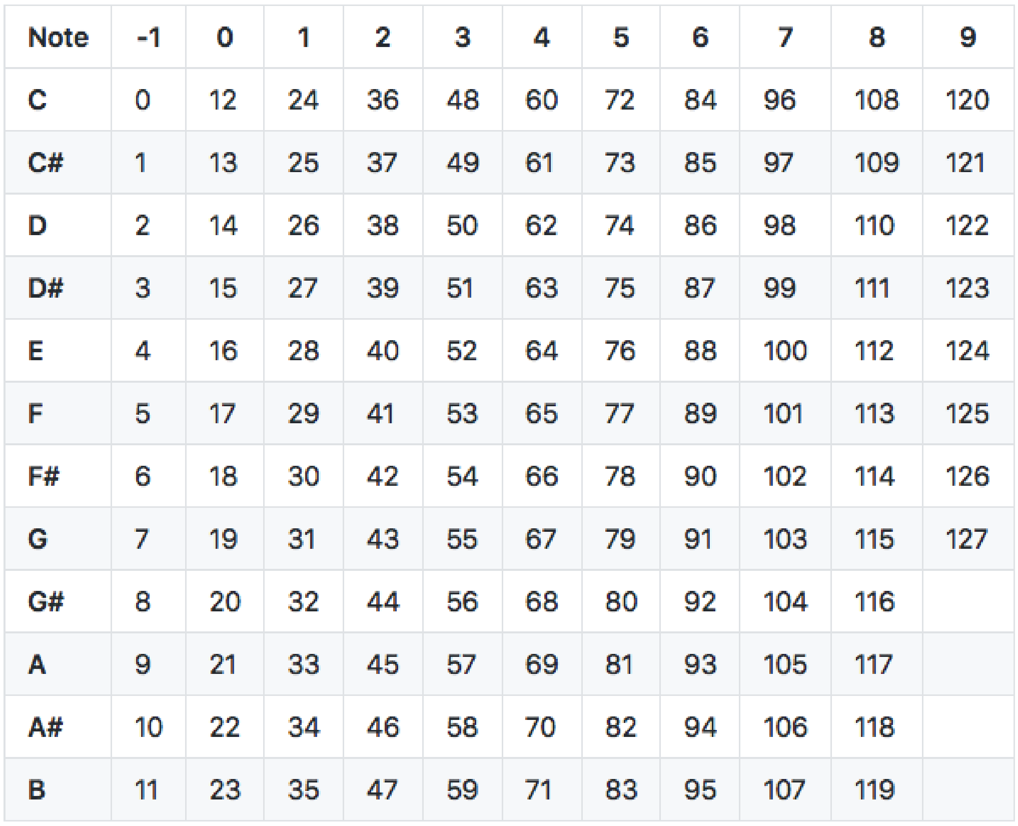
Figure 1.4. The structure of note-on message in MIDI [12]

channel, the user can send some MIDI messages such as note on, note off, and a control change sequentially. There is no limitation of how many MIDI messages can be sent, but there is a limit of how many messages can be sent per second due to communication speed limitation.

Moving on now to consider how does the MIDI message work. We take an example of the note-on message. Figure 1.4 shows the structure of the note-on message. Entirely, there are 3 bytes in the note-on message. The first byte specifies the command, the second byte specifies the note, and the third byte specifies the velocity. The first byte (command byte) embeds the channel information. A value between 0 to 127 must be passed on the second and third byte to play a specific note. Those values are representing 11 note octaves and the velocity respectively. Also, most of "standard" velocity value is set to 100. Figure 1.5 shows the correlation between MIDI value and note.

1.5 Structure of the report

This report is structured as follows. Chapter 2 will provide some of the related works. Chapter 3 will discuss the design of antennas' hardware and software. Chapter 4 will discuss the design of the base station's hardware and software. All of the works will be evaluated in Chapter 5. In the end, Chapter 6 will discuss the conclusion of the work.



Note	-1	0	1	2	3	4	5	6	7	8	9
C	0	12	24	36	48	60	72	84	96	108	120
C#	1	13	25	37	49	61	73	85	97	109	121
D	2	14	26	38	50	62	74	86	98	110	122
D#	3	15	27	39	51	63	75	87	99	111	123
E	4	16	28	40	52	64	76	88	100	112	124
F	5	17	29	41	53	65	77	89	101	113	125
F#	6	18	30	42	54	66	78	90	102	114	126
G	7	19	31	43	55	67	79	91	103	115	127
G#	8	20	32	44	56	68	80	92	104	116	
A	9	21	33	45	57	69	81	93	105	117	
A#	10	22	34	46	58	70	82	94	106	118	
B	11	23	35	47	59	71	83	95	107	119	

Figure 1.5. MIDI note value and the corresponding note [11]. Numbers in the first row show the octave

2. Literature Study

During pre-study, we investigated some references which relate to MIDI, FreeRTOS, the ultrasound sensor, and wireless sensor network using ZigBee. However, almost none of them (MIDI, FreeRTOS, ultrasound sensors, and ZigBee) are implemented together in the previous researches. This project will pick some parts of the related works to gain some insights on how should we implement the Thereminz.

MIDI

Liu et al. [20] digitalised the original Theremin by using Arduino and MAX/MSP. They modified the original Theremin to be a quad-theremin which is a Theremin that has four antennas. The Arduino is programmed using Processing language. Their work concludes that comparing with the original Theremin, playing quad-theremin is easier. They also claimed that the quad-theremin is more interactive because it can follow any scenarios which are planned on MAX/MSP.

They used term "digital Theremin" because they add a frequency-to-voltage converter LM2970 and feed it to the analogue-digital converter (ADC) in Arduino to generate MIDI message. Through MAX/MSP they defined specific functions to each antenna. The first one controls volume and velocity. The second one controls PGM and timbre during the performance. The third one controls the tremolo effect. The last one controls selection between usual way or reverse way of playing [20].

The main similarity between [20] and this project is we are sending MIDI message from Arduino to PC. However, on the other hand, Liu et al. still used the electromagnetic field as their note source since they digitalised four original Theremins into one digital Theremin. In our project, we replaced the electromagnetic field by ultrasound sensor.

Since Liu et al. used original Theremin, it is may still be challenging to play it properly due to sensitiveness. However, the quad-theremin has more MIDI controller because they configure each antenna to different functions. Our project hopefully will have more comfortable playability because we have less sensitivity than original Theremin.

In 2015, Guarnizo and Rios were developing a portable electronic percussion using some accelerometer sensors [17]. The accelerometer sensors acted as a movement sensor. By capturing the body movement of the user, the captured-movement was translated into MIDI message to produce the music. An eigenfilter identification algorithm was implemented to filter the raw movement data from the accelerometer sensor.

To avoid noise addition during a performance, they placed the sensor carefully inside a 3D-printed custom drumstick. The accelerometer sensor output is fed to ADC in a microcontroller.

Since they want to produce different sounds by some different stroke dynamics (piano, mezzo forte, and forte), they do filtering and processing of raw data from the sensors. Then, Eigenfilters is chosen. Eigenfilters enhance the signal segments that are meant to produce a sound response.

Eigenfilter is one of a statistical-type optimum filter. It is designed to maximise an input's Signal to Noise Ratio (SNR). Eigenfilter can estimate the waveform of the noise-free signal by maximising the highest component of a signal with added noise. When the system determines what sound response must be produced, it communicates with a portable device using the MIDI protocol.

The similarity between [17] and this project is accelerometer utilisation to detect a specific movement. However, in this project, we only take raw data from accelerometer sensors and filter it with a simple filter to determine a sound response.

Setiyono et al. implemented an infrared guitar instrument in 2012 [25]. Their guitar does not have strings to play the musical notes and chords. Pairs of infrared and photodiode sensors replaced the guitar strings. The user can easily block the infrared rays which are continuously received by the photodiode to play a note. The blocking signal will be processed by a microcontroller and translated to MIDI message. They claimed that their infrared guitar is more comfortable to be played than a conventional guitar. It is because the user can perform a chord effortlessly by using only one finger. Moreover, the player's fingertip will not hurt when playing the infrared guitar.

From [25], there is a similarity with this project. We both use a microcontroller to generate MIDI message to the computer. However, the infrared guitar has full MIDI tone span and can generate chord. To generate a chord, we need to play three different note at the same time. Unfortunately, Thereminz does not accommodate it.

FreeRTOS and ZigBee

Amine and Mohamed implemented both FreeRTOS and ZigBee to built their wireless sensor network which is reported in [7]. Their wireless sensor network is proposed to increase the safety in a risky area by locating the person and verify if the person has applied all of the safety standard procedures or not. The wireless sensor network itself is a system built of Arduino, FreeRTOS, and ZigBee protocol. They use six FreeRTOS tasks: Task 1 (RSSI Measurement); Task 2 (Measurement of acceleration); Task 3 (Gyroscope); Task 4 (Mobility); Task 5 (Communication); Task 6 (RFID reader). Task 5 (communication) has the higher priority among them. Unfortunately, in their report, they do not specify the ZigBee and FreeRTOS software. The result is their system able to protect the person in the workspace with limited accuracy.

In 2015, Patil and Patil implemented a real-time monitoring system in a wireless sensor network by using FreeRTOS and ZigBee [22]. FreeRTOS is running on an ATMega128A, which connects to other devices, such as ZigBee-device, sensors, and power source. In term of the FreeRTOS task, they implemented four tasks: IdleTask, SensorTask (priority 3), SerialTask (priority 1), and AlarmTask (priority 2). The semaphore is also implemented for task synchronisation. The scheduler will suspend and then call Idle Task when all of the tasks finish their task. A web-based application is also developed to acquire the data. The result of their implementation, their system works well and can be implemented where real-time response is needed.

Ultrasound sensor

In the animal world, there are some animals which using ultrasound wave to navigate their activity. One of them is bat. Bat navigation system is known as a very decent bio-sonar system. Some researchers even studied bat navigation system to develop vehicles with automatic obstacle avoidance [29].

Yamada et al. were using three adult Japanese horseshoe bats (Rhinolophus ferrumequinum nippon) in their experiment. The bats are observed in a controlled environment which has obstacles made of plastic chains that were attached to the ceiling of the room. Those bats were forced to fly in S-shaped without hitting the obstacles. Echolocation sounds emitted by the bats were recorded using custom telemetry microphone which is attached to the back of the bats. They also arranged a microphone array in the room to measure the horizontal pulse direction. By fitting the sound pressure levels vectors of each microphone in the array to a Gaussian function, then the pulse direction can be determined [29].

After extract the information from the bat's experiment, they developed an algorithm implemented to an experimental vehicle. After doing 100 obstacle-avoidance trials, the total success rates were 13% for the conventional system and 73% for double-pulse scanning (DPS) algorithm.

In Indonesia, Atmaja et al. [8] designed a parking guidance system (PGS) for UBINUS Anggrek Campus' parking. They built the PGS using ultrasound sensors (HC-SR04), a real-time operating system (FreeRTOS), microcontroller PIC32, and RS485 communication protocol.

Their system successfully helps drivers to find an available parking slot in the parking area inside UBINUS Anggrek Campus. While utilising RTOS and three ultrasound sensors, they can prevent the interference that might occur by placing each sensor in 2 m distance. They claim that the PGS can measure the distance of 10 - 280 cm.

According to [29] and [8], they both give a hypothesis that ultrasound sensors can be used as a replacement to the electromagnetic field in Theremin. Hence, a room-scale Theremin can be built of them.

3. The Antennas

In this project, we define the wireless node in the network as the antenna. There are two kinds of antennas in the project. The first one is the ultrasound antenna which will be called as distance antenna. The second one is the accelerometer antenna which will be called as acceleration antenna. In this section, those two kinds of the antenna will be discussed separately.

3.1 Distance Antenna

As mentioned earlier, the distance antenna is the antenna which utilises the ultrasound sensor as the distance detector. Within this subsection, the antenna will be discussed in two parts, hardware and software.

3.1.1 Hardware

The distance antennas primarily consist of microcontroller ATMega328P as the central processor, XBee module as the communication device, and four ultrasound sensors as the distance detector. Figure 3.1 shows the schematic for the antenna.

In term of hardware selection, we chose the ultrasound sensor as a distance sensor. Because an ultrasound sensor can sense an object up to four meters, it is possible to build a room-scale theremin. ATMega328P is chosen because a microcontroller is needed to calculate the distance. Lastly, for wireless communication, we chose to utilise XBee as it is accommodating the ZigBee protocol.

According to Figure 3.1, all of the ultrasound sensors have a dedicated trigger and echo pin. All of the trigger pins are connected to pin B, and the echoes are connected to pin D. Using microcontroller, the microcontroller will generate pulse through the trigger pin to produce an ultrasound wave within a specific interval. When the ultrasound wave hit an object, the sound wave will bounce back and captured by echo pin. More detail about the distance calculation will be discussed in the software subsection.

To notify the user, two Light Emitting Diodes (LED)s are also connected to PB0 and PB1. The LED on PB0 (Red LED) is notifying whether the user is still in the detection range or not. Another LED is notifying the XBee communication status. There is also a potentiometer connected to ADC0 through

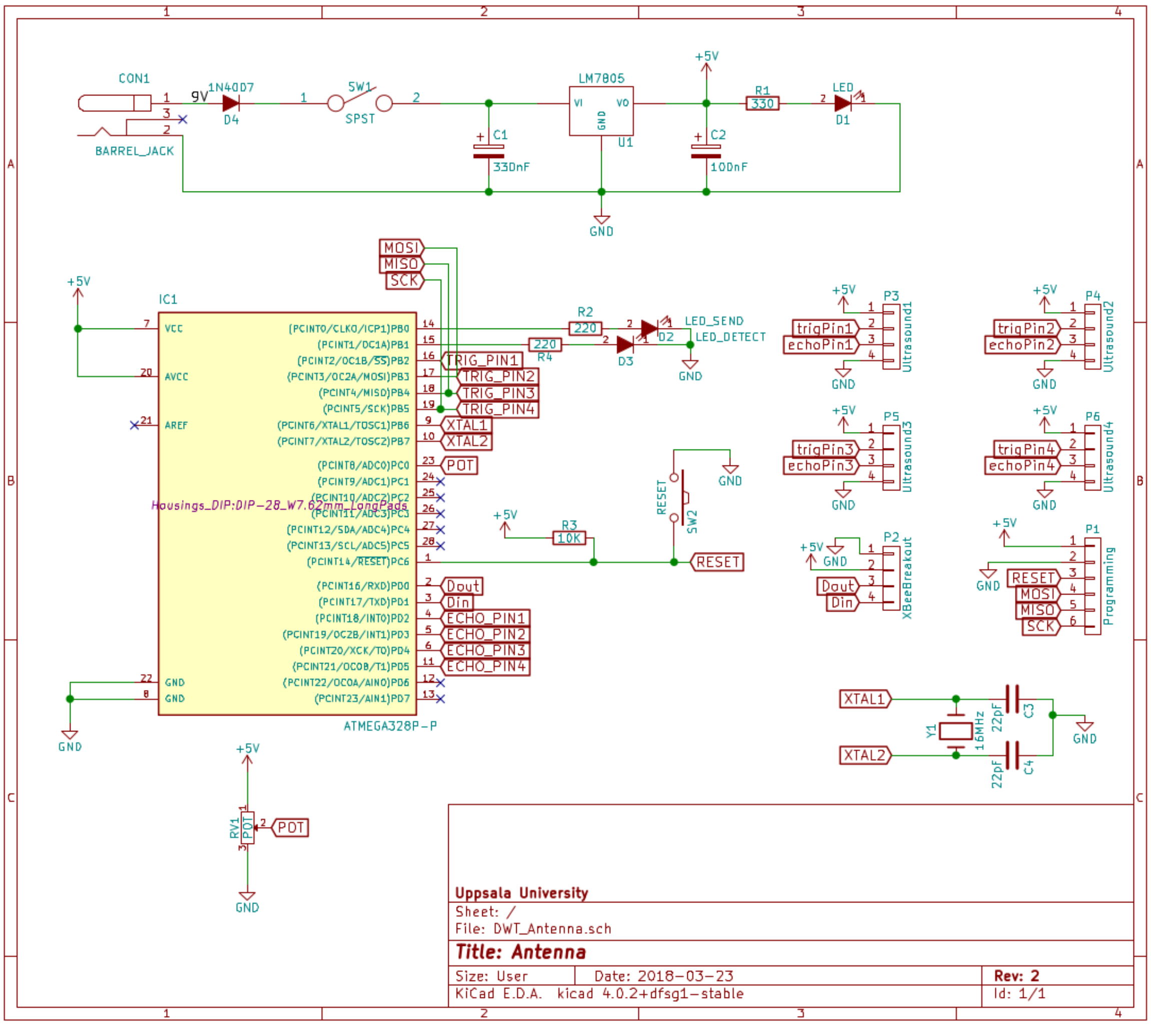


Figure 3.1. Schematic diagram of an antenna in the system

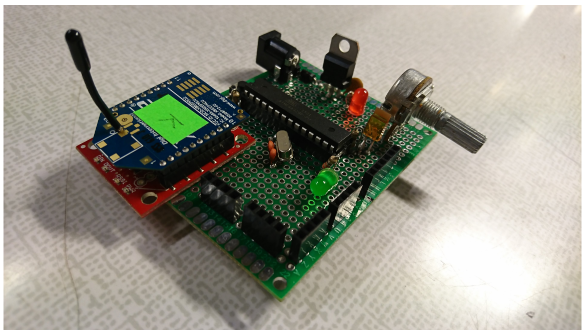


Figure 3.2. The implementation of distance antenna

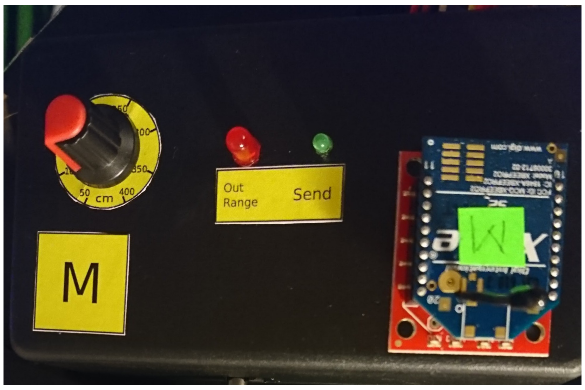


Figure 3.3. A distance antenna with the box case

pin PC0. The potentiometer can variate the maximum distance detection of the antenna. So the antenna can be fit in almost any place.

We also connected a 16 MHz crystal to XTAL1 and XTAL2 so the microcontroller can oscillate up to 16 MHz. The antenna is also programmable through programming port P1 via In System Programming (ISP).

The XBee connects to XBeeBreakout which is labelled as P2 in the Figure schematic. Port P2 is dedicated to serial communication especially XBee. Tx and Rx pin on the microcontroller is connected to Din and Dout respectively. Theoretically, as long as we have FTDI chip, those pins can also be used to do another serial communication such as access serial monitor in the Arduino IDE.

In term of the power supply, the antenna can be easily powered by a DC power source which has a maximum 35 V such as 9 V battery or a 12 V DC power source. It is recommended to use a 9V battery to power it up to make it portable. Figure 3.2 and 3.3 are showing the implemented distance antenna board and distance antenna with case respectively.

Figure 3.4 shows the final distance antenna. It was a little bit hard to adjust the ultrasounds position because they are easily interfering each other. To cover 90-degree direction, we arrange the ultrasound sensor with roughly 22.5-degree gap.

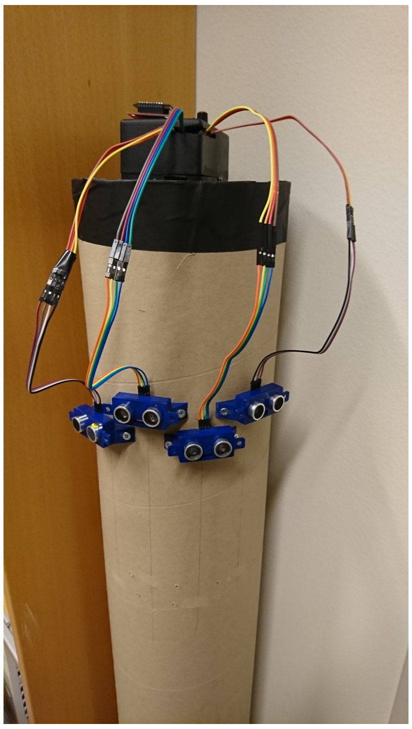


Figure 3.4. The final version of the distance antenna

3.1.2 Software

In the software part, Instead of make a bare metal software for the antenna, FreeRTOS is chosen to run it on a microcontroller. It will make future development more flexible. However the implementation seems too big for just some simple tasks, FreeRTOS gives some benefits such as modularity and easier testing.

There is a FreeRTOS library which can be used to implement it on a microcontroller. Even, there is specifically an Arduino FreeRTOS library which is compatible with several Arduino boards such as Uno, Leonardo, Mega and many more [26]. Because of it, we use Arduino FreeRTOS library in this project.

Other libraries which are used in this software are NewPing[1] and XBee[2] library. NewPing library is a library for managing ultrasound sensors. Inside it, some functions to calculate the distance is embedded. The XBee library is managing ZigBee communication via XBee. To use XBee library, the XBee must be configured in API mode. This library is supporting both XBee series 1 and 2.

In term of task design, there are two same priority tasks in the antenna software. They are TaskDistance and TaskBlinkLED. TaskBlinkLEDis always checking if the user is out-of-range or not. The second task, TaskDistance, is sending the detected distance to the base station over XBee. Listing 3.1 and

[1]https://bitbucket.org/teckel12/arduino-new-ping/wiki/Home
[2]https://github.com/andrewrapp/xbee-arduino

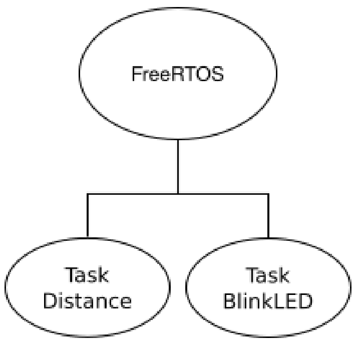


Figure 3.5. RTOS tasks design inside the antenna software

3.2 show the TaskDistance and TaskBlinkLED routine respectively. The tasks diagram is depicted in Figure 3.5.

We assigned the same priority level to both of tasks because those tasks must not preempt each other. If the tasks preempt each other, the antenna may send wrong data to base station. We tried to assign TaskDistance to have higher priority level than TaskBlinkLED. But it turned out that one of them was not executed. When TaskDistance has higher priority, scheduler always gives the time to TaskDistance and that makes TaskBlinkLED will not be executed.

Listing 3.1. TaskDistance routine in antenna K and M

```
for(;;){
        // Setup maxDistance by using Potentiometer
        maxDistance = analogRead(POT_PIN);
        maxDistance = map(maxDistance, 0, 1023, 50, 400);

        for (uint8_t i=0; i < SONAR_NUM; i++){
                sensor[currentSensor].timer_stop();
                currentSensor = i;
                cm[currentSensor] = 0;
                sensor[currentSensor].ping_timer(echoCheck, maxDistance
    );

                delay(50);
        }
} // End of for loop
```

Listing 3.2. TaskBlinkLED routine within all antennas

```
for(;;){
        messageSendStatus();

        if ( (cm[0] >= maxDistance) || (cm[1] >= maxDistance) ||
                (cm[2] >= maxDistance) || (cm[3] >= maxDistance) ){

                // Turn LED ON  to indicate "out of range
                digitalWrite(LED_DETECT, HIGH);

        }
        else {

                // Blink the Led to indicate inside the range
                flashLed(LED_DETECT, 100);

        } // End of Else statement

} // End for loop
```

In Listing 3.1, the distance is captured in line 9. Line 9 calls echoCheck() function to check if there is an object within maximum distance or not. When an object is detected, the microcontroller will capture the distance using Equation 1.1. The ultrasound sensor is pinged one by one sequentially to prevent interference within the sensor array.

The maximum distance detection for the antenna is captured by sensing an analogue signal value through pin A0 (ADC0). Then the analogue value is mapped to value between 50 to 400 which represents the maximum distance detection. A map() function in Arduino is shown by Equation 3.1. Where x is the value we want to map, in_min and in_max are the mapping minimum value and maximum value respectively, out_min and out_max are minimum and maximum target value.

$$result = (x - in\_min) * (out\_max - out\_min)/(in\_max - in\_min) + out\_min \tag{3.1}$$

To send only one distance data to the base station, we implemented a filter to filter the sensor reading. The filter is implemented as a function in Listing 3.3. It filters data from the sensors by taking the average distance between the two sensors. We consider filtering only two neighbour sensors because they are closed to each other.

Listing 3.3. Code snippet of distanceFilter() inside all distance antenna software

```
unsigned int distanceFilter(uint8_t sensor){
        if (cm[0] && cm[1]) {
                #ifdef ANTENNA_L_N
                        checkPosition();
                #endif

                avg = cm[0] + cm[1];
                avg = avg/2;
        }
        else if (cm[1] && cm[2]) {
                #ifdef ANTENNA_L_N
                        checkPosition();
                #endif

                avg = cm[1] + cm[2];
                avg = avg/2;
        }
        else if (cm[2] && cm[3]) {
                #ifdef ANTENNA_L_N
                        checkPosition();
                #endif

                avg = cm[2] + cm[3];
                avg = avg/2;
        }
        else avg = cm[sensor];

        return avg;

} // End function
```

The distance data range is between 0 to 400, but XBee library only supports 8-bit. It means we can only send distance data in range 0 to 255. Therefore, two bitmath operations are operated at line 11 and 12 in Listing 3.4. In line 11 we do bit shift and bitwise operation to capture the first byte of the distance. Then in line 12 only do bitwise to capture the second byte . When the message is sent to the base station, the base station will compile the parsed message.To illustrate how it works, there is an example in Listing 3.5.

Listing 3.4. Code snippet of pingResult() function

```
 1  void echoCheck() {
 2          if (sensor[currentSensor].check_timer()) {
 3                  timeStamp[currentSensor] = millis();
 4                  cm[currentSensor] = sensor[currentSensor].ping_result /
        US_ROUNDTRIP_CM;
 5                  pingResult(currentSensor);
 6          }
 7  }
 8
 9  void pingResult(uint8_t sensor){
10          unsigned int temp = distanceFilter(sensor);
11          payload[0] = temp >> 8 & 0xFF;
12          payload[1] = temp & 0xFF;
13          xbee.send(zbTx);
14
15          #ifdef DEBUG
16                  Serial.print(sensor);
17                  Serial.print("␣");
18                  Serial.print(temp);
19                  Serial.println("␣cm");
20          #endif
21  }
```

Listing 3.5. Bith-math operation to convert 16-bit number to 8-bit number

```
temp = 300
temp = 0b 0000 0001 0010 1100
payload[0] = 0000 0001 0010 1100 >> 8 & 0xFF = = 0000 0000 0000
     0001 & 0xFF = 0000 0001
payload[1] = 0000 0001 0010 1100 & 0xFF = 0010 1100
```

Concerning TaskBlinkLED, Listing 3.2 is showing the TaskBlinkLED. The blinking LED is decided to be implemented as a task because the checking mechanism should happen uninterruptibly. There is one essential condition in the task. If there is an object within the detection range or not. If the object is inside the range, the LED will keep blinking. Otherwise, the LED will keep turn on.

Antenna L and N not only detect the distance but also detect the user position against the antenna. Therefore, we added checkPosition() function in the software as shown in Listing 3.6. CheckPosition() function is checking user position when two ultrasound sensors in the antenna detect an object. If there is a movement from one ultrasound sensor to another within timeThreshold, the antenna will consider that as a movement. After that, the antenna will send either 800, 900, 1000 as left, right, and centre position respectively. We chose those numbers to distinguish between distance and movement detection.

Listing 3.6. Code snippet of checkPosition() inside antenna L and N software

```
void checkPosition(){
        int timeThreshold = 33;

        // Left
        if( (timeStamp[0] > timeStamp[1] + timeThreshold) ){
                int tempL = 800;
                payloadL[0] = tempL >> 8 & 0xFF;
                payloadL[1] = tempL & 0xFF;
                xbee.send(zbTxL);

                #ifdef DEBUG
                        Serial.println("left");
                #endif

        }
        // Right
        else if(timeStamp[3] > timeStamp[2] + timeThreshold){
                int tempR = 1000;
                payloadR[0] = tempR >> 8 & 0xFF;
                payloadR[1] = tempR & 0xFF;

                #ifdef DEBUG
                        Serial.println("right");
                #endif
        }
        // Center
        else {
                int tempC = 900;
                payloadC[0] = tempC >> 8 & 0xFF;
                payloadC[1] = tempC & 0xFF;
                xbee.send(zbTxC);

                #ifdef DEBUG
                        Serial.println("center");
                #endif
        }

} // End function
```

3.2 Acceleration Antenna

The hardware and software design is the same as the ultrasound antenna. The significant difference is in the power supply. In the acceleration antenna, there are two power outputs, 3.3V and 5V. In this section, we will discuss both the hardware and software implementation.

3.2.1 Hardware

As mentioned earlier, generally the hardware design is similar to the distance antenna. Figure 3.6 shows the schematic diagram of acceleration antenna. There are two accelerometer sensors (MPU6050) connected to microcontroller ATMega328P. Those two sensors connect to the microcontroller via Two Wire Interface (TWI). Also, TWI in ATMega328P is compatible with Inter-Integrated Circuit (I2C).

TWI is one of communication interface supported by ATMega328P via SCL (Serial Clock) and SDA (Serial Data) pin. It allows ATMega328P manage up to 128 slaves with different address [4].

As explained above, there are two accelerometer sensors connected to ATMega328P. To distinguish the address, we modified pin AD0 in MPU6050. In the schematic diagram of MPU6050 breakout [6], it is mentioned that two MPU6050 can be connected to the same TWI bus pin AD0 must be modified. As the default, an MPU6050 has address 0x68, and pin AD0 on the board has low logic (connected to ground). When pin AD0 is modified to have high logic (connected to VCC), the address will change to 0x69. Therefore, we modified one of the sensor's pin AD0. Thus, two MPU6050s are sharing same TWI bus with different addresses.

3.2.2 Software

Similar to the hardware part. The software for acceleration antenna is similar to distance-antenna software. However, there is only one periodic task in acceleration antenna software.

TaskIMURead, the only task, is responsible for reading acceleration data from the sensor periodically. There is motionDetection() function which is responsible to continuously polling if there is a specific movement detected. The specific movement is when X-axis or Z-axis is accelerating more than 9006 value. The value is the data from ADC reading. There is no unit for it. We chose 9006 based on experiment. That number is suitable to determine a hitting-drum movement. When a movement is detected, a message will be sent to the base station over XBee. Accelerometer sensor one will send the message “b”, and accelerometer sensor two will send message “c”. In addition, a LED is also turn on when it detects a movement. Listing 3.6 and 3.7

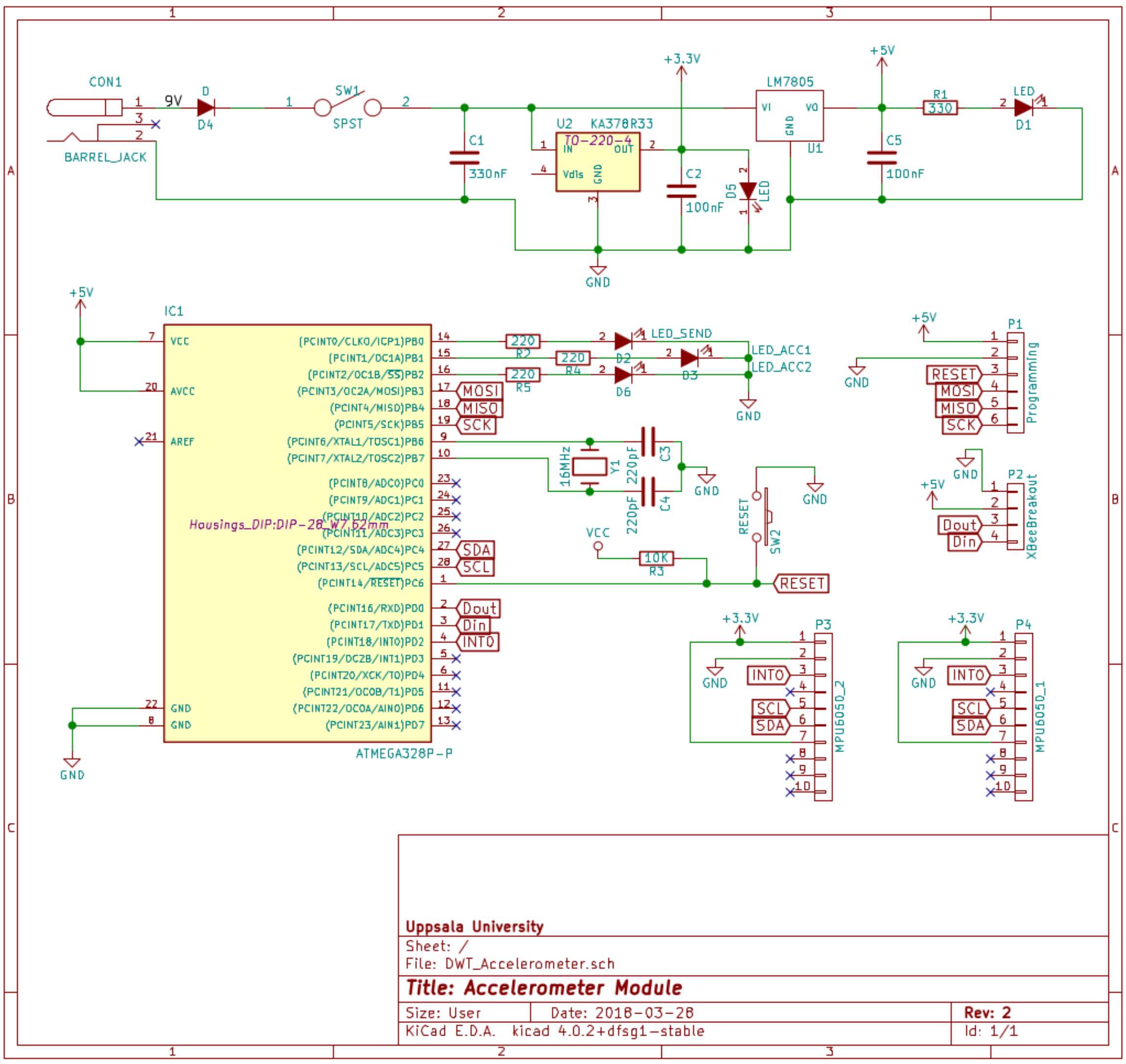


Figure 3.6. Schematic diagram of the acceleration antenna

are showing the routines.

Listing 3.7. TaskIMURead routine

```
for (;;){
        accelgyro1.getAcceleration(&ax1, &ay1, &az1);
        accelgyro2.getAcceleration(&ax2, &ay2, &az2);

        az1 = az1 - 16384;     // callibrate z (16384/g)
        az2 = az2 - 16384;     // callibrate z (16384/g)
        motionDetection();

        vTaskDelay(50/portTICK_PERIOD_MS);

}     // End for
```

Listing 3.8. Code snippet of motionDetection() function inside TaskIMURead routine

```
void motionDetection(){
        if (ax1 >= ax1Threshold || az1 >= az1Threshold) {
                digitalWrite(LED_ACC1, HIGH);
                xbee.send(zbTxC);
                messageSendStatus();

        }
        else digitalWrite(LED_ACC1, LOW);

        if (ax2 >= ax2Threshold || az2 >= az2Threshold){
                digitalWrite(LED_ACC2, HIGH);
                xbee.send(zbTxB);
                messageSendStatus();

        }
        else digitalWrite(LED_ACC2, LOW);
}
```

Not all of the codes are built from scratch. In order to increase the effectivity of MPU6050 utilisation, an MPU6060 library is used. That library is a common library when pairing an Arduino and MPU6050 sensor. In this program, we only captured raw acceleration data from the sensor. Because of that, to be able to play the acceleration antenna correctly, the two MPU6050 sensors must be precisely in a horizontal position. Otherwise, those sensors will eventually detect a movement in X or Z-axis whereas there is no movement made by the user.

4. Base Station

In this section, hardware and software of the base station will be discussed in separate sections.

4.1 Hardware

The base station is made of an Arduino, an XBee module, and some LEDs. The Arduino is acting as the controller and XBee as the communication module. There are also some LEDs which are giving notification about the wireless communication status and MIDI note-on/note-off messages. There is a switch to either sending MIDI messages or raw distance data to the computer. The switch defines what signal be sent to the computer over serial communication. A 10K variable resistor is also attached to the analogue pin A0 of Arduino. It adjusts the maximum distance of the reading sensor. Figure 4.1 shows the schematic of the base station.

The variable resistor is a little bit redundant since the maximum distance is adjustable directly from the antennas. However, because there is a possibility that the antennas send a maximum-adjusted-distance, the base station needs to filter it out to minimise an anomaly MIDI sound.

Figure 4.2 shows the final base station. Through USB cable, the base station is still programmable. However, there is a significant problem which can be happened while uploading the program. In order to upload a program to the base station, the Rx and Tx connection between XBee and microcontroller must be disconnected. Otherwise, the program cannot be uploaded. That is important because XBee and the microcontroller are sharing the same pins for serial communication.

4.2 Software

It has previously been mentioned this project runs FreeRTOS on the microcontroller. In the base station software, there are three tasks with the same priority level. They are TaskAntennaReceive, TaskAntennaKL, and TaskAntennaMN. TaskAntennaReceive is a periodic task, and the other two tasks are event-driven tasks. Figure 4.3 shows the task organisation inside the base station software.

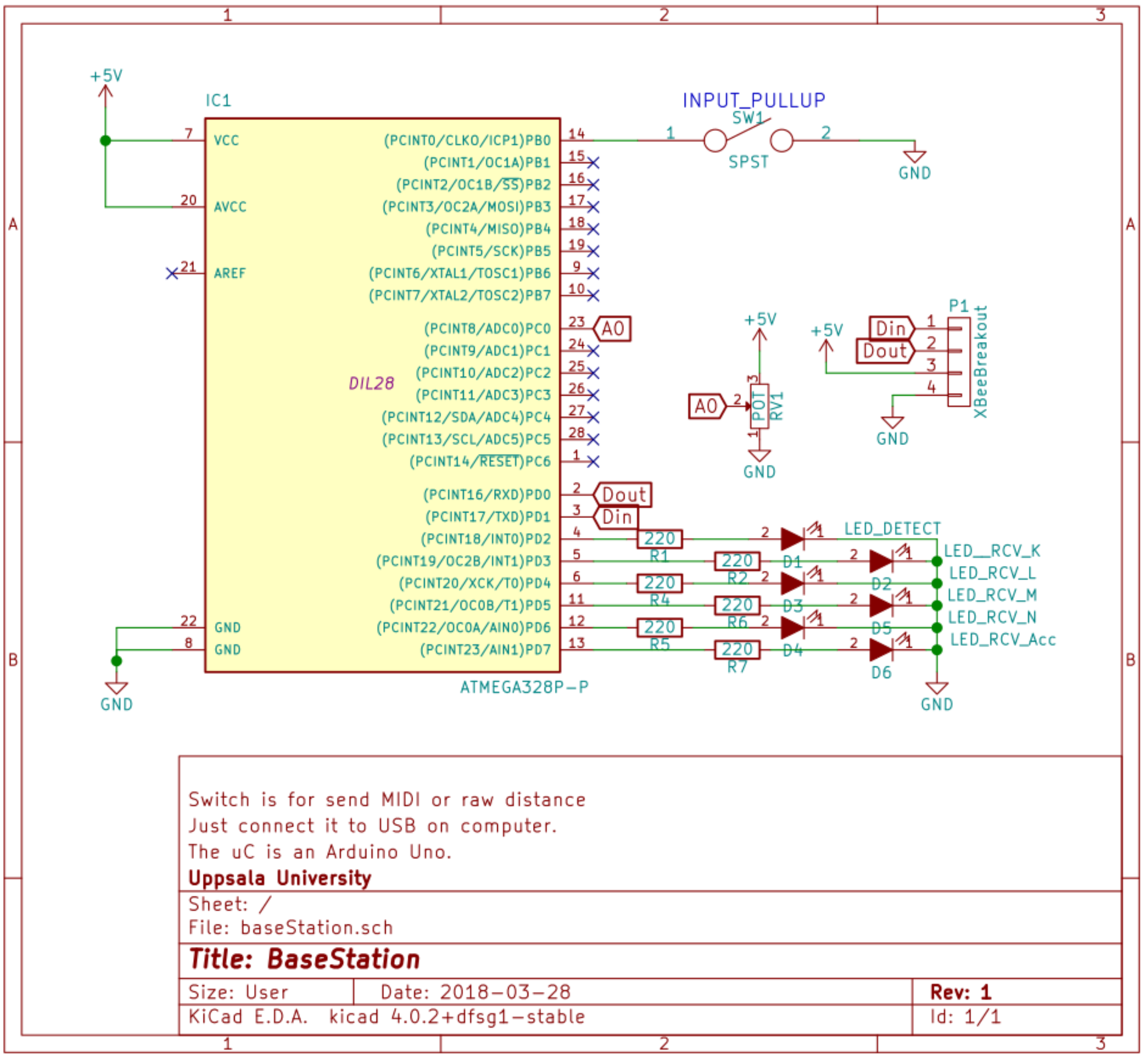


Figure 4.1. Schematic diagram of base station

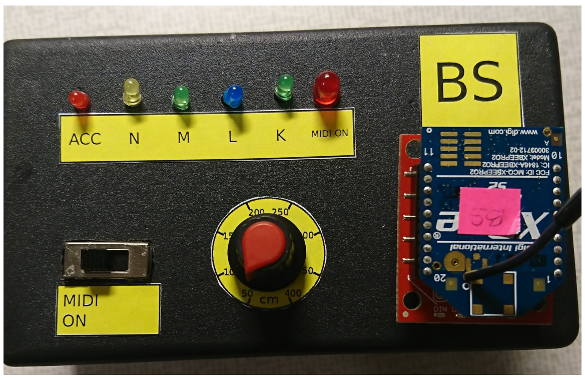


Figure 4.2. Final form of base station

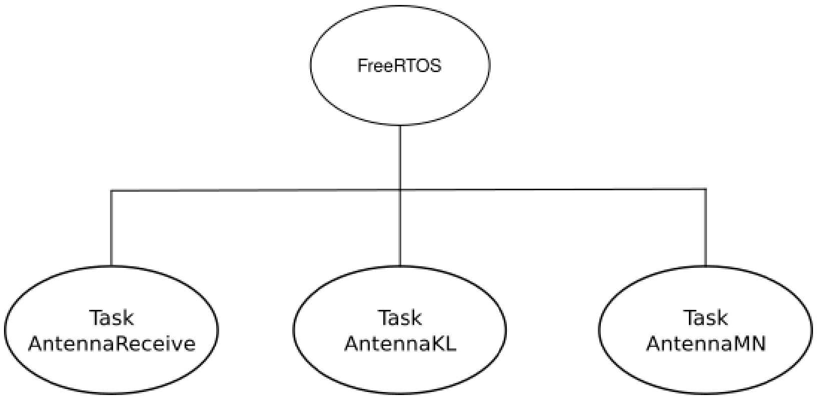


Figure 4.3. RTOS task design inside the base station software

As the default, FreeRTOS is a preemptive RTOS. That means a task can preempt another task. Regarding this part, we implemented a non-preemptive RTOS to produce some proper MIDI messages. A proper MIDI message means a note-on message must be turned-off by a note-off message in a specific interval. Otherwise, there will be a hanging note-on message which affects the sound.

Let us now consider the task routines. TaskAntennaKL and TaskAntennaMN have the same routine. They manage the received message from the respective antenna. The received message is gathered from TaskAntennaReceive. It collects parsed messages from the antenna and passes it to respective antenna subroutine. Listing 4.1 is showing the code snippets of the message collection method.

A good illustration of the workflow is investigating when there is an incoming message from antenna K. When the message comes, the message source is assessed. We compiled the message and stored it in a variable in line 15. In line 15, when the two 8-bit data arrive, they are united as one 16-bit number by doing bitmath operation. After that, a flag of the respective antenna subroutine is raised to give a signal of other tasks to run.

Listing 4.1. Code snippet of message collection method inside TaskAntennaReceive

```
1  xbee.readPacket();
2  // check if a packet was received:
3  if (xbee.getResponse().isAvailable()) {
4          if (xbee.getResponse().getApiId() == ZB_RX_RESPONSE) {
5                  xbee.getResponse().getZBRxResponse(ZbRx);
6          }
7
8          remoteAddr = ZbRx.getRemoteAddress64().getLsb();
9
10         switch (remoteAddr) {
11
12         // If there is data from Antenna K
13         case 0x5B45:
14                 // store 2 8-bit data to 1 16-bit data
15                 distanceK = (ZbRx.getData(0) * 256) + ZbRx.getData(1);
16                 flagAntennaK = true;
```

```
17                  break;
```

In the TaskAntennaKL point of view, the message is considered as distance data. If the switch is on MIDI-mode, the distance will be converted to MIDI note-on and note-off. Otherwise, it will not be converted. Listing 4.2 is showing the code snippet of it.

Listing 4.2. Code snippet of distance to MIDI conversion in the antenna K subroutine

```
1  if (!switchState){
2          digitalWrite(LED_DETECT, HIGH);
3
4          if (distanceK <= maxDistance){
5                  distanceK = map(distanceK, 1, maxDistance, targetValKLo
       , targetValKHi);
6                  note = distanceK;
7
8                  if ( xSemaphoreTake( xSerialSemph, ( TickType_t ) 5 )
       == pdTRUE ){
9
10                         digitalWrite(LED_K, HIGH);
11
12                         midi.noteOn(CH1, note, velocity);
13                         byte tempNote = note;
14                         delay(delayNoteOn);
15                         midi.noteOff(CH1, tempNote);
16                         delay(delayNoteOff);
17
18                         digitalWrite(LED_K, LOW);
19
20                         xSemaphoreGive( xSerialSemph );
21                 }
22         }
23 }
```

We can see in Listing 4.2; line 5 does the distance-to-MIDI conversion. Distance data is mapped to value between 95 and 60. According to Figure 1.5, those values represent MIDI note which covers 3 octaves.

The delayNoteOn and delayNoteOff are 150 and 50 respectively. Those delays are needed after noteOn() and noteOff() respectively in order to create an aesthetic sound. Otherwise, the noteOn and noteOff events cannot be distinguished.

Let us take another example. Now we are discussing the workflow when there is an incoming message from antenna L. We will skip the receiving-message part because it is the same for all messages.

In the antenna L point-of-view, the message will still be considered as a distance data with additional information. The additional information is values weather 800, 900, or 1000. Those values give the information of object movement. It represents move-to-left, move-to-right, and move-to-centre respectively. Using the messages from the antenna, base station send pitch-bend MIDI message over the respective channel to the computer. Listing 4.3 shows the code snippet of it.

Listing 4.3. Code snippet of distance-to-MIDI conversion in the antenna L subroutine

```
 1 if (distanceL == 800) {
 2         if ( xSemaphoreTake( xSerialSemph , ( TickType_t ) 5 ) == pdTRUE ){
 3 
 4                 digitalWrite(LED_L, HIGH);
 5                 pitchBend(CH1, localNoteL , -5000);
 6                 digitalWrite(LED_L, LOW);
 7 
 8                 xSemaphoreGive( xSerialSemph );
 9         }
10 }
11 // Center panning handling
12 else if (distanceL == 900) {
13         if ( xSemaphoreTake( xSerialSemph , ( TickType_t ) 5 ) == pdTRUE ){
14 
15                 digitalWrite(LED_L, HIGH);
16                 pitchBend(CH1, localNoteL , 0);
17                 digitalWrite(LED_L, LOW);
18 
19                 xSemaphoreGive( xSerialSemph );
20         }
21 }
22 // Right panning handling
23 else if (distanceL == 1000) {
24         if ( xSemaphoreTake( xSerialSemph , ( TickType_t ) 5 ) == pdTRUE ){
25 
26                 digitalWrite(LED_L, HIGH);
27                 pitchBend(CH1, localNoteL , 5000);
28                 digitalWrite(LED_L, LOW);
29 
30                 xSemaphoreGive( xSerialSemph );
31         }
32 }
```

We can see at line 5; the MIDI pitch-bend message is sent. Inside pitch-Bend() function there are also Note-on and note-off command sent along with pitch-bend command. It is because we need to bend a pitch (note). Otherwise, we cannot hear a note is bent when a pitch-bend is sent. A single pitchbend command is only bending a note if there a noteOn() message is sent. When there is not noteOn() command sent, a bended note will not be generated.

Semaphore

In the previous section, it is mentioned that MIDI communication based on serial communication. That indicates that XBee and MIDI are physically sharing serial communication bus. To prevent a preemption between TaskAntennaReceived, TaskAntennaKL, and TaskAntennaMN, aside assign some priority level, a binary-semaphore is also implemented. Listing 4.3 line 2 and 8 show how the semaphore is implemented.

At line 2 in the Listing 4.3, the semaphore is checked if it is available or not. If the semaphore is available, the MIDI-operation is running. Otherwise, the subroutine will still be blocked until the semaphore can be obtained.

Producing MIDI sound

There are two components which are needed to produce sound from MIDI. They are a serial-to-MIDI converter and a synthesiser. In this project, we used ttymid[1] as the serial-to-MIDI converter and fluidsynth[2] to produce some sounds. In the computer, serial-to-MIDI converter is converting the serial signal message from the base station to MIDI. Our ttymidi output is shown in Listing 4.4. Then those messages will be synthesised by fluidsynth.

Listing 4.4. Output of ttymidi when it receives MIDI message from base station

```
 1  Serial  0x90  Note on            001 085 100
 2  Serial  0x80  Note off           001 085 000
 3  Serial  0x90  Note on            000 081 100
 4  Serial  0x80  Note off           000 081 000
 5  Serial  0x90  Note on            001 000 100
 6  Serial  0xe0  Pitch bend         001 03192
 7  Serial  0x80  Note off           001 000 000
 8  Serial  0x90  Note on            001 000 100
 9  Serial  0xe0  Pitch bend         001 03192
10  Serial  0x80  Note off           001 000 000
11  Serial  0x90  Note on            000 083 100
12  Serial  0x80  Note off           000 083 000
13  Serial  0x90  Note on            000 092 100
14  Serial  0x80  Note off           000 092 000
```

According to Listing 4.4, ttymidi captured some MIDI messages. They are note-on and off in channel 2 (line 1 and 2), note-on and off in channel 1 (line 3 and 4), and pitch-bend in channel 2 (line 6 and 9).

Returning briefly to the MIDI message structure, in listing 4.4 line 1, the note-on command has three columns of numbers. Each column represents each data byte. The first column gives information on what channel is it. The second column gives information about the MIDI-note value. In addition, the last column represents the velocity.

Ideal distance to note

In term of distance-to-MIDI-note ratio, Thereminz has flexibility to adjust the ratio depends on the room size. Figure 4.4 is showing the plot of the ideal value distance to MIDI note in the Thereminz. The longer the distance detection, the distance-to-note is less sensitive. The shorter the distance, the distance-to-note is more sensitive.

[1]ttymidi is one of program to convert serial-to-midi in a computer. `http://www.varal.org/ttymidi/`

[2]fluidsynth is an synthesiser open-source software. `http://www.fluidsynth.org/`

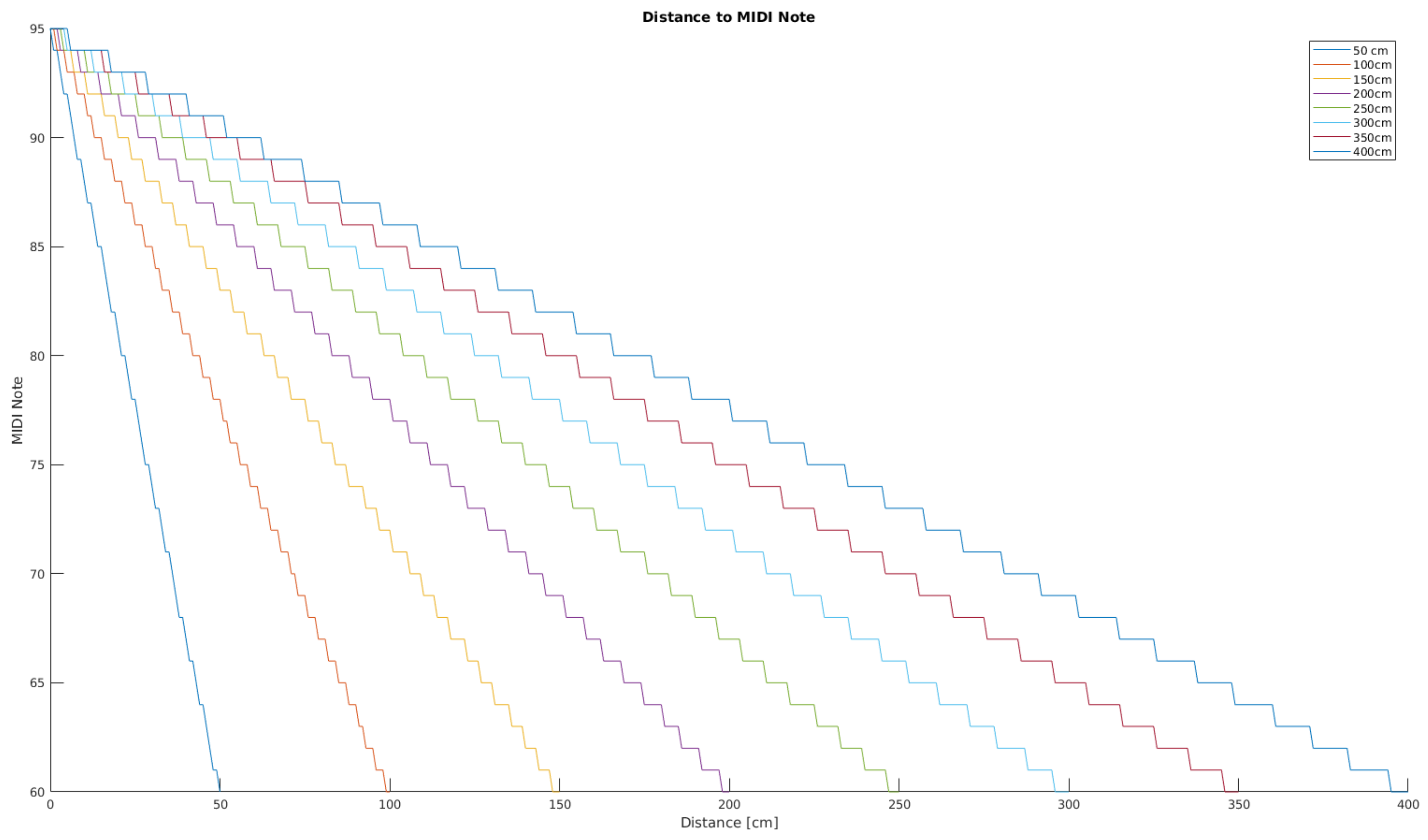


Figure 4.4. Plot of ideal distance to MIDI note in the Theremins between 50cm and 400cm

Software problem

There was a problem in the freeRTOS utilisation. In the base station, we could not create more than three tasks while utilise XBee library in API mode. In the library documentation [24], it mentioned that the library is designed to run in one thread application. Because of resource conservation, XBee library only supports storing one response packet in one time. Therefore, the antenna tasks (AntennaKL and AntennaMN) must be executed in the time prior when the flag is raised and must free the buffer so the next data can be stored safely. This solution can be applied in future development.

Another problem is sometimes the base station does not send a MIDI message after some periods. The base station does not give any response if there is an event from antenna L or N. It only gives a response to antenna K or M. We do not know yet what caused this problem. When that event occurs, the solution is just restarting the system, so the base station starts giving a response from antenna L or N.

Even though XBee has its collision-avoidance algorithm [19], the algorithm seemed not suitable for this project. The algorithm prevents the collision by creating a random delay before the packet is sent. When XBee wants to send a packet, it first checks if the channel clear or not. If the channel is clear, the packet will be sent. However, if the channel is not clear, the packet will be sent after a random time delay. The random time delay is based on the exponential backoff algorithm. When c collision happens, the algorithm will choose a time between 0 and $2c - 1$ as the delay. In the first collision, each sender will wait 0 or 1 slot time. After the second collision, senders will wait anytime between 0 and $22 - 1$ slot time. After the third collision, senders will wait anytime between 0 and $23 - 1$. The delay will increase exponentially as the number of retransmission attempts increase.

In this project, the random delay seemed not synchronised with the execution of the task. To illustrate, let's take a look following example. If there is an event detected by antenna L or N, the base station does not allocate a free slot for them. The base station is already flooded by the message from antenna K or M. The solution for future development is to implement another collision-avoidance algorithm such as do scheduling for the message transmitting. The base station gives a signal to antennas to send the message as follows scenario. The first one is signalling antenna K to send, then L to detect if there is a movement (pitch bend) or not. The second one is signalling antenna M and L sequentially.

5. Evaluation and Result

In this chapter, we evaluate the system. There are some evaluation components. The first, we evaluated the distance measurement accuracy. The second one, we evaluated the Thereminz under two configurations set up. The third, we compare the performance of Theremin and Thereminz in term of sensitivity. At last, from the technical point of view, task execution analysis is performed.

Distance measurement precision
During this evaluation, we compared the distance measurement between ultrasound sensors and ruler. The purpose of this evaluation is evaluating how accurate the ultrasound sensors compared to a ruler. There were three iterations in the experiment. The experiment was conducted in three rounds. At first two rounds, we only activated one and two ultrasound sensors in only one antenna to evaluate the accuracy of the sensor. Then, in the third round, we activated all of the antennae and set up the experiment based on Figure 5.1.

In the first two round experiments, we used antenna K as the measuring device and another (turned off) antenna to be the detected object. The object was correctly set to face the activated ultrasound sensor in antenna K. On the ground; we placed a 200 cm ruler toward ultrasound sensor direction. The object (another antenna) was moved from 10 cm to 200 cm with 10 cm gap. We monitored the ultrasound sensor measurement at the computer connected to the base station. Table 5.1 shows the experiment result.

According to Table 5.1, the standard deviation of both 1-sensor and 2-sensors measurement increase along with longer distance. The smallest deviation is 0.50 when 1-sensor measures the distance from 10 to 50 cm. On the other hand, the most significant deviation is 2.50 when 1-sensor measures 200 cm and 2-sensors measure 140, 170, 180, and 200 cm.

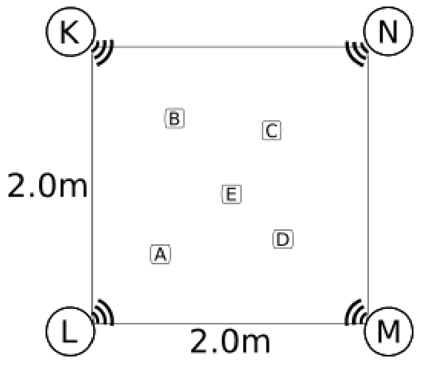


Figure 5.1. Configuration set up of third round of distance accuracy measurement

Table 5.1. *Experiment result of ruler and ultrasound sensor measurement (all units are cm)*

Ruler	1-sensor (1)	1-sensor (2)	1-sensor (3)	2-sensors (1)	2-sensors (2)	2-sensors (3)	Avg 1-sensor	Avg 2-sensors	Stdev 1-sensor	Stdev 2-sensors
10	11	11	11	12	11	12	11.00	11.67	0.50	0.96
20	21	21	21	22	21	24	21.00	23.33	0.50	1.71
30	31	31	31	32	32	32	31.00	32.00	0.50	1.00
40	41	41	41	43	42	45	41.00	43.33	0.50	2.08
50	51	51	51	53	53	54	51.00	53.33	0.50	1.73
60	62	62	63	64	64	63	62.33	63.67	1.26	1.89
70	73	72	72	74	74	74	72.33	74.00	1.26	2.00
80	82	82	82	84	85	84	82.00	84.44	1.00	2.22
90	93	92	93	94	95	95	92.67	94.67	1.41	2.38
100	102	102	103	105	105	104	102.33	104.67	1.26	2.38
110	112	112	112	113	113	113	112.00	113.33	1.00	1.73
120	123	122	122	123	123	123	122.33	123.67	1.43	1.89
130	133	133	132	134	134	133	132.67	133.67	1.73	1.89
140	143	143	144	144	144	144	143.33	144.33	1.50	2.50
150	153	153	153	154	154	154	153.00	154.00	1.50	2.50
160	163	163	163	164	164	164	163.00	164.00	1.50	2.00
170	173	173	173	175	175	175	173.00	175.00	1.50	2.50
180	183	183	184	185	185	185	183.33	185.00	1.73	2.50
190	195	195	194	195	195	195	194.67	194.67	2.38	2.38
200	205	205	205	205	205	205	205	205	2.50	2.50

We also performed regression linear to both averages of 1-sensor and 2-sensors for each measurement. The regression linear shows the standard error for 2-sensor is slightly higher than 1-sensor. Those are 0.60 and 0.47 respectively.

Those errors occur because the detected object does not have a flat surface. As we used one of the antennae as the object, it has a spherical surface. Recall briefly about the way of ultrasound sensor work based on the Doppler principle. To be able to get an accurate measurement value as ruler, a flat surface object is needed. Because of measurement accuracy is not critical in this project. We can say that the deviation is tolerable.

At the third round, we asked one participant to stand on five different spots A, B, C, D, and E one by one based on Figure 5.1. While he stood, we monitored the measurement at the computer connected to the base station. The experiment was conducted in three iterations. Table 5.2 shows the result.

Regarding Table 5.2, K-real, L-real, M-real, N-real are the measurement distance using a ruler. In three iterations, we measured each spot from four different antennae. The number shows each iteration result after the antenna name (i.e. K1, K2, K3).

Generally, the deviation for each measurement is big. Only four measurements have less than 10 cm deviation. Those big deviations occur because the ultrasound sensor did not measure the same spot as ruler did. Ruler measured the distance between the edge of the antenna and the spot. On the other hand, ultrasound sensor measured the distance between ultrasound sensor and participant's body. Physically, the participant is a big person. When he stood on a spot, ultrasound wave hit his body and measure his body's distance.

However the person stood still on the spot, the measurements were different each time. The ultrasound sensor may have crosstalk with other ultrasound sensors in the system. Because the antenna is facing each other when locating the person, the ultrasounds at the antenna are easily doing crosstalk with each other. The crosstalk made the ultrasound sensor receive wrong ultrasound wave; then it made a wrong calculation.

Thereminz evaluation

In this part, Thereminz is evaluated under two scenarios. We invited 5 participants to play the Thereminz. Each of them faced one antenna. Four participants faced distance antenna, and one participant was playing accelerometer antenna. In the first scenario, each antenna is placed at each corner of the experiment area. The second, antennas were placed in the centre of the area. Figure 5.2 and 5.3 shows the scenarios.

Under the first and second scenario, we dynamically variated maximum distance from 50 cm to 200 cm. It turned out; Thereminz did not perform as good as expected. There were some unresponsive behaviours. The pitch bend was hard to produce by base station whereas the antenna L or N is sending the

Table 5.2. *Experiment result of ruler and ultrasound sensor measurement in different spots (all units are cm)*

Spot	K real	L real	M real	N real	K1	L1	M1	N1	K2	L2	M2	N2	K3	L3	M3	N3	$\overline{K}$	$\overline{L}$	$\overline{M}$	$\overline{N}$	Std. dev K	Std. dev L	Std. dev M	Std. dev N
A	160	70	170	214	137	60	174	184	144	112	162	164	152	111	163	154	144.33	94.33	166.33	167.33	9.95	27.16	5.74	26.46
B	80	158	218	153	70	144	192	97	109	92	164	122	72	159	183	123	83.67	131.67	179.67	114.00	18.03	31.58	22.44	22.91
C	156	192	160	91	140	180	160	97	135	177	170	92	121	175	142	61	132.00	177.33	157.33	83.33	14.45	7.62	11.66	16.38
D	212	150	88	155	203	143	130	141	203	174	123	164	204	168	137	145	203.33	161.67	130.00	150.00	4.36	14.64	21.76	10.34
E	148	134	151	151	128	140	159	125	169	146	168	119	145	121	127	122	147.33	135.67	151.33	122.00	16.82	10.69	17.59	14.71

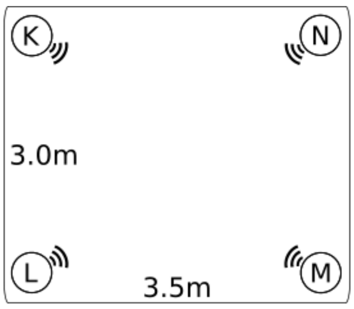


Figure 5.2. Scenario 1: placing each antenna on each corner of the area

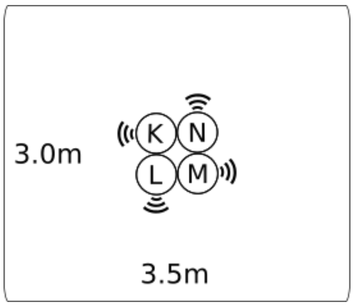


Figure 5.3. Scenario 2: placing all of the antenna in the center of the area

message to the base station. The participant must move several times in front of antenna L or N until base station gave a response.

It seems the base station was flooded by the messages from antenna K and M. Thus when a message from antenna L or N came; base station did not recognise it. In addition, there is a delay for 200 ms in base station receiving task. It makes base station loses some messages. In the future development, the delay of NoteOn/Off must be considered more carefully to prevent message looseness.

In term of interference, Thereminz can manage interference within the sensor array. However, the interference with another antenna still happens. Under scenario 2 (Figure 5.3), the interference with other antennas can be reduced. Even though, under scenario 1, the interference with other antenna occurs all the time. The interference increases gradually as long as the maximum distance. In 50 cm and 100 cm, Thereminz could manage the interference. However, in the 150cm and 200 cm, Thereminz could not manage it.

Thereminz and Theremini comparison

Beside evaluate the performance of the Thereminz, we also compared the playability with an original Theremin. There some opinions say playing the Theremin is very challenging due to its sensitivity. Even by a small movement of a finger, the pitch of original Theremin is changing [20].

Our experiment is comparing the distance-to-note performance between the original Theremin and the Thereminz. In this case, we used a modern version of Theremin by Moog (called as Theremini) as the comparison. Before the experiment was conducted, we calibrate Theremini to generate MIDI note from

Table 5.3. *Experiment result of Theremini's note generation regarding distance between pitch antenna and user hand.*

cm	MIDI 1	MIDI 2	MIDI 3	$\overline{MIDI}$	Std. dev. MIDI
0	95	95	95	95.00	0.00
1	88	86	88	87.33	1.15
2	86	84	84	84.67	1.15
3	81	83	81	81.67	1.15
4	79	79	79	79.00	0.00
5	77	77	77	77.00	0.00
6	74	76	74	74.67	1.15
7	72	74	72	72.67	1.15
8	71	72	71	71.33	0.58
9	71	71	69	70.33	1.15
10	69	69	69	69.00	0.00
11	67	67	67	67.00	0.00
12	65	65	65	65.00	0.00
13	64	64	64	64.00	0.00
14	64	64	64	64.00	0.00
15	62	62	64	62.67	1.15
16	62	62	62	62.00	0.00
17	62	62	62	62.00	0.00
18	60	60	60	60.00	0.00

60 to 95. The calibration purpose is for gaining as close as the condition between Thereminz and Theremini. During the experiment, we gradually moved our hand from 0 cm to 20 cm with 1 cm gap. The experiment is iterated three times. Table 5.3 shows the experiment result. MIDI 1, MIDI 2, MIDI 3 show the experiment iteration. After 18 cm, Theremini did not give any response. In the calibration, we selected an option that Theremini does not give a response if it does not sense hand's presence. Also, we selected the Ionian scale at root C.

Thereminz has more significant coverage area than original Theremin. To create MIDI note from 60 to 95 using Thereminz, the user must move several centimetres from one spot to another regarding maximum distance. On the other hand, using original Theremin, the movement can be done in millimetre ord. To be able to play an original Theremin properly, the player must do some intensive practices.

According to Table 5.3, generally, there is no standard deviation for each distance. It means Theremini is reliable to generate the note. Comparing column "Avg MIDI" with "MIDI Thereminz" which is the distance-to-note of Thereminz, the trend of MIDI note is similar. However, the trend of "Avg MIDI" is steeper than "MIDI Thereminz". That is because Theremini has several different scales such as Ionian, blues, major 3rd, and many more. However, Thereminz does not have the scales. Thereminz generates some plain

MIDI notes. In term of MIDI note reliability production, Thereminz is as reliable as Theremini. Thereminz can produce MIDI note regarding distance.

Execution time

In this section, we evaluate the execution time of each task in each program (i.e. taskBlinkLed in Distance Antenna, taskReceived in Base Station) separately. In order to measure the execution time, we made a timestamp in the beginning and ending of the task loop and record 1000 data from each task.

For the distance-antenna, based on the measurement, we estimate the WCET of task distance is 387 ms. This is quite a long execution time because there are four ultrasound sensors measure the distance in each 50 ms sequentially. Other time which should be taken into account is the XBee transmission time, instructions executions, and transmission collision which give the most significant time addition because of the exponential random time.

Inside Task Blink LED routine, there is messageSendStatus() function which blinks the LED regarding the message status. When the message is successfully sent to the base station, LED will blink with 50 ms delay. However, when there is an error during transmission, LED will blink with 200 ms delay. This method is not appropriate because we occasionally add a delay in the task. Another solution is replacing the LED with RGB LED. The LED can be configured to different light colour regarding message status. The messageSendStatus() function shows in the Listing 5.1

For the acceleration-antenna, we estimate the WCET is 460 ms. This time contains delay 50 ms and the delay inside messageStatus() function which must be taken into account. In the usual routine, the estimated execution time is 160 ms which consist of task delay 50 ms, blink LED delay 100 ms, and some executed instruction by the microcontroller. However, if there is an error (i.e. the base station is not turned on), the estimated WCET is 460 ms because there is 400 ms delay in LED blinking.

The most complex device, base station, we also measured the execution time of each task. On Task Receive, after collecting 1000 measurement data, we estimated the WCET is 252 ms. This time contains 200 ms task delay and instructions execution. On the other hand, Task KL and Task MN have different WCET even though they have the same routine. Based on the measurement, we estimated the WCET of Task KL and Task MN are 524 ms and 336 ms respectively. Those two tasks' WCET estimation contains a time delay of 300 ms of MIDI productions, waiting for the semaphore, and XBee collision-avoidance random time which increases exponentially.

The reason behind different WCET is Task KL was executed more frequent than Task MN. Task Receive is way too slow in data sampling. It makes Task KL, and Task MN are executed unfairly. Task KL was executed more frequent than Task MN. This data surely give a suggestion not to put delay inside semaphore.

Listing 5.1. Code snippet of messageSendStatus() function inside the distance-antenna and aceleration-antenna software

```
void messageSendStatus(){
        if (xbee.readPacket(100)) {

                // should be a znet tx status
                if (xbee.getResponse().getApiId() ==
    ZB_TX_STATUS_RESPONSE) {
                        xbee.getResponse().getZBTxStatusResponse(
    txStatus);

                        // get the delivery status, the fifth byte
                        if (txStatus.getDeliveryStatus() == SUCCESS) {
                        // success.  time to celebrate
                                flashLed(LED_SEND, 50);
                        }
                        else {

                                flashLed(LED_SEND, 200);
                        }
                }
        }
        else if (xbee.getResponse().isError()) {
        flashLed(LED_SEND, 500);
        }
}
```

6. Conclusion and Future Work

To conclude the work, regarding the evaluation we did, there is a big possibility to build a room-scale Theremin. Because of the scale and safety reasons, some ultrasound sensors can be utilised. The utilisation of the ultrasound sensor and microcontroller give the flexibility to variate maximum detection range. The maximum detection range can be adjusted between 50cm to 400cm.

Comparing to Theremini, Thereminz has less sensitivity in term of distance-to-note ratio. This is suitable to be played by ordinary people who do not play a Theremin nor Theremini before. Because of the size, Thereminz must be played by more than one person. Five people are recommended because each person can face each antenna and do the movement respectively. It is recommended to set up the Thereminz like in Figure 5.3 to prevent interference between antenna.

For the future work, there are some parts which should be improved. Especially the wireless communication (ZigBee) and FreeRTOS. In the ZigBee part, a custom collision-avoidance algorithm can be implemented to decrease the message collision occurrences. The base station can be programmed to tell the antenna to send the message in a specific time. In addition, to decrease ultrasound sensor interference between the antenna, the base station should tell other antennas not to activate ultrasound when an antenna activate its ultrasound sensor. With this solution, hopefully, the interference between antennas can be reduced drastically.

In the FreeRTOS part, the Thereminz will be more responsive if each antenna subroutines are handled by one particular task. Thus, the similar antenna can be assigned to have a different priority to others. This solution hopefully can increase the base station response to any event from the antenna.

Another recommended improvement is making some note scales like in Theremini. These scales can be chosen by investigating what scales are suitable to Thereminz. This improvement hopefully will increase the sound aesthetic and make Thereminz more playable as an alternative musical instrument.

References

[1] DIGI . Digi XBee Ecosystem - Everything you need to explore and create wireless connectivity. `https://www.digi.com/xbee`. visited on 2018-04-05.

[2] FreeRTOS . FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions. `https://freertos.org/index.html`. visited on 2018-03-20.

[3] Intorobotics . Ultrasonic Sensors – Tutorials and Resources. `https://www.intorobotics.com/interfacing-programming-ultrasonic-sensors-tutorials-resources/`. visited on 2018-04-24.

[4] Microchip . AVR ATMega328p Datasheet, 2018.

[5] MIDI.org . Homepage MIDI.org. `https://www.midi.org/`. visited on 2018-04-25.

[6] Sparkfun . SparkFun Triple Axis Accelerometer and Gyro Breakout - MPU-6050 - SEN-11028 - SparkFun Electronics. `https://www.sparkfun.com/products/11028`. visited on 2018-04-25.

[7] C. M. El Amine and O. Mohamed. A localization and an identification system of personnel in areas at risk using a wireless sensor network. In *2013 The International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE)*, pages 127–131, May 2013.

[8] W. Atmadja, J. Yosafat, R. A. Setiawan, and I. I. Irendy. Parking guidance system based on real time operating system. In *2014 International Conference on Industrial Automation, Information and Communications Technology*, pages 5–8, August 2014.

[9] Carmen Bachiller Martín, Jorge Sastre Martínez, Amelia Ricchiuti, Héctor Esteban González, and Carlos Hernández Franco. Study of the Interference Affecting the Performance of the Theremin, 2012.

[10] Richard Barry. Mastering The FreeRTOS Real Time Kernel, 2016.

[11] Greg Cerveny. MIDI Note Number Chart for iOS Music Apps. `https://medium.com/@gmcerveny/midi-note-number-chart-for-ios-music-apps-b3c01df3cb19`, September 2017. visited on 2018-05-22.

[12] Mike Cook. Basic MIDI. In *Arduino Music and Audio Projects*, pages 31–47. Apress, Berkeley, CA, 2015.

[13] Mike Cook. MIDI Instruments. In *Arduino Music and Audio Projects*, pages 103–139. Apress, Berkeley, CA, 2015.

[14] Mike Cook. More MIDI. In *Arduino Music and Audio Projects*, pages 49–65. Apress, Berkeley, CA, 2015.

[15] Fred Eady. *Hands-On ZigBee: Implementing 802.15.4 with Microcontrollers*. Elsevier Science & Technology, Oxford, UNITED STATES, 2007.

[16] Robert Faludi. *Building Wireless Sensor Networks*. O'Reilly Media, Inc, United States of America, first edition, 2011.

[17] E. A. S. Guarnizo and L. M. R. Rios. Portable percussion MIDI controller. In *2015 20th Symposium on Signal Processing, Images and Computer Vision (STSIVA)*, pages 1–7, September 2015.

[18] Leslie Hodges. Ultrasonic and Passive Infrared Sensor Integration for Dual Technology User Detection Sensors, 2011.

[19] DIGI International. Product Manual v1.xEx - 802.15.4 Protocol. `http://www.digi.com`, 2009.

[20] Tsung-Ching Liu, Shu-Hui Chang, and Che-Yi Hsiao. A modified Quad-Theremin for interactive computer music control. In *2011 International Conference on Multimedia Technology*, pages 6179–6182, July 2011.

[21] Elijah J Morgan. HCSR04 Ultrasonic Sensor, November 2014.

[22] S. P. Patil and S. C. Patil. A real time sensor data monitoring system for wireless sensor network. In *2015 International Conference on Information Processing (ICIP)*, pages 525–528, December 2015.

[23] Emery Premeaux, Brian Evans, and Michael Turner. *Arduino projects to save the world*. Springer, 2011.

[24] Andrew Rapp. xbee-arduino: Arduino library for communicating with XBee radios in API mode. `https://github.com/andrewrapp/xbee-arduino`, April 2018. original-date: 2015-02-17T03:56:48Z.

[25] R. Setiyono, A. S. Prihatmanto, and P. H. Rusmin. Design and implementation Infrared Guitar based on playing chords. In *2012 International Conference on System Engineering and Technology (ICSET)*, pages 1–5, September 2012.

[26] Phillip Stevens. Arduino_freertos_library: A FreeRTOS Library for all Arduino AVR Devices (Uno, Leonardo, Mega, etc), April 2018. original-date: 2015-11-26T01:56:10Z.

[27] Stefano Tennina, Anis Koubâa, Roberta Daidone, Mário Alves, Petr Jurčík, Ricardo Severino, Marco Tiloca, Jan-Hinrich Hauer, Nuno Pereira, Gianluca Dini, Mélanie Bouroche, and Eduardo Tovar. *IEEE 802.15.4 and ZigBee as Enabling Technologies for Low-Power Wireless Systems with Quality-of-Service Constraints*. SpringerBriefs in Electrical and Computer Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[28] World Health Organisation (WHO). WHO | What are electromagnetic fields? `http://www.who.int/peh-emf/about/WhatisEMF/en/`. visited on 2018-05-13.

[29] Y. Yamada, K. Ito, R. Kobayashi, and S. Hiryu. Obstacle avoidance navigation system for cheap design sensing inspired by bio-sonar navigation of bats. In *2017 56th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, pages 3–6, September 2017.

Appendices

Listing 1. Source code for distance antenna

```
/*
 * Dancing with Theremin
 *
 * Rahmanu Hermawan
 * Master Thesis under MUMIN Project
 * Uppsala University
 * Sweden
 *
 * This program is for the Distance Antenna in the system
 *
 * Some libraries are used, they are:
 * 1. Arduino FreeRTOS by https://github.com/feilipu/Arduino_FreeRTOS_Library
 * 2. Xbee library by Andrew Rapp (https://github.com/andrewrapp/xbee-arduino)
 * 3. NewPing library by Tim Eckel (https://playground.arduino.cc/Code/NewPing)
 *
 */


#include <Arduino.h>    // In Eclipse-IDE, this library is needed (must be uncommented)
#include <Arduino_FreeRTOS.h>
#include <XBee.h>
//#include <NewPing.h>
#include "libraries/NewPing/src/NewPing.h"

/*==============================================*/
/*      GLOBAL              */
/*==============================================*/

void TaskDistance( void *pvParameters );
void TaskBlinkLED( void *pvParameters );

void detectExistance();
void checkPosition();
int distanceFilter();
void flashLed(int pin, int wait);
void messageSendStatus();
void echoCheck();
void pingResult(uint8_t sensor);
unsigned int distanceFilter(uint8_t sensor);

//#define DEBUG         // uncomment this for activating debug mode (i.e send serial print)
#define ANTENNA_L_N     // uncomment this for activating antenna L and N

#define LED_DETECT 8
#define LED_SEND 9
#define SONAR_NUM 4
#define PING_INTERVAL 33 // Milliseconds between sensor pings (29ms is about the min to avoid cross-sensor echo).

```

```
#define POT_PIN A0

// Sensor A
#define TRIG_PIN1 10
#define ECHO_PIN1 2 // PD2
// Sensor B
#define TRIG_PIN2 11 // PB3
#define ECHO_PIN2 3 // PD3
// Sensor C
#define TRIG_PIN3 12 // PB4
#define ECHO_PIN3 4 // PD4
// Sensor D
#define TRIG_PIN4 13 // PB5
#define ECHO_PIN4 5 // PD5


// Variables //
int maxDistance; // centimeter
int avg;
unsigned long timeStamp[SONAR_NUM] = {0, 0, 0, 0};
unsigned long pingTimer[SONAR_NUM];
unsigned int cm[SONAR_NUM];
uint8_t currentSensor = 0;

// Sensor ultrasonic object array
NewPing sensor[SONAR_NUM] = {
  NewPing(TRIG_PIN1, ECHO_PIN1, maxDistance),
  NewPing(TRIG_PIN2, ECHO_PIN2, maxDistance),
  NewPing(TRIG_PIN3, ECHO_PIN3, maxDistance),
  NewPing(TRIG_PIN4, ECHO_PIN4, maxDistance)
};

//Create new instance of XBee library:
XBee xbee = XBee();

uint8_t payload[] = { 0, 0 };
uint8_t payloadL[] = { 0, 0 }; // LEFT
uint8_t payloadC[] = { 0, 0 }; // CENTER
uint8_t payloadR[] = { 0, 0 }; // RIGHT

// uint8_t payloadDummy[] = { 'L'};

// SH + SL Address of receiving XBee
XBeeAddress64 sink64 = XBeeAddress64(0x0013a200, 0x408D5A3E);    // Send
       to Base Station
ZBTxStatusResponse txStatus = ZBTxStatusResponse();        // Status
     response

ZBTxRequest zbTx = ZBTxRequest(sink64, payload, sizeof(payload));
ZBTxRequest zbTxL = ZBTxRequest(sink64, payloadL, sizeof(payloadL));
ZBTxRequest zbTxC = ZBTxRequest(sink64, payloadC, sizeof(payloadC));
ZBTxRequest zbTxR = ZBTxRequest(sink64, payloadR, sizeof(payloadR));

/*================================================*/
/*       SETUP              */
/*================================================*/
```

```

void setup() {
  Serial.begin(115200);

  pinMode(LED_SEND, OUTPUT);
  pinMode(LED_DETECT, OUTPUT);

  pingTimer[0] = millis() + 75;
  for (uint8_t i = 1; i < SONAR_NUM; i++)  pingTimer[i] = pingTimer[i -
     1] + PING_INTERVAL;

  // XBee Setup
  // Tell XBee to use Hardware Serial
  xbee.setSerial(Serial);

  // Task Stuffs

  xTaskCreate(
  TaskDistance
    ,  (const portCHAR *) "Distance"
    ,  128  // Stack size
    ,  NULL
    ,  1  // Priority
    ,  NULL );

  xTaskCreate(
  TaskBlinkLED
  ,  (const portCHAR *) "BlinkLED"
  ,  128  // Stack size
  ,  NULL
  ,  1  // Priority
  ,  NULL );

} // End setup

void loop()
{
  // Empty. Things are done in Tasks.
}

/*====================================================*/
/*       TASKS             */
/*====================================================*/

void TaskDistance(void *pvParameters){
  /* This task will continuously sending the distance data trough
     serial
   *
   * cm[0] = distance from sensor A
   * cm[1] = distance from sensor B
   * cm[2] = distance from sensor C
   * cm[3] = distance from sensor D
   *
   * */

  (void) pvParameters;

  for(;;){
    maxDistance = analogRead(POT_PIN);
    maxDistance = map(maxDistance, 0, 1023, 50, 400);

    for (uint8_t i=0; i < SONAR_NUM; i++){
        sensor[currentSensor].timer_stop();
        currentSensor = i;
        cm[currentSensor] = 0;
        sensor[currentSensor].ping_timer(echoCheck, maxDistance);

        delay(50);
    }
  } // End of for loop

} // End of Task Send Data

void TaskBlinkLED(void *pvParameters){
  /* This task will keep periodically checking if the detected distance
     is inside the range or not,
   * If the detected distance is out of range,  the LED will not
     blinking.
   *
   * Inside this task, we also are able to set the maximum distance
     using potentiometer.
   * The number there is in centimeter.
   * */

  (void) pvParameters;

  for(;;){
    messageSendStatus();

     if ( (cm[0] >= maxDistance) || (cm[1] >= maxDistance) ||
         (cm[2] >= maxDistance) || (cm[3] >= maxDistance) ){

      // Turn LED ON  to indicate "out of range
      digitalWrite(LED_DETECT, HIGH);

      }
      else {

        // Blink the Led to indicate inside the range
        flashLed(LED_DETECT, 100);

      } // End of Else statement

  } // End for loop

} // End of Task Blink LED

/*==================================================*/
/*        FUNCTIONS            */
/*==================================================*/

/*
```

```
 * Function: checkPosition()
 *
 * It is a function to detect if there is a movement from a sensor to
   another sensor or not.
 * The detection is performed by two ultrasound sensors.
 * The algorithm uses a time difference comparison between those two
   sensors.
 *
 */
void checkPosition(){
  int timeThreshold = 50;

  // Left
  if( (timeStamp[0] > timeStamp[1] + timeThreshold) ){
    int tempL = 800;
    payloadL[0] = tempL >> 8 & 0xFF;
    payloadL[1] = tempL & 0xFF;
    xbee.send(zbTxL);

    #ifdef DEBUG
      Serial.println("left");
    #endif

  }
  // Right
  else if(timeStamp[3] > timeStamp[2] + timeThreshold){
    int tempR = 1000;
    payloadR[0] = tempR >> 8 & 0xFF;
    payloadR[1] = tempR & 0xFF;
    xbee.send(zbTxR);

    #ifdef DEBUG
      Serial.println("right");
    #endif
  }
  // CENTER AREA
  else {
    int tempC = 900;
    payloadC[0] = tempC >> 8 & 0xFF;
    payloadC[1] = tempC & 0xFF;
    xbee.send(zbTxC);

    #ifdef DEBUG
      Serial.println("center");
    #endif
  }

} // End function

/*
 * Function: flashLed()
 *
 * Just a function to blink a LED.
 *
 */
void flashLed(int pin, int wait) {

    digitalWrite(pin, HIGH);
    delay(wait);
    digitalWrite(pin, LOW);
    delay(wait);
}

/*
 * Function: messageSendStatus()
 *
 * This is a function to give information if the message sending is
   SUCCESS, UNSUCCESS, or ERROR.
 * I cited from xbee library by Adrew Rapp
 *
 */
void messageSendStatus(){
  if (xbee.readPacket(100)) {
    // got a response!

    // should be a znet tx status
    if (xbee.getResponse().getApiId() == ZB_TX_STATUS_RESPONSE) {
      xbee.getResponse().getZBTxStatusResponse(txStatus);

      // get the delivery status, the fifth byte
      if (txStatus.getDeliveryStatus() == SUCCESS) {
        // success.  time to celebrate
        flashLed(LED_SEND, 50);
      }
      else {
        flashLed(LED_SEND, 200);
      }
    }
  }
  else if (xbee.getResponse().isError()) {
    flashLed(LED_SEND, 500);
  }

}
/*
 * Function: echoCheck()
 *
 * It is a function to check if there is an object detected within
   range
 *
 */
void echoCheck() {
  if (sensor[currentSensor].check_timer()) {
    timeStamp[currentSensor] = millis();
    cm[currentSensor] = sensor[currentSensor].ping_result /
    US_ROUNDTRIP_CM;
    pingResult(currentSensor);
  }
}

/*
 * Function: pingResult(uint8_t sensor)
 *
 * It is a function to capture the distance measurement,
 * then send it to base station over XBee
 *
 */
void pingResult(uint8_t sensor){
    unsigned int temp = distanceFilter(sensor);
    payload[0] = temp >> 8 & 0xFF;
    payload[1] = temp & 0xFF;
    xbee.send(zbTx);

  #ifdef DEBUG
    Serial.print(sensor);
    Serial.print("␣");
    Serial.print(temp);
    Serial.println("␣cm");
  #endif
}

/*
 * Function: distanceFilter(uint8_t sensor)
 *
 * It is a function to filter the distance measurement.
 * If 2 sensors detect an object, it will return the average of it.
 *
 */
unsigned int distanceFilter(uint8_t sensor){
  if (cm[0] && cm[1]) {
        #ifdef ANTENNA_L_N
          checkPosition();
        #endif

    avg = cm[0] + cm[1];
    avg = avg/2;
  }
  else if (cm[1] && cm[2]) {
        #ifdef ANTENNA_L_N
          checkPosition();
        #endif

    avg = cm[1] + cm[2];
    avg = avg/2;
  }
  else if (cm[2] && cm[3]) {
        #ifdef ANTENNA_L_N
          checkPosition();
        #endif

    avg = cm[2] + cm[3];
    avg = avg/2;
  }
  else avg = cm[sensor];

  return avg;

} // End function
```

Listing 2. Source code for acceleration antenna

```
/*
 * Dancing with Theremin
 *
 * Rahmanu Hermawan
 * Master Thesis under MUMIN Project
 * Uppsala University
 * Sweden
 *
 * This program is for the accelerometer Antenna in the system.
 * This program is adapted from https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050
 *
 * Some libraries are also used in this program, they are:
 * 1. Arduino FreeRTOS by https://github.com/feilipu/Arduino_FreeRTOS_Library
 * 2. Xbee library by Andrew Rapp (https://github.com/andrewrapp/xbee-arduino)
 * 3. MPU6050 library by J. Rowberg (https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050)
 *
 */

#include <Arduino.h>                    // in Eclipse-IDE, this library is needed
#include <Arduino_FreeRTOS.h>
#include <XBee.h>
#include "libraries/MPU6050/MPU6050.h"
//#include "libraries/ArduMIDI/ArduMIDI.h"

/*==================================================*/
/*                      GLOBAL                      */
/*==================================================*/

void TaskIMURead( void *pvParameters );

void flashLed(int pin, int wait);
void messageSendStatus();
void motionDetection();


// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

#define LED_SEND 8
#define LED_ACC1 9
#define LED_ACC2 10

bool blinkState = false;
int ax1Threshold = 9006, az1Threshold = 9006, ax2Threshold = 9006, az2Threshold = 9006; // motion detection threshold

```

```
int16_t ax1, ay1, az1;
int16_t ax2, ay2, az2;
int16_t tempSense;

MPU6050 accelgyro1; // Addr 0x68
MPU6050 accelgyro2(0x69); // Addr 0x69 (with AD0 is connected to VDD)

//Create new instance of XBee library:
XBee xbee = XBee();

uint8_t payloadAcc1[] = { 'b' }; // Acc 1 'b' in ASCII = 0x62
uint8_t payloadAcc2[] = { 'c' }; // Acc 2 'c' in ASCII = 0x63

// SH + SL Address of receiving XBee
XBeeAddress64 sink64 = XBeeAddress64(0x0013a200, 0x408D5A3E);
ZBTxStatusResponse txStatus = ZBTxStatusResponse();

ZBTxRequest zbTxB = ZBTxRequest(sink64, payloadAcc1, sizeof(payloadAcc1
    ));
ZBTxRequest zbTxC = ZBTxRequest(sink64, payloadAcc2, sizeof(payloadAcc2
    ));

/*==================================================*/
/*                        SETUP                     */
/*==================================================*/

// the setup function runs once when you press reset or power the board
void setup() {

        // join I2C bus (I2Cdev library doesn't do this automatically)
        #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
                Wire.begin();
        #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
                Fastwire::setup(400, true);
        #endif

        //          midi.begin();
        Serial.begin(115200);

        // initialize device
        accelgyro1.initialize();
        accelgyro2.initialize();

        // verify connection
        Serial.println("Testing device connections...");
        Serial.println(accelgyro1.testConnection() ? "MPU6050_1 
    connection successful" : "MPU6050_1 connection failed");
        Serial.println(accelgyro2.testConnection() ? "MPU6050_2 
    connection successful" : "MPU6050_2 connection failed");

        pinMode(LED_SEND, OUTPUT);
        pinMode(LED_ACC1, OUTPUT);
        pinMode(LED_ACC2, OUTPUT);

        // supply your own accelgyro offsets here, scaled for min
    sensitivity
```

```
        accelgyro1.setXAccelOffset(-272);
        accelgyro1.setYAccelOffset(780);
        accelgyro1.setZAccelOffset(1071);
        accelgyro1.setXGyroOffset(167);
        accelgyro1.setYGyroOffset(-9);
        accelgyro1.setZGyroOffset(-2);

        // supply your own gyro offsets here, scaled for min
    sensitivity
        accelgyro2.setXAccelOffset(1101);
        accelgyro2.setYAccelOffset(-939);
        accelgyro2.setZAccelOffset(1107);
        accelgyro2.setXGyroOffset(81);
        accelgyro2.setYGyroOffset(36);
        accelgyro2.setZGyroOffset(-5);

        // Tasks
        xTaskCreate(
        TaskIMURead
        ,  (const portCHAR *) "IMURead"
        ,  128
        ,  NULL
        ,  1
        ,  NULL );

        // Now the task scheduler, which takes over control of
    scheduling individual tasks, is automatically started.
}

void loop()
{
  // Empty. Things are done in Tasks.
}

/*=================================================*/
/*                       TASKS                     */
/*=================================================*/

void TaskIMURead(void *pvParameters){  // This is a task.
        /*
         * This task is responsible for acquiring acceleration data
    from sensor 1 and sensor 2.
         * The period for this task is 50 ms.
         *
         */

        (void) pvParameters;

        for (;;){

                accelgyro1.getAcceleration(&ax1, &ay1, &az1);
                accelgyro2.getAcceleration(&ax2, &ay2, &az2);

                az1 = az1 - 16384;      // callibrate z to be as close
    as 0 (16384/g)
```

```
                az2 = az2 - 16384;      // callibrate z to be as close as 0 (16384/g)


                motionDetection();

                vTaskDelay(50/portTICK_PERIOD_MS);

        }       // End for
}       // End task

/*==================================================*/
/*                      FUNCTIONS                   */
/*==================================================*/

/*
 * Function: flashLed()
 *
 * Just a function to blink a LED.
 *
 */

void flashLed(int pin, int wait) {

    digitalWrite(pin, HIGH);
    delay(wait);
    digitalWrite(pin, LOW);
    delay(wait);
}

/*
 * Function: messageSendStatus()
 *
 * This is a function to give information if the message sending is SUCCESS, UNSUCCESS, or ERROR.
 * I cited from xbee library by Adrew Rapp
 *
 */
void messageSendStatus(){
        if (xbee.readPacket(100)) {

                // should be a znet tx status
                if (xbee.getResponse().getApiId() == ZB_TX_STATUS_RESPONSE) {
                        xbee.getResponse().getZBTxStatusResponse(txStatus);

                        // get the delivery status, the fifth byte
                        if (txStatus.getDeliveryStatus() == SUCCESS) {
                                // success.  time to celebrate
                                flashLed(LED_SEND, 50);
                        } else {
                          // the remote XBee did not receive our packet. is it powered on?
                          flashLed(LED_SEND, 200);
                        }
                }
         }
        else if (xbee.getResponse().isError()) {
                flashLed(LED_SEND, 500);
        }

}

/*
 * Function: motionDetection()
 *
 * This function is responsible to always checking if there is a movement which is beyond the threshold or not.
 * When that is happen, a message will be sent to base station through xbee.
 *
 */
void motionDetection(){
        if (ax1 >= ax1Threshold || az1 >= az1Threshold) {
                digitalWrite(LED_ACC1, HIGH);
                xbee.send(zbTxC);
                messageSendStatus();

        }
        else digitalWrite(LED_ACC1, LOW);

        if (ax2 >= ax2Threshold || az2 >= az2Threshold){
                digitalWrite(LED_ACC2, HIGH);
                xbee.send(zbTxB);
                messageSendStatus();

        }
        else digitalWrite(LED_ACC2, LOW);
}
```

Listing 3. Source code for base station

```
/*
 * Dancing with Theremin
 *
 * Rahmanu Hermawan
 * Master Thesis under MUMIN Project
 * Uppsala University
 * Sweden
 *
 * This program is for the Base Station in the system.
 *
 * Some libraries are used, they are:
 * 1. Arduino FreeRTOS by https://github.com/feilipu/Arduino_FreeRTOS_Library
 * 2. Xbee library by Andrew Rapp (https://github.com/andrewrapp/xbee-arduino)
 * 3. ArduMIDI library by (http://github.com/Pecacheu/ArduMIDI/)
 *
 */

```

```
#include <Arduino.h>            // In Eclipse-IDE, this library is needed
#include <Arduino_FreeRTOS.h>
#include "semphr.h"
#include <HardwareSerial.h>
#include <XBee.h>
#include "libraries/ArduMIDI/ArduMIDI.h"
//#include <ArduMIDI.h>         // uncomment this line if work in Arduino IDE

/*==================================================*/
/*       GLOBAL              */
/*==================================================*/

// define tasks
void TaskAntennaReceived( void *pvParameters );
void TaskAntennaKL( void *pvParameters );
void TaskAntennaMN( void *pvParameters );

void flashLed(uint8_t pin, int wait);
void pitchBend(uint8_t ch, uint8_t localNote, int bendVal);


#define LED_MIDI_ON 2
#define LED_K 3
#define LED_L 4
#define LED_M 5
#define LED_N 6
#define LED_ACC 7

#define POT_PIN A0
#define SWITCH 8

SemaphoreHandle_t xSerialSemph;

uint8_t switchState = 0;
uint16_t maxDistance = 0;
uint16_t distanceK = 0, distanceL = 0, distanceM = 0, distanceN = 0, messageAcc = 0;
uint16_t delayNoteOn = 150;
uint16_t delayNoteOff = 50;


// MIDI stuff
uint8_t noteK = 0, noteM = 0, velocity = 100;

// Antenna Acc MIDI variables
uint8_t beatNote = 81;  // drum note
uint8_t beatVel = 100;  // Velocity value for the beat sound

//Create new instance of ArduMIDI library:
ArduMIDI midi = ArduMIDI(Serial, CH_ALL); // all channels

// Serial stuffs
int remoteAddr = 0;
bool flagAntennaK = false, flagAntennaL = false,
        flagAntennaM = false , flagAntennaN = false ,
        flagAntennaAcc = false , flagNothing = false;

// XBee stuffs
XBee xbee = XBee();
XBeeResponse response = XBeeResponse();

ZBRxResponse ZbRx = ZBRxResponse();


/*================================================*/
/*      SETUP           */
/*================================================*/

// the setup function runs once when you press reset or power the board
void setup() {

        midi.begin(); // Serial communication using 115200 baud rate.

        // Tell XBee to use Hardware Serial
        xbee.begin(Serial);

        pinMode(LED_MIDI_ON, OUTPUT);
        pinMode(LED_K, OUTPUT);
        pinMode(LED_L, OUTPUT);
        pinMode(LED_M, OUTPUT);
        pinMode(LED_N, OUTPUT);
        pinMode(LED_ACC, OUTPUT);

        pinMode(SWITCH,INPUT_PULLUP);

        // Now set up tasks to run independently.
        xTaskCreate(
        TaskAntennaReceived
          , (const portCHAR *) "TaskAntennaReceived"
          , 128  // Stack size
          , NULL
          , 1  // Priority
          , NULL );

        xTaskCreate(
        TaskAntennaKL
          , (const portCHAR *) "TaskAntennaKM"
          , 128  // Stack size
          , NULL
          , 1  // Priority
          , NULL );

        xTaskCreate(
        TaskAntennaMN
          , (const portCHAR *) "TaskAntennaLN"
          , 128  // Stack size
          , NULL
          , 1  // Priority
          , NULL );

        if ( xSerialSemph == NULL ){

                xSerialSemph = xSemaphoreCreateMutex();
                if ( ( xSerialSemph ) != NULL )
                xSemaphoreGive( ( xSerialSemph ) );
        }

        // Now the task scheduler, which takes over control of scheduling individual tasks, is automatically started.

} // End setup

void loop()
{
  // Empty. Things are done in Tasks.
}

/*==================================================*/
/*      TASKS           */
/*==================================================*/

/*--------------------------------------------------*/
/*      Antenna Receiver            */
/*--------------------------------------------------*/
void TaskAntennaReceived(void *pvParameters){
        /*
         * This task is to periodically received messages from the antennas.
         * A semaphore is used because the serial communication is alternately occupied by XBee and ArduMIDI.
         *
         * When a message is received, the source will be checked then the parsed message will be collected.
         * After it parsed, it will be sent to respective "antenna duty"
         *
         */
        (void) pvParameters;

        for(;;){

                if ( xSemaphoreTake( xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

                        xbee.readPacket();

                        // check if a packet was received:
                        if (xbee.getResponse().isAvailable()) {

                                if (xbee.getResponse().getApiId() == ZB_RX_RESPONSE) {
                                         xbee.getResponse().getZBRxResponse(ZbRx);

                                }
                                remoteAddr = ZbRx.getRemoteAddress64().getLsb();

                                switch (remoteAddr) {

                                        // If there is data from Antenna K
//                                      case 0x5B45:
                                        case 0x5B42:    // if we want to activate acceleration antenna, please comment this line
                                                // store 2 8-bit data to 1 16-bit data
                                                distanceK = (ZbRx.getData(0) * 256) + ZbRx.getData(1);
                                                flagAntennaK = true;
                                                break;

                                        // If there is data from Antenna L
                                        case 0x5AB0:
                                                // store 2 8-bit data to 1 16-bit data
                                                distanceL = (ZbRx.getData(0) * 256) + ZbRx.getData(1);
                                                flagAntennaL = true;
                                                break;

                                        // If there is data from Antenna M
                                        case 0x5B4F:
                                                // store 2 8-bit data to 1 16-bit data
                                                distanceM = (ZbRx.getData(0) * 256) + ZbRx.getData(1);
                                                flagAntennaM = true;
                                                break;

                                        // If there is data from Antenna N
                                        case 0x5B4B:
                                                // store 2 8-bit data to 1 16-bit data
                                                distanceN = (ZbRx.getData(0) * 256) + ZbRx.getData(1);
//                                              Serial.println(distanceN);
                                                flagAntennaN = true;
                                                break;


                                } // End switch case
                        }       // End if data available
                        xSemaphoreGive( xSerialSemph );
                }       // End if semaphore

                // Read analog value from potentiometer
                maxDistance = analogRead(POT_PIN);
                maxDistance = map(maxDistance, 0, 1023, 50, 400);

                vTaskDelay(200/portTICK_PERIOD_MS);
        } // End For loop
} // End task
/*--------------------------------------------------*/

/*--------------------------------------------------*/
/*      Task Antenna K and  L           */
/*--------------------------------------------------*/

void TaskAntennaKL(void *pvParameters){
        /*
        * This task is to manage the message from antenna K, L, and Accelerometer.
        *
        * There are 3 subroutines:
        * 1. Accelerometer. Manage movement data from accelerometer.
        * 2. Antenna K. Manage distance data from antenna K
        * 3. Antenna L. Manage the movement detection which is sent from antenna L
        *
        */

        (void) pvParameters;

        // Antenna K variables
        uint8_t targetValKLo = 60;
        uint8_t targetValKHi = 95;

        // Antenna L variables
        uint8_t targetValLLo = 50;
        uint8_t targetValLHi = 100;
        uint8_t localNoteL = 0;


        for(;;){
                switchState = digitalRead(SWITCH);

                /*
                   * ****************************************************
                   * Antenna Acc Sub-subroutine
                   * ****************************************************
                   */

                  if (flagAntennaAcc){
                          digitalWrite(LED_ACC, HIGH);
                                  if(messageAcc == 'b') {

                                        if ( xSemaphoreTake( xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

```

```
                      midi.noteOn(CH3, beatNote, beatVel);
                      delay(delayNoteOn);
                      midi.noteOff(CH3, beatNote);
                      delay(delayNoteOff);

                      xSemaphoreGive( xSerialSemph );
                  }

               }
               else if(messageAcc == 'c'){
                  if ( xSemaphoreTake( xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

                      midi.noteOn(CH3, beatNote, beatVel);
                      delay(delayNoteOn);
                      midi.noteOff(CH3, beatNote);
                      delay(delayNoteOff);

                      xSemaphoreGive( xSerialSemph );
                   }
               }
               flagAntennaAcc = false;
               digitalWrite(LED_ACC, LOW);
           } // End if


       /*
        *
 **************************************************
        * Antenna L Sub-subroutine
        *
 **************************************************
        */

       else if (flagAntennaL) {
           digitalWrite(LED_L, HIGH);
           localNoteL = map(distanceK, 1, maxDistance, targetValLHi, targetValLLo);

           if (!switchState){
               digitalWrite(LED_MIDI_ON, HIGH);

              // Left panning handling
              if (distanceL == 800) {

                  if ( xSemaphoreTake( xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

                      pitchBend(CH1, localNoteL, -5000);
```

```

                                                xSemaphoreGive( xSerialSemph );
                                        }


                                }
                                // Center panning handling
                                else if (distanceL == 900) {

                                        if ( xSemaphoreTake( xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

                                                pitchBend(CH1, localNoteL, 0);

                                                xSemaphoreGive( xSerialSemph );
                                        }

                                }
                                // Right panning handling
                                else if (distanceL == 1000) {

                                        if ( xSemaphoreTake( xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

                                                pitchBend(CH1, localNoteL, 5000);

                                                xSemaphoreGive( xSerialSemph );
                                        }


                                }

                        } // If the switch is OFF, it will send "RAW" data (distance measurement) to computer.
                        else if (switchState){
                                digitalWrite(LED_MIDI_ON, LOW); // give signal we are NOT in MIDI mode

                                if (distanceL <= maxDistance) {
                                        if ( xSemaphoreTake( xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

                                                Serial.print("L"); Serial.println(distanceL);

                                                xSemaphoreGive( xSerialSemph );
                                        }
                                }
                                else if (distanceL > maxDistance){
```

```
                                            if ( xSemaphoreTake(
xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){
                                                    if (distanceL
== 800) Serial.println("L_left");
                                                    else if (
distanceL == 900) Serial.println("L_center");
                                                    else if (
distanceL == 1000) Serial.println("L_right");
                                                    xSemaphoreGive(
 xSerialSemph );
                                            }
                                    }

                      }
                      flagAntennaL = false;
                      digitalWrite(LED_L, LOW);
              } // End if L

      /*
       * ***************************************************
       * Antenna K Sub-subroutine
       * ***************************************************
       */

      if (flagAntennaK){
              digitalWrite(LED_K, HIGH);
                      // If the switch is ON, it will send MIDI
signal to computer.
                      // Because of INPUT_PULLUP, the state is
inverted
                      if (!switchState){
                              digitalWrite(LED_MIDI_ON, HIGH);

                                    if (distanceK <= maxDistance){
                                            noteK = map(distanceK,
1, maxDistance, targetValKHi, targetValKLo);

                                            if ( xSemaphoreTake(
xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

                                                    midi.noteOn(CH1
, noteK, velocity);
                                                    byte tempNoteK
= noteK;
                                                    delay(
delayNoteOn);
                                                    midi.noteOff(
CH1, tempNoteK);
                                                    delay(
delayNoteOff);

                                                    xSemaphoreGive(
 xSerialSemph );
                                            }

                                    }
```

```
                    else if (distanceK > maxDistance){

                        if ( xSemaphoreTake( xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

                         midi.controlChange(CH1, 123, 0);
                         xSemaphoreGive( xSerialSemph );
                        }

                    }
             } // End if switch
             // If the switch is OFF, it will send RAW data (distance measurement) to computer.
             else if (switchState){
                    digitalWrite(LED_MIDI_ON, LOW);

                if ( xSemaphoreTake( xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

                    Serial.print("K"); Serial.println(distanceK);

                    xSemaphoreGive( xSerialSemph );
                }

             }
         flagAntennaK = false;
         digitalWrite(LED_K, LOW);
     } // End if K

    } // End for loop

} // End Task

/*--------------------------------------------------*/
/*      Task Antenna M and N        */
/*--------------------------------------------------*/

void TaskAntennaMN(void *pvParameters){
    /*
    * This task is to manage the message from antenna M, N, and Accelerometer.
    *
    * There are 3 subroutines:
    * 1. Accelerometer. Manage movement data from accelerometer.
    * 2. Antenna M. Manage distance data from antenna M.
    * 3. Antenna N. Manage the movement detection which is sent from antenna N.
    *
    */

    (void) pvParameters;

    // Antenna M variables
    uint8_t targetValMLo = 60;
    uint8_t targetValMHi = 95;

    // Antenna N variables
    uint8_t targetValNLo = 50;
    uint8_t targetValNHi = 100;
    uint8_t localNoteN = 0;

    for(;;){
            switchState = digitalRead(SWITCH);

              /*
               *
***************************************************
               * Antenna Acc Sub-subroutine
               *
***************************************************
               */

              if (flagAntennaAcc){
                      digitalWrite(LED_ACC, HIGH);
                      if(messageAcc == 'b') {

                            if ( xSemaphoreTake( xSerialSemph, (
TickType_t ) 5 ) == pdTRUE ){

                              midi.noteOn(CH3, beatNote, beatVel);
                              delay(delayNoteOn);
                              midi.noteOff(CH3, beatNote);
                              delay(delayNoteOff);

                              xSemaphoreGive( xSerialSemph );
                            }

                      }
                      else if(messageAcc == 'c'){

                              if ( xSemaphoreTake( xSerialSemph, (
TickType_t ) 5 ) == pdTRUE ){

                                      midi.noteOn(CH3, beatNote,
beatVel);
                                      delay(delayNoteOn);
                                      midi.noteOff(CH3, beatNote);
                                      delay(delayNoteOff);

                                      xSemaphoreGive( xSerialSemph
);
                              }
                      }
                      flagAntennaAcc = false;
                      digitalWrite(LED_ACC, LOW);
              } // End if

        /*
```

```
         * ************************************************
         * Antenna N Sub-subroutine
         * ************************************************
         */
        else if (flagAntennaN){
                digitalWrite(LED_N, HIGH);
                localNoteN = map(distanceM, 1, maxDistance, targetValNHi, targetValNLo);

                // If the switch is ON, it will send MIDI signal to computer.
                if (!switchState){
                        digitalWrite(LED_MIDI_ON, HIGH);

                        // Left panning handling
                        if (distanceN == 800) {

                                if ( xSemaphoreTake( xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

                                        pitchBend(CH2, localNoteN, -5000);

                                        xSemaphoreGive( xSerialSemph );
                                }

                        }
                        // Center panning handling
                        else if (distanceN == 900) {

                                if ( xSemaphoreTake( xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

                                        pitchBend(CH2, localNoteN, 0);

                                        xSemaphoreGive( xSerialSemph );
                                }

                        }
                        // Right panning handling
                        else if (distanceN == 1000) {

                                if ( xSemaphoreTake( xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

                                        pitchBend(CH2, localNoteN, 5000);

                                        xSemaphoreGive( xSerialSemph );
                                }
                        }
                }     // End if switch
                // If the switch is OFF, it will send RAW data (
distance measurement) to computer.
                else if (switchState){
                        digitalWrite(LED_MIDI_ON, LOW); // give
signal we are NOT in MIDI mode

                        if (distanceN <= maxDistance) {
                                if ( xSemaphoreTake(
xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

                                        Serial.print("N");
Serial.println(distanceN);

                                        xSemaphoreGive(
xSerialSemph );
                                }
                        }
                        else if (distanceK > maxDistance){
                                if ( xSemaphoreTake(
xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

                                        if (distanceN == 800)
Serial.println("N_left");
                                        else if (distanceN ==
900) Serial.println("N_center");
                                        else if (distanceN ==
1000) Serial.println("N_right");

                                        xSemaphoreGive(
xSerialSemph );
                                }
                        }
                }
                flagAntennaN = false;
                digitalWrite(LED_N, LOW);
        } // End if N
        /*-----------------------------------------------------*/

        /*
         * ***************************************************
         * Antenna M Sub-subroutine
         * ***************************************************
         */
        else if (flagAntennaM){
                digitalWrite(LED_M, HIGH);
                if (!switchState){
                        digitalWrite(LED_MIDI_ON, HIGH);

                            if (distanceM <= maxDistance){
                              noteM = map(distanceM, 1, maxDistance
, targetValMHi, targetValMLo);

                              if ( xSemaphoreTake( xSerialSemph, (
TickType_t ) 5 ) == pdTRUE ){

```

```
                                          midi.noteOn(CH2, noteM, velocity);
                                          byte tempNote = noteM;
                                          delay(delayNoteOn);
                                          midi.noteOff(CH2, tempNote);
                                          delay(delayNoteOff);

                                          xSemaphoreGive( xSerialSemph );
                                   }

                                }
                                else if (distanceM > maxDistance){

                                         if ( xSemaphoreTake( xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

                                                midi.controlChange(CH2, 123, 0);

                                                xSemaphoreGive( xSerialSemph );
                                         }

                                }

                  }     // End if switch
                  // If the switch is OFF, it will send RAW data (distance measurement) to computer.
                  else if (switchState){
                          digitalWrite(LED_MIDI_ON, LOW);

                          if ( xSemaphoreTake( xSerialSemph, ( TickType_t ) 5 ) == pdTRUE ){

                                  Serial.print("M"); Serial.println(distanceM);

                                  xSemaphoreGive( xSerialSemph );
                          }

                  }
                  flagAntennaM = false; // reset flag Antenna M
                  digitalWrite(LED_M, LOW);
          } // End If M
  } // End for
} // End task

/*=====================================================*/
/*       FUNCTIONS            */
/*=====================================================*/

/*
 * Function: flashLed()
 *
 * Just a function to blink a LED.
 *
 */
void flashLed(uint8_t pin, int wait) {
        digitalWrite(pin, HIGH);
        delay(wait);
        digitalWrite(pin, LOW);
        delay(wait);
}

/*
 * Function: pitchBend(uint8_t ch, uint8_t localNote, int bendVal)
 *
 * ch = channel (CH1, CH2, ..., CH16)
 * localNote = localNoteL (antenna L) or localNoteN (antenna N)
 * bendVal = number between -8000 and 8000
 *
 * Function to produce pitchbend message.
 *
 */
void pitchBend(uint8_t ch, uint8_t localNote, int bendVal){
        midi.noteOn(CH1, localNote , velocity);
        midi.pitchBendChange(CH1, bendVal);
        byte tempLocNote = localNote;
        delay(delayNoteOn);
        midi.noteOff(CH1, tempLocNote);
        delay(delayNoteOff);
}
```