POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

A Networked Modular Robot

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Supervisor:

PROF. ANDREA BONARINI

Master Graduation Thesis by:

HAMIDREZA HANAFI
Student Id n. 841408

Academic Year 2016-2017

ACKNOWLEDGMENTS

I would like to thank my supervisor, Professor Andrea Bonarini, for his help and guidance throughout my project, who make this project possible. Professor Bonarini gave me this opportunity to develop my own solutions and helped with his valuable feedback. Second, I want to thank my lab mates and faculty members who make this project a great and unforgettable experience.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LISTINGS

ABSTRACT

There are an estimated 150 million children worldwide living with a disability. Children affected by physical and cognitive disabilities may have difficulty participating in play activities. The use of robots can promote and facilitate playing for them.

This project is aimed to the development of a Networked Modular Robot which is able to play with disabled or normally developed children. The hardware components and modules of this robot are removable and the children are able to attach or de-attach these components in order to see what games the robot is still able to play. This makes it possible to adapt the robot to the specific child abilities and needs.

To achieve this goal and design the modules, different embedded networks have been analyzed and the processor with the desired network capabilities has been selected. A set of hardware components has been developed in order to interact with the modules. The distributed system is able to detect the presence of the removable modules or components and to reconfigure dynamically to use them. This robot can perform various games based on different configurations.

Keywords: Embedded Network, Robotic Modules, Microphones, Networked Modular Robot, Sound angle detection

SOMMARIO

Ci sono circa di 150 milioni di bambini in tutto il mondo che vivono con disabilità. Normalmente I bambini affetti da disabilità fisiche e cognitive fisiche hanno difficoltà a partecipare alle attività di gioco per vedere quali giochi il robot è ancora in grado di riprodurre. L'uso di robot può promuovere e facilitare il gioco per loro.

Questo progetto è rivolto a sviluppare di un "modular robot network" che è in grado di giocare con i bambini disabili o i bambini con una crescita normale. I componenti hardware e i moduli di questo robot sono rimovibili e i bambini sono in grado di collegare o disinstallare questi componenti. Ciò consente di adattare il robot alle abilità e alle esigenze specifiche dei bambini.

Per raggiungere questo obiettivo e progettare i moduli, sono state analizzate diversi reti inserito e il processore con le funzionalità di rete desiderate è stato selezionato. Un set di componenti hardware sviluppati per interagire con i moduli. Il sistema distribuito è in grado di rilevare la presenza dei moduli o dei componenti rimovibili e di riconfigurare dinamicamente per utilizzarli. Questo robot può eseguire vari giochi basati su diverse configurazioni.

Keywords: Rete Embedded, moduli robotici, microfoni, robot modulare in rete, rilevamento angolo sonoro

1 INTRODUCTION

Artificial intelligence is growing up fast, as are robots whose facial expressions can elicit empathy and make your mirror neurons quiver.

— Diane Ackerman

Robotics today is changing and growing rapidly. As more robots are doing people's job, the interaction between human and robots comes to the center of attention. Social robotics and human-robot interaction are growing research fields concerned with the questions how humans and robots can better live, work, and interact together [1].

There are an estimated 150 million children worldwide living with a disability. Children affected by physical and cognitive disabilities may have difficulty participating in play activities. The use of robots can promote and facilitate playing for them. Robot play could give insight to the developmental level of play of children with disabilities who are difficult to assess with standardized tests. Moreover, Robots can facilitate discovery and enhance opportunities for play, learning and cognitive development in children who have motor disabilities. [2].

The robot in this project is aiming to play with children with or without disabilities who, possibly supported by companions or caregivers, should be able to add or remove the hardware components to or from the robot in order to see what games the robot is still able to play. This makes it possible to adapt the robot to the specific child abilities and needs.

1.1 Problem statement

This thesis dealt with developing a low-cost modular robot which can perform different behaviors based on the connection of different module's combination. The aim of the robot is to play with kids especially ones with disabilities. The robot should decide autonomously what to do and what game to play.
The modules should perform a single task. The implemented modules

and components are: microphone, distance detector, speaker, hand-arm and LED lights.
The modules should be easy to plug or unplug as a kid will be able to decide what game to play. The following games would be interesting:

Game 1: Propose a rhythm through beeper, become happy if detected sounds (such as clap) have a similar rhythm, sad and encouraging otherwise. Modules: loudspeaker, microphone, either arm or movement (for being happy)

Game 2: Same as the game one, but with LEDs. Modules: LEDs, microphone, either arm or movement (for being happy, but this may be done also only with LEDs or beeper)

Game 3: Positioning the distance sensors appropriately, one can throw a ball and the arm can send it back. Modules: arm, distance sensors, (possibly LEDs and beeper)

Game 4: Robot moves randomly avoiding obstacles, a player can put obstacles or manage to move a stick to make it reaching a target. Modules: distance sensors, mobility (optional: LEDs and/or loudspeaker for reactions)

Moreover, It should be easy to program the robot and implement new games or modules.

1.2 Assumptions

The robot only works in the following conditions:

- The robot will only move on a planar ground.
- The robot will only work on an indoor environment where the noises are not high.
- The robot will only detect sound angle if all four microphones are connected.

1.3 Outline

This thesis consists of six chapters.

- In chapter 2, we well describe the related background knowledge.

- In chapter 3, we explain the reasons for choosing each hardware components.
- Throughout chapter 4, we introduce the various features and software implementations in the robot modules.
- In chapter 5, we describe the module architecture and the way they communicate, synchronize and decide together.
- Finally, we provide the conclusion and possible future works in chapter 6.

2

BACKGROUND KNOWLEDGE

In this chapter, we are going to review some related fundamental background knowledge.

- First, we have a brief description about embedded networks and its famous standards as a set of alternatives for the module's network in order to have a better comparison.
- Then, we describe the Multimedia in embedded systems and its hardware/software requirements.
- Moreover, we review technical details of the board which is used in this project.
- Finally, we describe the tools and libraries which are important for this project.

2.1 Embedded Networks

As embedded systems are becoming more and more complex, the knowledge about various disciplines like data processing, electronics, telecommunications, and networks becomes mandatory for all. Nowadays, "Network" plays a prominent role in embedded systems. A proper understanding of networks is also equally important.

The embedded system was originally designed to work on a single device. However, in the current scenario, implementation of different networking options has increased the overall performance of the embedded system in terms of economy as well as technical considerations.
The most efficient types of networks used in the embedded system are Bus network and Ethernet network.

A Bus is used to connect different network devices and to transfer a huge range of data, for example, serial bus, I2C[1] bus, CAN[2] bus, etc. The Ethernet type network works with the TCP/IP protocol [3].

1 Inter-Integrated Circuit
2 Controller Area Network

Serial Communication is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus. This is in contrast to parallel communication, where several bits are sent as a whole, on a link with several parallel channels.
Examples of serial embedded networking include CAN, I2C, Component, RS232, and RS485 bus networking.

2.1.1 RS-232

RS-232 is a standard for serial communication transmission of data. It is commonly used in computer serial ports. The standard defines the electrical characteristics and timing of signals, the meaning of signals, and the physical size and pin-out of connectors.

RS-232 can support data rates of up to 920 kbps (normally 9600 and 115.2K are the maximum rates) and is commonly found in 9 or 25 pin configurations, however, only three pins are required. Most applications drop many of the less commonly used pins, though some configurations such as a data modem connect every pin for full handshaking capabilities. An RS-232 is a point to point connection made between a Data Terminal Equipment (DTE) device and a Data Communications Equipment (DCE) device. RS-232 has a maximum cable length of 50 Ft at 9600 baud. [4].

Advantages of RS-232:

- Simple wiring and connectors
- Widely available
- Low cost
- Most embedded processors include this interface
- Software to implement a serial port is easy

Disadvantages of RS-232:

- Only point to point
- Incompatibilities in wiring and configuration between devices
- Short cable lengths

- Subject to noise interference
- Low data rates
- Many different software protocols

2.1.2 RS-422

RS422 is a high speed and/or long distance data transmission. Each signal is carried by a pair of wires and is thus a differential data transmission system.

RS-422 offers one of the fastest serial data rates at 10M bps RS-422 is a multi-drop configuration, allowing for up to 10 unit loads. Its use of voltage differences makes it ideal for noisy environments. RS-422 can support cable lengths of up to 4000 feet, however, its data rate lessens as distance increases [5].

Advantages of RS-422:

- High data rates
- Less subject to noise
- Longer cable lengths

Disadvantages of RS-422:

- Only point to point
- Not as commonly used
- Unidirectional
- For most applications, only one transmitter is used

2.1.3 RS-485

RS-485, also known as TIA-485(-A), EIA-485, is a standard defining the electrical characteristics of drivers and receivers for use in serial communications systems. Electrical signaling is balanced, and multi-point systems are supported.

RS-485 provides similar speed advantages as RS-422(2.1.2), allowing for data rates up to 10M bps It differs in that it is a multi-point

configuration, allowing for support of multiple drivers and multiple receivers. RS-485 can support up to 32 unit loads due to its bi-directional interface [6].

Advantages of RS-485:

- Low cost
- Immune to noise
- Multipoint applications
- Operates on a single pair of wires

Disadvantages of RS-485:

- Not as commonly used
- Less standardized connectors and terminology
- Half-duplex master-slave operation

2.1.4 I2C

I²C (Inter-Integrated Circuit), pronounced I-squared-C or I-two-C, is a multi-master, multi-slave, packet switched, single-ended, serial computer bus invented by Philips Semiconductor (now NXP Semiconductors) [7].

Advantages of I2C:

- Maintains low pin/signal count even with numerous devices on the bus
- Adapts to the needs of different slave devices
- Readily supports multiple masters
- Incorporates ACK/NACK functionality for improved error handling

Disadvantages of I2C:

- Increases the complexity of firmware or low-level hardware

- Imposes protocol overhead that reduces throughput
- requires pull-up resistors, which
  - Limits clock speed
  - Consumes valuable PCB real estate in extremely space-constrained systems
  - Increases power dissipation

2.1.5 CAN

A Controller Area Network (CAN-bus) is a robust vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. It is a message-based protocol, designed originally for multiplex electrical wiring within automobiles, but is also used in many other contexts [8].

Advantages of CAN:

- It supports multi-master and multi-cast features.
- The CAN bus has the maximum length of 40 meters.
- The CAN provides the ability to work in a different electrical environment.
- The controller area network (CAN) reduces wiring since it is a distributed control and this ensures enhancing the system performance.
- It has single serial bidirectional line to achieve half duplex communication.
- It has a standard bus in distributed network.
- It costs low and it is a lightweight network.
- It has automatic retransmission for the message that lost attribution.

Disadvantages of CAN:

- It has high software expenditure.
- Undesirable interaction is more probable.

2.1.6 Ethernet

Ethernet is a network protocol that controls how data is transmitted over a LAN. Technically it is referred to as the IEEE 802.3 protocol. The protocol has evolved and improved over time and can now deliver at the speed of a G-bit per second.

In recent years usage of Ethernet in embedded applications have been increased but the heavy TCP/IP protocol is really hard to be implemented for microcontrollers with low SRAM[3] and flash memory.

Advantages of Ethernet:

- It supports multi-points.
- Low cost without consideration of processor.
- Easy to use in high-level programming languages.
- Fault tolerant.

Disadvantages of Ethernet:

- Heavy TCP/IP protocol for microcontrollers.
- Hard to use with low-level programming languages.
- Ethernet cable has more wires.
- It needs a hub for multi-point.

2.2 Sensors

infrared sensor An Infrared sensor is an electronic device, that emits infrared radiation in order to sense some aspects of the surroundings. An IR sensor can measure the heat of an object as well as detects the motion.These types of sensors measure only infrared radiation, rather than emitting it that is called as a passive IR sensor.

microphone A microphone, colloquially nicknamed mic or mike is a transducer that converts sound into an electrical signal.

3 Static random-access memory

2.3 Multimedia

2.3.1 Recording Audio

If you want to record an external audio from a musical instrument or simple voice in an embedded project, you need to have a microphone connected to one of the Analog to Digital converter (ADC) ports of your microcontroller.

In above case, you will be able to record and save the audio in WAV format but you should create correct WAV file header.

2.3.2 Audio Playback

Sometimes you have an embedded project that needs to play audio. Maybe you just need to make a beep, in which case a simple speaker and a square wave will work, but other times you'll need to play actual audio, like voice or music. In that case, you need to use a Digital to Analog Converter (DAC) to generate the stored waveforms.

If the Audio source is not encoded like WAV files, we can use a library in Arduino which is called TMRpcm. On the other hand, if your Audio source is encoded with a format like MP3, we should first decode the file and then we will be able to play it.

In order to use MP3 audio format, we should use an external decoder which is specially designed to do the job like VS1053 from VLSI company. In the following, we will see some of the libraries and boards for this purpose.

2.3.3 TMRpcm Library

Arduino library for asynchronous playback of PCM/WAV files directly from SD card. This library can be found at https://github.com/TMRh20/TMRpcm.

Features

- PCM/WAV playback direct from SD card
- Main formats: WAV files, 8-bit, 8-32khz Sample Rate, mono. See the wiki for other options.

- Asynchronous Playback: Allows code in the main loop to run while audio playback occurs.
- Single timer operation: TIMER1 (Uno,Mega) or TIMER3,4 or 5 (Mega)
- Complimentary output or dual speakers
- 2x Oversampling

2.3.4 VS1053

VS1053 is a versatile "MP3 decoder chip" belonging to VLSI Solution's extensive slave audio processor family. In addition to being able to decode all the most common audio formats - including the advanced features of newer AAC files - functionality of this IC can be greatly expanded just by loading a bit of extra software to its RAM memory. In addition to being able to decode all major audio formats, VS1053 is capable of recording in three different audio formats, from lossless 16-bit PCM to highly compressed, yet high-quality Ogg Vorbis files. For the best headphone listening experience, the VS1053 includes EarSpeaker spatial processing which accurately simulates how a room with stereo loudspeakers would sound. This option can naturally be turned off when required. All in all, VS1053 is an easy-to-use, powerful workhorse for audio playback and recording applications [9].

Features

- Decodes multiple formats
  - Ogg Vorbis
  - MP3 = MPEG 1 & 2 audio layer III (CBR+VBR+ABR)
  - MP1 & MP2 = MPEG 1 & 2 audio layers I & II optional
  - MPEG4 / 2 AAC-LC(+PNS), HE-AAC v2 (Level 3) (SBR + PS)
  - WMA4.0/4.1/7/8/9 all profiles (5-384 kbps)
  - FLAC lossless audio with software plugin (upto 24 bits, 48 kHz)
  - WAV (PCM + IMA ADPCM)
  - General MIDI 1 / SP-MIDI format 0

- Encodes three different formats from mic/line in mono or stereo
  - Ogg Vorbis with software plugin
  - IMA ADPCM
  - 16-bit PCM
- Encodes three different formats from mic/line in mono or stereo
- Ogg Vorbis with software plugin
- IMA ADPCM
- 16-bit PCM
- Streaming support
- EarSpeaker Spatial Processing
- Bass and treble controls
- Operates with a single 12-13 MHz or 24-26 MHz clock
- Internal PLL clock multiplier
- Low-power operation
- High-quality on-chip stereo DAC with no phase error
- between channels
- Zero-cross detection for smooth volume change
- Stereo earphone driver capable of driving a 30- ohm load
- Quiet power-on and power-off

2.3.5 Amplifier

An amplifier, electronic amplifier or (informally) amp is an electronic device that can increase the power of a signal (a time-varying voltage or current). An amplifier uses electric power from a power supply to increase the amplitude of a signal. The amount of amplification provided by an amplifier is measured by its gain: the ratio of output to input. An amplifier is a circuit that can give a power gain greater than one.

2.4 Olimex STM32

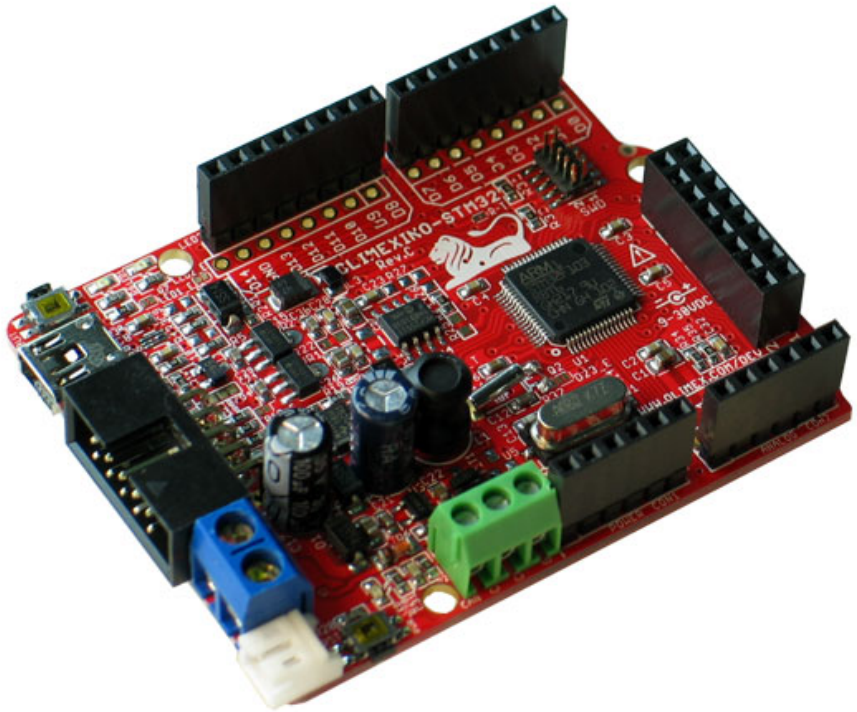

Figure 2.1: Olimex STM32 Board

2.4.1 STM32F103RBT6 microcontroller

- ARM® 32-bit Cortex®-M3 CPU Core
  - 72 MHz maximum frequency, 1.25 DMIPS/MHz (Dhrystone 2.1)
  - Performance at 0 wait state memory access
  - Single-cycle multiplication and hardware division
- Memories
  - 64 or 128 Kbytes of Flash memory
  - 20 Kbytes of SRAM division
- Up to 80 fast I/O ports
  - 26/37/51/80 I/Os, all mappable on 16 external interrupt vectors and almost all 5 V-tolerant

2.4.2 Connectivity

- CAN with driver
- USB
- RS-232

2.4.3 Power

OLIMEXINO-STM32 can be powered from:

- External power supply (9-30) VDC.
- + 5V from USB
- 3.7 V Li-ion battery
- JTAG/SWD programmer/debugger

The programmed board power consumption is about 50 mA with all peripherals enabled.

2.4.4 Storage

micoSD-card for data logging

2.4.5 Working temperature

Carefully selected, all of the components work reliably in the INDUSTRIAL temperature range -25+85C

2.4.6 Dimensions

PCB dimensions: (2.7 x 2.1)" (6.9 x 5.3)cm

2.4.7 Other Information

Further information can be obtained from the product website at: https://www.olimex.com/Products/Duino/STM32/OLIMEXINO-STM32/

2.5 Arduino

Arduino is an open-source electronics prototyping platform, designed to make the process of using electronics in multidisciplinary projects easily accessible. The hardware consists of a simple open hardware design for the Arduino board with an Atmel AVR processor and on-board I/O support. The software consists of a standard programming language and the boot loader that runs on the board.

Arduino hardware is programmed using a Wiring-based language (syntax + libraries), similar to C++ with some simplifications and modifications, and a Processing-based Integrated Development Environment (IDE). The project began in Ivrea, Italy in 2005 aiming to make a device for controlling student-built interaction design projects less expensively than other prototyping systems available at the time. As of February 2010 more than 120,000 Arduino boards had been shipped. Founders Massimo Banzi and David Cuartielles named the project after a local bar named “Arduino”. The name is an Italian masculine first name, meaning "strong friend". The English pronunciation is "Hardwin", a namesake of Arduino of Ivrea. More information could be found at the creator’s web page http://arduino.cc/ and in the Arduino Wiki http://en.wikipedia.org/wiki/Arduino

2.6 STM32duino

The STM32 boards are not originally supported by Arduino. In order to make the Arduino support these board, we should install an extension which will add all library and support files for STM32 boards.

STM32duino is an Arduino Core for the Maple Mini and other STM32 F1 and F4 boards, continuing where Leaflabs left off. More information about the STM32duino can be found at http://wiki.stm32duino.com/index.php?title=Main_Page

2.7 Cocox OS (RTOS)

Cocox OS and in short term CoOS is a Real-Time Operating System (RTOS). Like a normal operating system, CoOS works with tasks. Because this microcontroller has only one core, it can only execute one task a time but it uses software interrupts to quickly change between tasks. A software interrupt works on the same principal as a hardware

interrupt, it can jump to another subroutine and run that code first. This is what an RTOS does, a number of tasks created and executed and because of the fast switching between tasks the Simplecortex is basically multitasking and can run multiple tasks at once without the programmer worrying about it.[10]

With CoOS it is possible to manage which subroutine has to be executed first. Each task can be given a priority from 0 to 10. The task with the highest priority will be executed first. It is also possible to give a task temporarily the highest priority, this is called a Mutex.

Scheduling is needed to start the operating system, scheduling means that there is a plan which specifies what tasks need to be executed in what order and how fast. When CoOS is started CoOS does this automatically.

Flags are used to communicate between tasks. CoOS has a maximum of 32 flags.

Why use CoOS? Normally a microcontroller will execute a certain piece of code sentence by sentence. CoOS can stop in the middle of a subroutine and continue at another subroutine. In this way, for example it is possible to send data to an LCD and run an Ethernet webserver at the same time.

Advantages:

- Multitasking

Disadvantages:

- CoOS uses more memory, about 5K flash and 2K RAM.
- It is not possible to use SLEEP mode because the software interrupts will wake up the microcontroller immediately.

Table 2.1 shows the minimum and maximum timings that are needed to run CoOS.

Table 2.1: CoOS minimum and maximum timings

Create defined task, no task switch	5.3us / 5.8us
Create defined task, switch task	7.5us / 8.6us
Delete task (ExitTask)	4.8us / 5.2us
Task switch (SwitchContext)	1.5us / 1.5us
Task switch (upon set flag)	7.5us / 8.1us
Task switch (upon sent semaphore)	6.3us / 7.0us
Task switch (upon sent mail)	6.1us / 7.1us
Task switch (upon sent queue)	7.0us / 7.6us
Set Flag (no task switch)	1.3us / 1.3us
Send semaphore (no task switch)	1.6us / 1.6us
Send mail (no task switch)	1.5us / 1.5us
Send queue (no task switch)	1.8us / 1.8us

3

HARDWARE SETUP

In this Chapter, we explain the hardware setup like network, processor, media board and other hardware components. More importantly, we would describe the main reasons to choose each of them. The design of mechanical connectors is at the end of this chapter.

3.1 Robot platform

Wheeled robots are robots that navigate around the ground using motorized wheels to propel themselves. This design is simpler than using treads or legs and by using wheels they are easier to design, build, and program for movement in flat, not-so-rugged terrain. They are also better controlled than other types of robots. Disadvantages of wheeled robots are that they can not navigate well over obstacles, such as rocky terrain, sharp declines, or areas with low friction. Wheeled robots are most popular among the consumer market, their differential steering provides low cost and simplicity. Robots can have any number of wheels, but three wheels are sufficient for static and dynamic balance. Additional wheels can add to balance; however, additional mechanisms will be required to keep all the wheels on the ground, when the terrain is not flat [11].

For this project a 4 wheel robot is considered. The main reason for this choice is being easy to use and cheap.

Figure 3.1: Sample of 4 wheel robot platform

3.1.1 2-by-2 powered wheels for tank-like movement

This kind of robot uses 2 pairs of powered wheels. Each pair (connected by a line) turn in the same direction. The tricky part of this kind of propulsion is getting all the wheels to turn with the same speed. If the wheels in a pair aren't running with the same speed, the slower one will slip (inefficient). If the pairs don't run at the same speed the robot won't be able to drive straight. A good design will have to incorporate some form of car-like steering [11]. The scheme is available in fig 3.2

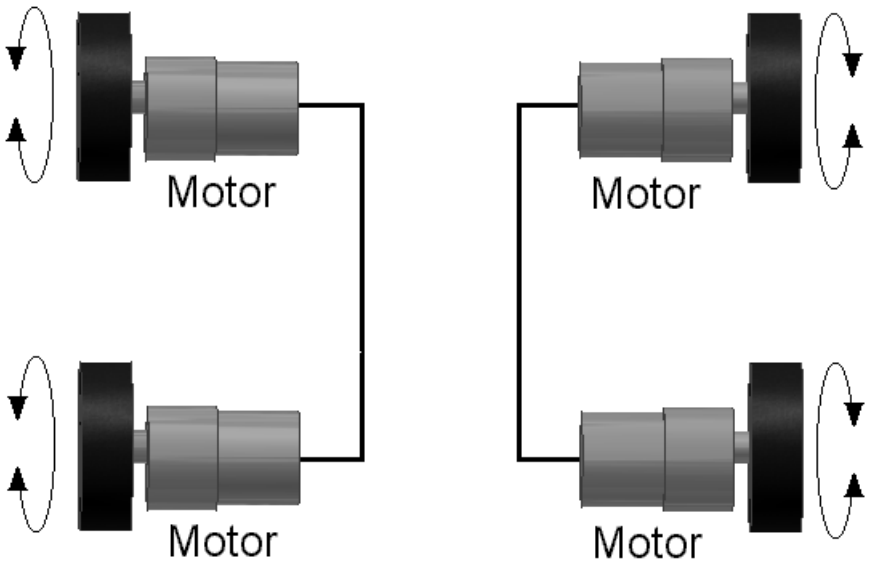

Figure 3.2: 4 Wheel 4 Drive

3.2 Choosing network

The main network requirements for the robot are as follows:

- Good speed
- Low cost
- Multi-point configuration
- Less wires
- Easy to use

For these requirements and from the network solutions reviewed in section 2.1, I found out that two of them can be used:

- RS485 2.1.3
- CAN 2.1.5

3.2.1 Network Topology of RS485

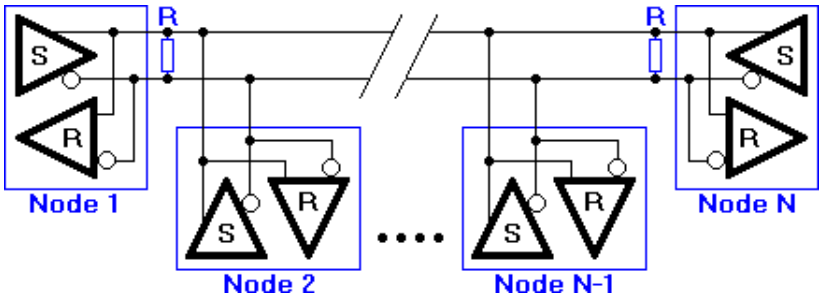

Figure 3.3: Network topology of RS485

In Fig 3.3 above, the general network topology of RS485 is shown. N nodes are connected in a multi-point RS485 network. For higher speeds and longer lines, the termination resistances are necessary on both ends of the line to eliminate reflections. Use 100 Ω resistors on both ends. The RS485 network must be designed as one line with multiple drops, not as a star [12].

3.2.2 RS485 functionality

Default, all the senders on the RS485 bus are in tri-state with high impedance. In most higher level protocols, one of the nodes is defined as a master which sends queries or commands over the RS485 bus. All other nodes receive these data. Depending on the information in the sent data, zero or more nodes on the line respond to the master. In this situation, bandwidth can be used for almost 100%. There are other implementations of RS485 networks where every node can start a data session on its own. This is comparable to the way Ethernet networks function. Because there is a chance of data collision with this implementation, theory tells us that in this case only 37% of the bandwidth will be effectively used. With such an implementation of a RS485 network it is necessary that there is error detection implemented in the higher level protocol to detect the data corruption and resend the information at a later time [12].

3.2.3 Network Topology of CAN

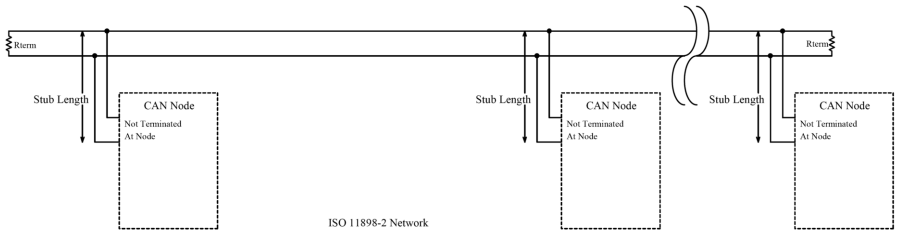

Figure 3.4: CAN network topology

Same as RS485, in the general CAN network topology, each node connected by two wire bus name CAN High and CAN Low. Use 120Ω resistors on both ends. Other topologies like "star" are allowed 3.4.

3.2.4 CAN functionality

Each node is able to send and receive messages, but not simultaneously. CAN data transmission uses a lossless bitwise arbitration method of contention resolution. This arbitration method requires all nodes on the CAN network to be synchronized to sample every bit on the CAN network at the same time. This is why some call CAN synchronous. Unfortunately the term synchronous is imprecise since the data is transmitted without a clock signal in an asynchronous format. [8].

3.2.5 Winner Network - CAN

Since there is no need to have a master node in CAN network and therefore no implementation of the transmission mechanism is needed for the functionality, we choose CAN as the network of the modules.

3.3 Choosing board

The main hardware board requirements are as follows:

- High Processing Speed
- Low cost

- CAN bus equipped
- Good SRAM
- Easy to use
- Available support libraries

Because we need higher processing speed since the modules have real-time requirements, AVR microcontrollers were not good options. STM32 microcontrollers which have more than 72MHz clock rate would satisfy the mentioned requirement.

The second factor was the network. As I chose before that CAN would be the network of the modules, One of the STM32 microcontrollers which already support this kind of peripherals should be selected.

The third factor was the price. Most of the STM32 microcontrollers have low-price but they have expensive development boards which were in contradiction with my goal to keep the prices low.

Based on all factors, I chose the OLIMEXINO-STM32 board which was mentioned in section 2.4. This board uses STM32F103RBT6 microcontroller which support CAN network. The network driver is already equipped with the board. Most importantly, the price of the board is less than 20 Euro which make it suitable for my application.

At last, I should mention that the board is based on maple design which is fully compatible with Arduino 2.5 and STM32duino 2.6 extension.

3.4 Choosing media board

The main media board requirements are as follows:

- Sound playback(MP3 Preferred)
- Sound recording
- Low cost
- Easy to use
- Available support libraries

From the options which were mentioned in section 2.3, I couldn't use the TMRpcm Library 2.3.3. The main reason was that the processor which I selected in the previous section, doesn't have Digital to Analog(DAC) support. DAC is the main requirement for this library since the sound data should be converted to an analog signal and transmitted to the speakers.

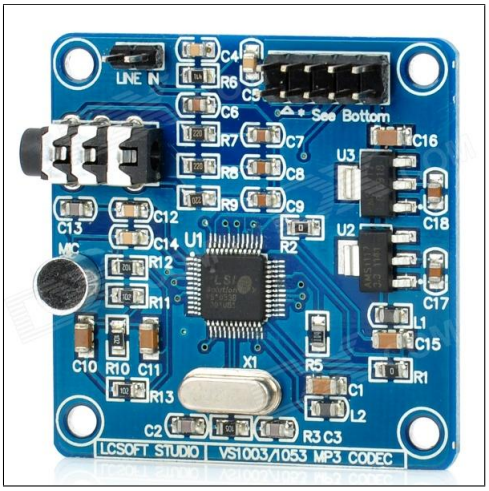

Figure 3.5: VS1053 board

On the other hand, Media board VS1053 2.3.4 from VLSI company was a good option. It supports MP3 playback, Ogg and Wave recording. The support libraries are available and finally, the price is as low as 6 Euro per unit which makes it an ideal selection.

3.5 Mechanical connectors

Since we should be able to remove the components from the main platform, A mechanical connector is needed.

The main requirements for the mechanical connectors are as follows:

- 3 or 4 wires
- Strong connection
- Easy to connect and disconnect
- Low cost

Based on the requirements and various connectors which are available in the market, I selected the 3.5mm audio jack which is available

in male and female format. One would be fixed on the main platform and the other would be on the module.

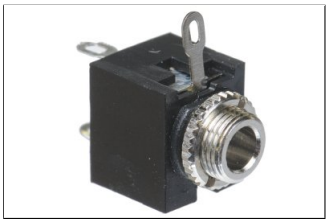

Figure 3.6: 3.5mm Panel Mount

Figure 3.7: 3.5mm Male Jack

The in work result of the connector shown in the below image.

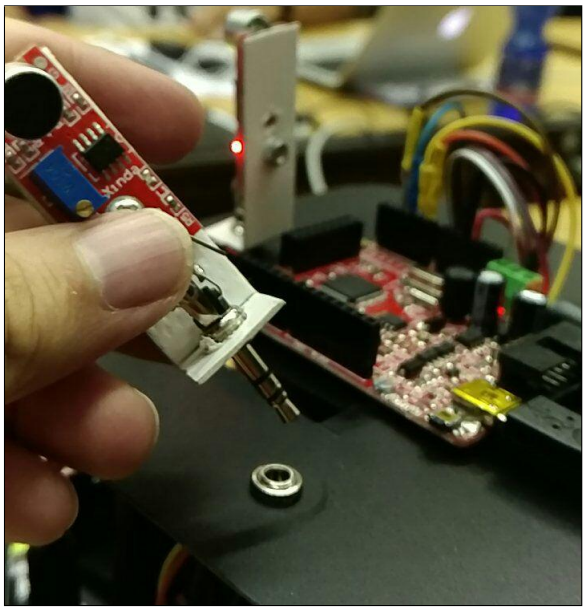

Figure 3.8: Connector in robot

4

COMPONENTS SOFTWARE IMPLEMENTATION

In this chapter, we describe components software implementations which we used independently in each module. We will describe the following items in this chapter:

- How to use the sensors.
- How to configure the CAN network and use it.
- The methods and algorithms used to detect the connection of modules to system or components to modules.
- Implementation of multimedia features.

4.1 Sensors

In this project, Two type of sensors were used. Microphones and Infrared distance detectors. Both of these sensors produce analog signals which can be read by Analog to Digital(ADC) converters of the microcontrollers.

Our microcontrollers has 12-bit ADC which help us to have a great resolution of the incoming signal.

Sampling rate The analog signal is continuous in time and it is necessary to convert this to a flow of digital values. It is therefore required to define the rate at which new digital values are sampled from the analog signal. The rate of new values is called the sampling rate or sampling frequency of the converter.

The time required in our current setup to read one sample from microcontroller ADC is 7 microsecond which is too high and also one sample is not a value that we can rely on in order to detect the changes. Therefore, a sampling method was used in order to have a better and reliable value from each ADC. The following code was used in order to have a good sample from ADC.

Here is the code of our ADC sampler. It uses a simple averaging algorithm.

```
int16 SampleADC(uint16 *values) {
      unsigned long sum = 0;
      for (uint16 i = 0; i < ADC_SAMPLE_SIZE; i++) {
            sum += values[i];
      }
      int16 average = sum / ADC_SAMPLE_SIZE;
      return average
}
```

In the above code, The ADC_SAMPLE_SIZE is 1024. This average value is considered as one true value for each sensor.

4.2 CAN Network Setup

In order to make the CAN network working, the "HardwareCAN" support library from Maple was used. This library provides comprehensive functions of the CAN network and makes it easy to use.

First let's summarize the general information about CAN bus: [8]

4.2.1 Basic Message Format

The frame format is as fig 4.1:

Field	Bits	Field	Bits
Start-of-frame (SOF)	1	Data field	0 – 64
Identifier (ID)	11	CRC	15
Remote transmission request (RTR)	1	CRC delimiter	1
Identifer extension bit (IDE)	1	ACK slot	1
Reserved bit (r0)	1	ACK delimiter	1
Data length code (DLC)	4	End-of-frame (EOF)	7

Figure 4.1: CAN message format

Identifier(ID) A (unique) identifier which also represents the message priority(11 bits)

Data length code (DLC) Number of bytes of data (4bits)

Data Field Data to be transmitted (length in bytes dictated by DLC field) 0–64 bits (0-8 bytes)

4.2.2 ID allocation

Message IDs must be unique on a single CAN bus, otherwise, two nodes would continue transmission beyond the end of the arbitration field (ID) causing an error.
In the early 1990s, the choice of IDs for messages was done simply on the basis of identifying the type of data and the sending node; however, as the ID is also used as the message priority, this led to poor real-time performance. In those scenarios, a low CAN bus utilization of circa 30% was commonly required to ensure that all messages would meet their deadlines. However, if IDs are instead determined based on the deadline of the message, the lower the numerical ID and hence the higher the message priority, then bus utilizations of 70 to 80% can typically be achieved before any message deadlines are missed.

4.2.3 Setup

Setting up the library is quite easy. It is needed to be included in code and run a few lines to set up pins and etc. For comfortability reasons two functions for sending and receiving the messages have been written.

```
CAN_TX_MBX RobotModule::CANsend(CanMsg *pmsg)
{
      CAN_TX_MBX mbx;

      do
      mbx = canBus.send(pmsg) ;
      while(mbx == CAN_TX_NO_MBX) ;
      return mbx ;
}
```

Listing 4.1: Send a CAN message

```
CanMsg * RobotModule::CANreceive() {
      while(!canBus.available());
      Serial1.print("Got the Data");
      CanMsg *r_msg = canBus.recv();
      if (r_msg) {
```

```
            return r_msg;
        }
}
```

Listing 4.2: Receive a CAN message

4.3 Hardware Connection Detection

One of the main goals of this project is to be able to detect that the modules or hardware components are connected to the system or not. We divided this task into two main part. First to detect if the module is in the network and Second for each module to detect their sensors or actuators are connected or not.

4.3.1 Module Connection

In order to detect, a periodical beacon has been considered. This beacon which is a 8-byte message contains the Identification number, type and other necessary information related to each module. Then every one second each module send this "Life" beacon to notify other modules of their existence. Following is the procedure of sending life beacon.

Algorithm 1 Life Beacon send algorithm

```
1: procedure SENDING LIFE BEACON
2:     while true do
3:         Message(1) = Module Id
4:         Message(2) = Module Type
5:         SendCAN(Message)
6:         Delay 1 second
7:     end while
8: end procedure
```

The other task is to find other modules in the network based on their life beacons. In CAN network the messages are broadcast to every node, so, All modules will receive other's life beacons.

Provided that, if I receive a life beacon I will add it to the modules list and if I don't receive the life beacon of one module for more than 3 periods, I consider it disconnected. Two different Procedures are

needed. One for receiving beacons and update the modules list with the time of the receive and the other for checking the list and see if there is any disconnected module.

Algorithm 2 Life Beacon receive algorithm

```
1: procedure Sending Life Beacon
2:     while true do
3:         while Beacon message not received do
4:         end while;
5:         Beacon(id) = life beacon
6:         Beacon_times(2) = Now
7:     end while
8: end procedure
```

Algorithm 3 Beacon check algorithm

```
1: procedure Sending Life Beacon
2:     while true do
3:         for i=0; i<Beacon.size;i++ do;
4:             if Beacon_times(i) + 3 * 4 second < Now then
5:                 Remove beacon from the list
6:             end if
7:         end for
8:         Delay 4 second
9:     end while
10: end procedure
```

4.3.2 Components Connection

In this part various techniques defined in order to detect the connection of hardware components such as Microphones and Infrared Sensors, LEDs and etc.

4.3.2.1 Microphones and Infrared Sensors

Microphones and Infrared sensors produce analog signals which can be read by Analog to Digital(ADC) converters of the microcontroller. When a sensor connected and we read one sample from the ADC, The value is one 12-bit representing the current value of the sensor. In normal situation, the value is between 0 and 4095(maximum value of

the ADC). But there is a trick to fix the value to maximum when there is no sensor connected.

Pull-up resistor In electronic logic circuits, a pull-up resistor is a resistor connected between a signal conductor and a positive power supply voltage to ensure that the signal will be a valid logic level if external devices are disconnected or high-impedance is introduced [13].

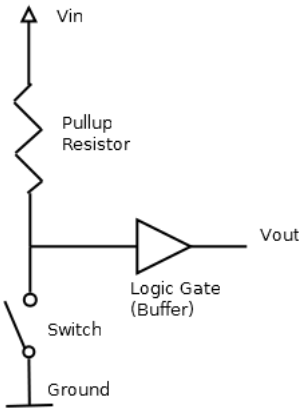

Figure 4.2: Pull-up resistor

When the switch is open the voltage of the gate input is pulled up to the level of Vin. When the switch is closed, the input voltage at the gate goes to ground.

The STM32F103RBT6 processor uses default pull-up resistors on I/O pins. So, It's pretty straight forward to use them in this application.

In Arduino, keyword INPUT_PULLUP has defined for this reason.

```
void setPullUp(int pin)
     pinMode(pin, INPUT_PULLUP);
}
```

Finally, in the ADC sampling algorithm, we check if the average value of samples is more than 4040, we assume that there is no sensor connected to that ADC pin.

4.3.2.2 LEDs

To detect the connection of LEDs to the pins, a simple logic was used. One GND and one input for each LED component considered. In panel mount 3.5mm jack these two pins are not connected together but in the component it-self, they are soldered together. Also, The input mode is INPUT_PULLUP. Consequently, if the component is disconnected, the input is in the HIGH state and if connected it is in the LOW state.

Provided that, following algorithm used to periodically check the status of LED connection.

Algorithm 4 Check status of LED component

```
1: while true do
2:     Set the input pin to INPUT_PULLUP mode
3:     if input == LOW then
4:         LED connected
5:     else
6:         LED disconnected
7:     end if
8:     Delay 4 second
9: end while
```

4.3.2.3 Speakers

The speakers were a different story. The INPUT_PULLUP could not be used in order to detect the connection since it will interfere the operation of the speaker.

Since we have four microphones mounted on one of the modules, we decided to use them to determine the connection of the speakers. The media player used in the system is stereo and has two channels, one for left and the other for the right. We put two MP3 sound files which each of them only has one channel data and when we play them only one of the speakers will produce the sound. Finally, with microphones, we detect if there is any sound playing around. If yes that speaker is connected.

Unfortunately, if there is no microphone connected, we can not detect the speakers which represent real life!

4.4 Media

In this section, we would describe the methods of using VS1053 2.3.4. All this functionality is implemented in a light-weight SPI interface so nearly any microcontroller can play audio from an SD card. There's also a special MIDI mode that you can boot the chip into that and the board will read 'classic' 31250Kbaud MIDI data on a UART pin and act like a synth/drum machine - there are dozens of built-in drum and sample effects!

The connections of the board to the microcontroller is like the scheme in fig 4.3.

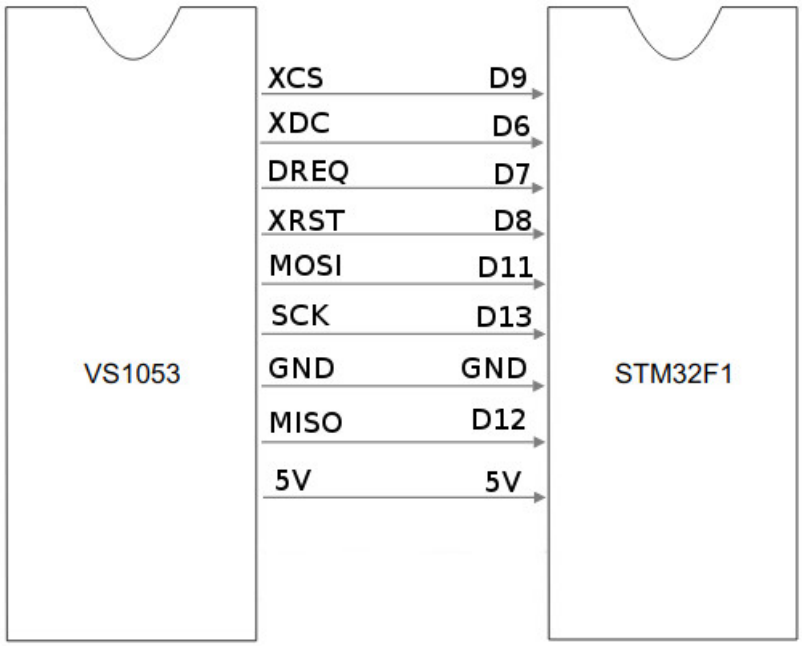

Figure 4.3: Connections of VS1053 to microcontroller

In all scenarios, the board needs to interact with SD card which is already present in the Olimex STM32 board. Since both (SD and VS1053) using SPI interface and Olimex has two SPIs, we are good to go. The SD card is connected to the second SPI and fully compatible with Arduino's SD library.

4.4.1 MP3 Playback

In order to play MP3 files with the board, we perform the following procedure.
First, we Include the necessary libraries.

```
#include <SPI.h>
#include <VS1003_STM.h>
#include <SD.h>
```

Then, we define the player.

```
SPIClass spiVS(1);
VS1003 player(9, 6, 7, 8, spiVS);
```

We should also Initialize the SD Card.

```
    while (!SD.begin(25)) {
        Serial2.println("Card failed, or not
            present");// don't do anything more:
        delay(100);
    }
```

Then, we set the player parameters and change it to MP3 mode.

```
player.begin();
player.modeSwitch(); //Change mode from MIDI to MP3
    decoding (Vassilis Serasidis).
player.setVolume(0x00);
```

Finally using "playfile" function, we can play the MP3 sound.

```
playFile(RobotModule::CurrentSound);
```

The full code of "playfile" function is available in Appendix A at the end of this document.

4.4.2 Recording Audio

Recording audio using VS1053 2.3.4 can be done using two output formats. One is uncompressed WAV and the Other is Ogg Vorbis.

4.4.2.1 WAV Audio format

Waveform Audio File Format (WAVE, or more commonly known as WAV due to its filename extension) is a Microsoft and IBM audio file format standard for storing an audio bitstream on PCs.

Though a WAV file can contain compressed audio, the most common WAV audio format is uncompressed audio in the linear pulse code modulation (LPCM) format. LPCM is also the standard audio coding format for audio CDs, which store two-channel LPCM audio sampled 44,100 times per second with 16 bits per sample. Since LPCM is uncompressed and retains all of the samples of an audio track, professional users or audio experts may use the WAV format with LPCM audio for maximum audio quality.

4.4.2.2 Ogg Vorbis format

Vorbis is a free and open-source software project headed by the Xiph.Org Foundation. The project produces an audio coding format and software reference encoder/decoder (codec) for lossy audio compression. Vorbis is most commonly used in conjunction with the Ogg container format and it is therefore often referred to as Ogg Vorbis.

Vorbis had been shown to perform significantly better than many other lossy audio formats in the past in that it produced smaller files at equivalent or higher quality while retaining computational complexity comparable to other MDCT formats such as AAC or Windows Media Audio.

4.4.2.3 Recording Ogg

We choose Ogg Vorbis as audio recording format because of the size. The Ogg files are much smaller than the wave files due to the compression.

Provided that, we perform the following procedure for recording audio in Ogg format.

First, we Include the necessary libraries.

```
#include <SPI.h>
```

```
#include <VS1003_STM.h>
#include <SD.h>
```

Then, we define the recorder.

```
SPIClass spiVS(1);
VS1003 recorder(9, 6, 7, 8, spiVS);
```

We should also Initialize the SD Card.

```
while (!SD.begin(25)) {
     Serial2.println("Card failed, or not present");
        // don't do anything more:
     delay(100);
}
```

Then, we set the player parameters and change it to MP3 mode.

```
recorder.begin();
recorder.setVolume(0x00);
// load plugin from SD card! We'll use mono 44.1KHz,
   high quality
if (! recorder.prepareRecordOgg("v44k1q05.img")) {
     Serial.println("Couldn't load plugin!");
     while (1);
}
```

Finally using "saveRecordedData" function, we can play the MP3 sound.

```
saveRecordedData();
```

The full code of "saveRecordedData" function is available in Appendix B at the end of this document.

4.4.3 Sound Localization

Rather than detecting what has been spoken, the work in this dissertation concerns the detection of where the sound originated from. The use of multiple, spatially distributed microphones allows the localization of sound sources by detecting differences between the signals received at each microphone. [14]

Time delay estimation is a method used to localize the targets depending on the sound; this method can be used in active or passive sound localization systems. The active systems send a sound pulse and receive the echo to estimate the time delay and localize the targets; an example for this method is the sonar (sound navigation system). The passive systems use the source sound itself. In the passive system, several methods could be used to localize the sound source that differs in the physical variables which they use to localize the sound; these methods are divided into three categories [15], which are

- Time difference of arrival, where the systems measure the difference in time between the signals received by the microphones to localize the sound source.
- Direction of arrival, where the phase difference between the signals is used to locate the sound source.
- Energy based sound localization, where the energy of sound wave decreases when the sound wave propagates in the air. By measuring the sound energy at different sensor locations, one may localize the sound source.

Time Delay Estimation Algorithms Several algorithms have been created to estimate the time delay; they vary in degrees of accuracy and computational complexity.

4.4.4 Cross Correlation Algorithm

This algorithm is a method to find the degree which the signals are correlated. The time delay estimation is obtained as the time lag that maximizes the cross correlation function. Assuming that we have a microphones array m 1, m 2 ... m i , the signals picked up by these microphones are

$$x_i = a_i s(t - \tau_i) + n_i(t). \tag{4.1}$$

where i denotes the microphone, s(t) is the sound signal, $n_i(t)$ is the stationary additive noise, τ_i is the propagation delay difference between two microphones and α_i is the signal attenuation.

The Fourier transform for the previous equation gives

$$X_i(f) = \alpha_i S_i(f) exp(-j2\pi f\tau_i) + N_i(f). \tag{4.2}$$

The cross power spectrum density function is calculated by the equation

$$R_{x_1x_2} = E[x_1(t)x_2(t-\tau)]. \tag{4.3}$$

where E{.} is the expected value, the value of τ which maximizes this function is the time delay estimation.

4.4.5 Generalized Cross Correlation

This algorithm is an improved version of cross correlation algorithm. The main advantages of this algorithm are the high accuracy and low computational cost. Fig 4.4 shows a diagram for this method. As can be seen, two filters H 1 , H 2 are added to ensure large sharp peak in the obtained cross correlation, which improves the accuracy of time delay estimation and improves SNR. After filtering the signals x 1 , x 2 , the resultant signals are delayed and then multiplied, then the resultant signals are integrated for a variety of delays until peak output is obtained.

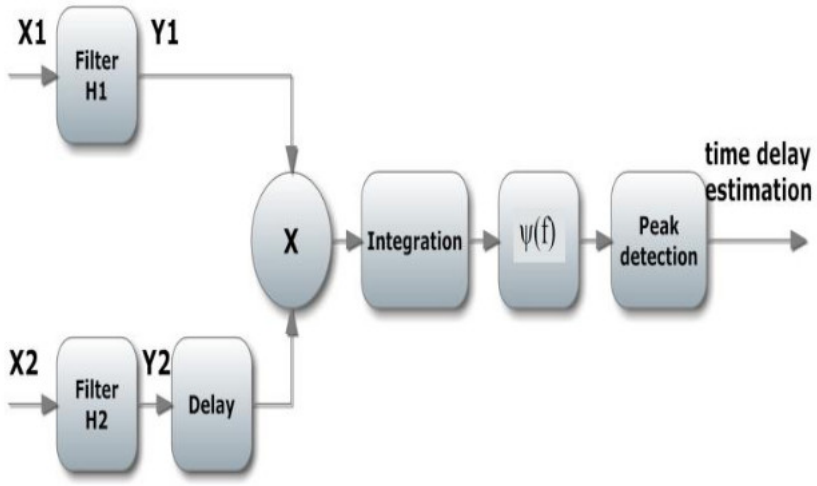

Figure 4.4: Generalized cross correlation method.

4.4.6 Microphone Setup

Our microphones setup is like Fig 4.5.

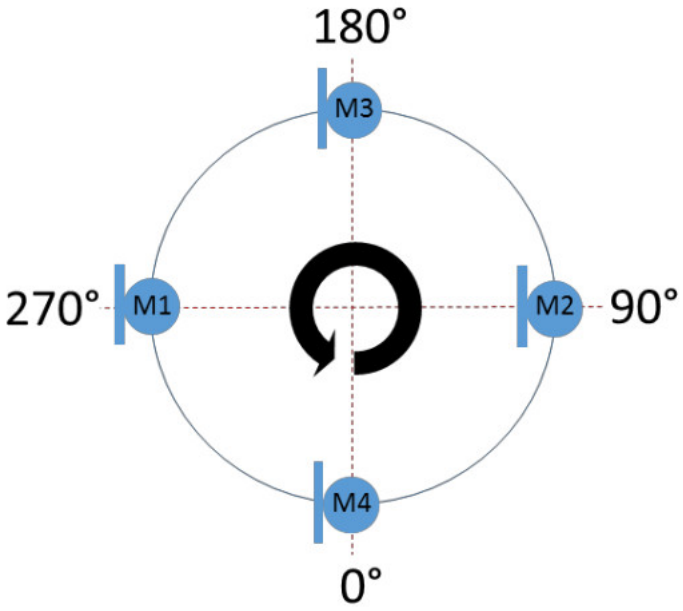

Figure 4.5: Generalized cross correlation method.

With the use of Generalized Cross Correlation 4.4.5 running on M1 and M2 and then on M3 and M4 we got two time delays which represent two points in 2D plain. By getting the arctan of these two points we get the angle of sound by the degree which is visible on the Fig 4.5.

The code for this algorithm is on Appendix C.

5

MODULE DESIGN AND IMPLEMENTATION

This chapter is considered one of the main parts of the project. In this chapter:

- First, we describe the overall system configuration.
- Then, we define the modules architecture and their responsibilities.
- Next, we review the operating system of the modules. Tasks, Mutex, Flags and etc are part of this section.
- Finally, we describe the module library which is designed specially for this robot and responsible for defining shared data, synchronizing the modules and etc.

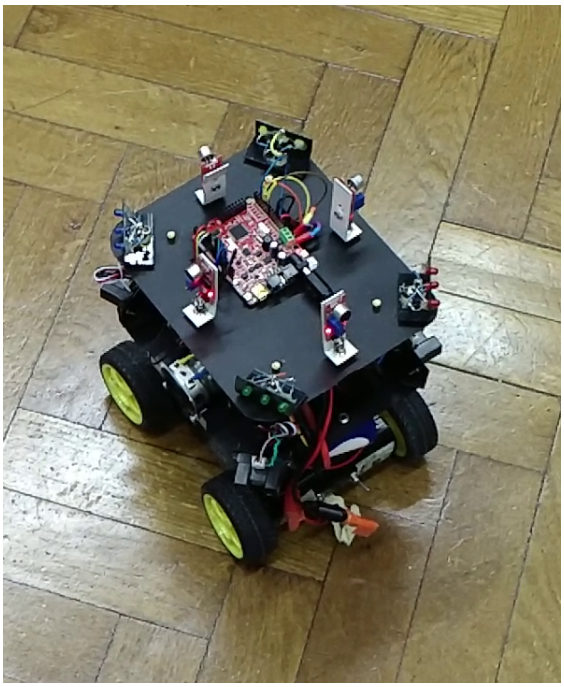

Figure 5.1: System Configuration

5.1 System Configuration

The robot we designed, consists of 3 modules in the top, middle and bottom levels of the platform.
The top module is connected to LEDs and Microphones.
The middle one is connected to Media board and Servo Motor.
Finally, the bottom module is connected to Wheel Motors and Infrared sensors.

You can see the current system configuration in Fig 5.2.

5.2 Module Description

The main unit of this robot is modules. One module consists of one main processor with networking capabilities and a set of hardware components either sensors or actuators.

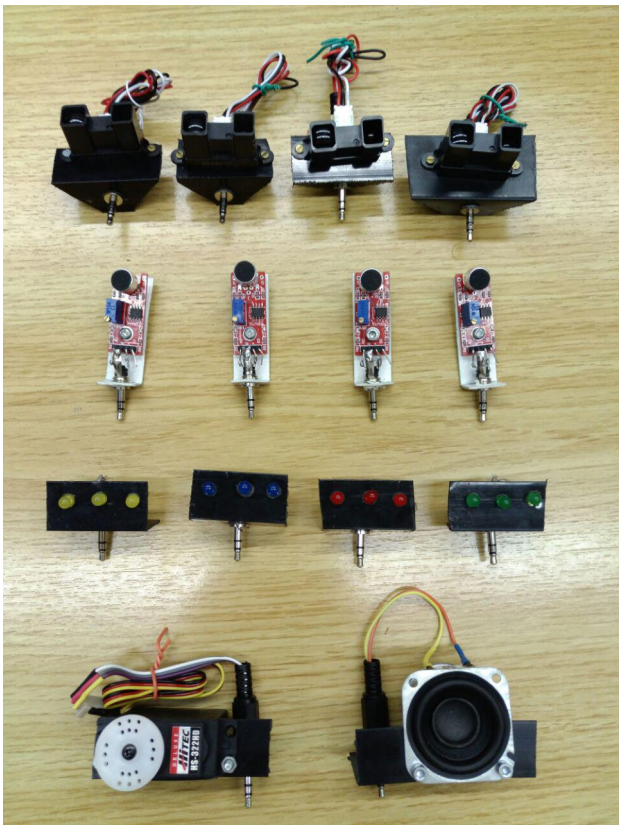

Figure 5.2: Module's hardware components. From top to bottom: Infrared Sensors, Microphones, LEDs, Servo Motor and Speaker.

In our case, the main processor is Olimex STM32 with CAN network equipped 2.4. Each module might have microphones or Infrared sensors, Multimedia capabilities or actuators like Servo Motor, Wheel motors or LED based on their design.

5.2.1 Module Architecture

Every module is connected to the system bus. System bus consists of 4 wires, which two of them are for power (5v and GND) and the other 2 are for networking (CAN High and CAN Low). If the module is not connected to the system, other modules will acknowledge by missing the "life beacon" 4.3.1 from that module. In Figure 5.3 the general system architecture is displayed.

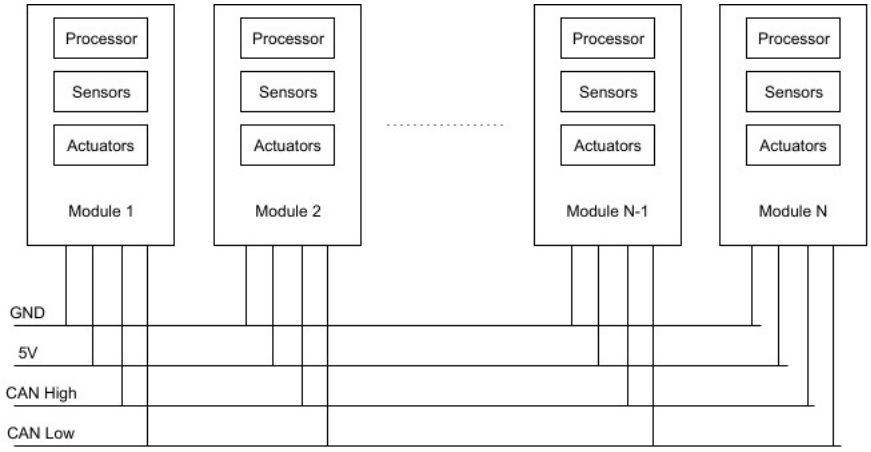

Figure 5.3: Modules connected to System Bus

5.2.2 Module Responsibilities

One module is responsible for collecting data from its sensors and managing their actuators. It should send the collected data from its sensors to other modules in the system and retrieve other modules collected data and use them in various decision-making scenarios.

The module should also send some "life" data periodically in order to announce its presence and capabilities to other modules.

Moreover, each module may process some of the collected data. One may calculate the angle of incoming sound from the environment or the other may play a sound or record audio.

5.3 Module Operating System - CoOS

We choose Cocox OS 2.7 as the operating system for each module which satisfies our real-time requirement. With the use of this operating system, each module is able to run up to 8 different tasks simultaneously. We used these tasks to send data to the network, receive data from the network, perform periodic jobs and etc. Since we use many functions of the CoOS, we describe the core functions of the CoOS in this section.

5.3.1 Task Architecture

A task can be in one of the following states in CoOS [16].

Ready State(TASK_READY): Ready tasks are tasks that can be executed (they are not waiting or dormant). They are not currently executing because another task of equal or higher priority is already in the Running state. A task will be in this state after it is created.

Running State(TASK_RUNNING): When a task is executing, the task is in Running state.

Waiting State(TASK_WAITING): Wait for an event to occur. A task will be in the waiting state if it is waiting for an event in CoOS.

The Dormant State(TASK_DORMANT): The task has been deleted and can not be used again. The dormant state is not the same as the waiting state. Tasks in the waiting state can be reactivated and be available again for scheduling when its waiting events have been satisfied. However, tasks in the dormant state can never be reactivated.

The state of a task can be changed in the above four states. CoSuspendTask() can be used to convert a task which is in the running or ready state to the waiting state. With calling CoAwakeTask() it is possible to convert the state of a task from the waiting state to the ready state.

The figure 5.4 shows which transitions are possible.

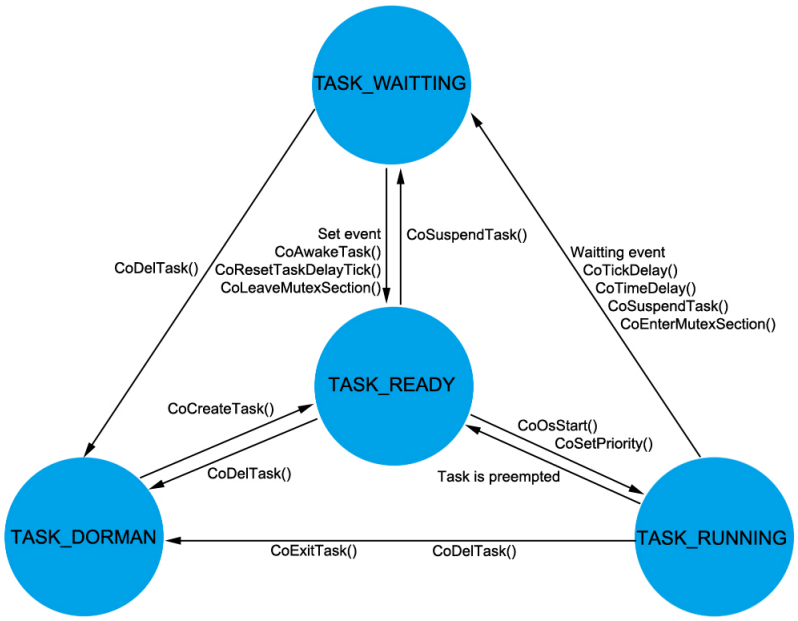

Figure 5.4: Module and System Architecture

Example:
This example uses 3 tasks (taskA, taskB, taskC). the main code looks like this:

```
int main () {
      CoInitOS ();// Initial CooCox CoOS

      /*!< Create three tasks     */
      CoCreateTask (taskA,0,0,&taskA_stk[
         STACK_SIZE_TASKA-1],STACK_SIZE_TASKA);
      CoCreateTask (taskB,0,1,&taskB_stk[
         STACK_SIZE_TASKB-1],STACK_SIZE_TASKB);
      CoCreateTask (taskC,0,2,&taskC_stk[
         STACK_SIZE_TASKC-1],STACK_SIZE_TASKC);

      CoStartOS ();  //  Start multitask
}
```

What happened is the CoOS (Real Time Operating System) is initialized. The next step is to create the all tasks. When that's done the CoOS can be started.

These are the 3 tasks:

```
void taskA (void* pdata)
{
     // place your code here.
     while(1);
}

void taskB (void* pdata)
{
     // place your code here.
     while(1);
}

void taskC (void* pdata)
{
     // place your code here.
     while(1);
}
```

5.3.2 Mutex

Mutexes can solve the "mutually exclusion" problem in CoOS. Coocox has solved this issue by the method of priority inheritance. Priority inversion means that a high-priority task is waiting for the low-priority task to release resources, at the same time the low-priority task is waiting for the middle priority tasks. In layman's terms, the high priority task can't function because it needs data from a low resource task that is waiting for a middle resource task to finish [16]. There are two classical methods to prevent this from happening:

- The priority inheritance strategy: The task which is possessing the critical section inherits the highest priority of all the tasks that request for this critical section. When the task exits from the critical section, it will restore to its original priority.
- The ceiling priority strategy: Upgrade the priority of the task which requests a certain resource to the highest priority of all the tasks that be likely to access this resource (and the highest priority is called the ceiling priority of this resource).

The mutex sections work like this: All tasks have a priority, when a task has a low priority, but one or a few lines of code that needs to be in the higher priority (must be executed directly) those lines of code must be in a mutex. Everything that is in the mutex gets the highest priority.
Example:

```
OS_MutexID mutexID;  // create a mutex variable

void myTaskA(void* pdata)
{
     // create a mutex section
     mutexID = CoCreateMutex ( );

     // enter the mutex section
     CoEnterMutexSection(mutexID );

     // critical code

     // leave the mutex section
     CoLeaveMutexSection(mutexID );
}
```

5.3.3 Critical Section

CoOS can handle critical code, if you have a piece of code that has to be executed in one go no matter what, use this method: Different from other kernels, CoOS does not handle the critical code section by closing interrupts, but locking the scheduler. Therefore, CoOS has a shorter latency for interrupt compared with others.

Since the time of enabling the interrupt relates to system responsiveness towards the real-time events, it is one of the most important factors offered by the real-time kernel developers. By locking the scheduler we can improve system real-time feature to the maximum comparing to other approaches. Since CoOS manages the critical section by forbidding to schedule task, user applications cannot call any API functions which will suspend the currently running task in critical sections.
Example:

```
void myTaskA(void* pdata)
{
     for (;;) {
          CoSchedLock ( ); // Enter Critical Section
          // Critical Code
          CoSchedUnlock ( ); // Exit Critical Section
     }
}
```

5.3.4 Flags

Flags can be used to communicate between tasks. A task can set or reset a flag and another task can check the status of a flag or wait with executing until a flag is set.

There are two ways of using flags: a single flag and multiple flags. Multiple flags must wait for each other, so the single flag is faster. CoOS supports up to 32 flags.

Moreover, There are two kinds of flags in CoOS: the ones reset manually and the ones reset automatically. When a task has waited for a flag which reset automatically, the system will convert the flag to not-ready state. If the flag is reset manually, there won't be any side effect. Therefore, when a flag which reset manually converts to the ready state, all the tasks which is waiting for this event will convert to the ready state until you call CoClearFlag() to reset the flag to the not-ready state.

When a flag which reset automatically converts to the ready state, only one task which are waiting for this event will convert to the ready state. Since the waiting list of the event flags is ordered by the principle of FIFO, towards the event which reset automatically only the first task of the waiting list converts to the ready state and others that are waiting for this flag are still in the waiting state.

Suppose there are three tasks (A, B, C) waiting for the same flag "I" which resets manually. When "I" is ready, all the tasks will be converted (A, B, C) to the ready state and then inserted into the ready list. Suppose "I" is a flag which reset automatically and the tasks (A, B, C) are listed in sequence in the waiting list. When "I" is ready, it will inform task A. Then I will be converted to the not-ready state. Therefore B and C will keep waiting for the next ready state of flag I

in the waiting list. You can create a flag by calling CoCreateFlag() in CooCox CoOS. After being created, you can call CoWaitForSingleFlag() and CoWaitForMultipleFlags() to wait for a single flag or multiple flags.
Example:

```
OS_FlagID flagID;   // declare a flag

void taskA(void* pdata)  // This is one of the two
   tasks that is created in the main loop
{
     GPIOSetDir(1, 18, 1); // Set PIO1_18 as output
     uint32_t flipflop = 0; // Make a local variable
        and give it the value 0
     flagID = CoCreateFlag(0, 0);// Create a flag and
         set it in manual reset modus
     for (;;) {        // Make a never ending loop
          if(flipflop == 0)    // If variable is zero
              make variable 1 and clear the flag
          {
               flipflop = 1;
               CoClearFlag(flagID);
          }
          else    // If variable is not zero make
             variable 0 and set the flag
          {
               flipflop = 0;
               CoSetFlag(flagID);
          }
          GPIOSetValue(1, 18, flipflop);   // Set
             PIO1_18 HIGH or LOW
          CoTickDelay (100);      // A delay of 100
             milliseconds
     }
}
void taskB (void* pdata){         // This is one of
   the two tasks that is created in the main loop
     GPIOSetDir(1, 19, 1);   // Set PIO1_19 as output
     StatusType result;     // Create a status type
        variable
     for (;;) // Make a never ending loop
     {
          GPIOSetValue(1, 19, 0);       // Make
             PIO1_19 LOW
```

```
            result = CoWaitForSingleFlag(flagID, 500);
                // Put the status of the flag in the
                variable, if the flag wasn't recieved in
                 500mS a timeout occurs
            GPIOSetValue(1, 19, 1);        // Make
                PIO1_19 HIGH
        }
}
```

5.4 Module Library

For ease of use reasons, we write a complete module library where we well define all the data, functions and tasks which are shared among different modules. The complete source code of this library is available in the Appendix D.

Each module in the system should include the module library, initialize it and set its own Identifier number. Here is an example:

```
#include <module.h>

int Id = 1;
RobotModule::Init(Id);
```

More importantly, every module should declare that which of the hardware components which are available in robot platform such as motors, microphones, distance sensors and etc, are connected to them. Consequently, the module is responsible for collecting data for its sensors or managing the actuators. Data from other modules update from them through the network. Here is an example of declaring ownership of the components:

```
RobotModule::MicrophoneNorth.mine = true;
RobotModule::MicrophoneNorth.pin = 15;
...
RobotModule::LEDNorthEast.mine = true;
RobotModule::LEDNorthEast.pin = 4;
RobotModule::LEDNorthEast.pin_support = 5;
```

Then, we start the tasks which may be related to the specific module or shared among the modules. Here is an example:

```
CoCreateTask(&(RobotModule::GameOneTask),
(void *) 0,
2,
&RobotModule::GameOneTaskStk[TASK_STK_SIZE - 1],
TASK_STK_SIZE
);
CoCreateTask(ADCSamplerTask,
(void *) 0,
2,
&ADCSamplerTaskStk[TASK_STK_SIZE - 1],
TASK_STK_SIZE
);
```

5.4.1 Inner Task Communication

We use CoOS flags 5.3.4 to make inner task communications possible. Examples of needed communications are when we want to ping the other tasks to send the data to the network or when a task receives data from the network and should ping another task to run.

5.4.2 Shared Tasks

We write the total of 4 shared tasks which all modules should run in order to make system working and in sync.

- Network Send Task
- Network Receive Task
- Interval Task
- Game Perform Task

5.4.2.1 Network Send Task

This task is responsible for sending the sensor collected or another type of data to other modules through the CAN network. Network

Send task is always waiting to be pinged 5.4.1 by other tasks when their data is ready.

We defined 7 different flags in which we declare which type of new data is available.

- CAN_Microphone_FlagId : The flag to declare new microphone sample is recorded.
- CAN_Sound_Angle_FlagId : The flag to declare new incoming sound angle is calculated.
- CAN_IR_FlagId : The flag to declare new Infrared sample is recorded.
- CAN_LED_FlagId : The flag to declare that the LED status is changed.
- CAN_Wheel_Motors_FlagId : The flag to declare that the Motor status is changed.
- CAN_Servo_Motors_FlagId : The flag to declare that the Servo Motor status is changed.
- CA_Interval_FlagId : The flag to declare that new data is generated in Interval task 5.4.2.3.

Then, We know that there is data available to send. We detect the type of that data and also check if other type of data is also ready to dispatch in this session.

In order to make the network packet ready, we should pay attention that the length of each CAN message is 8 byte at maximum. In every message, we have two static byte which is the same in all messages. First is the module Identifier number (Id) and the second is Command Code which is the code which helps the other modules to understand which type of data is in the message.

Provided that, There are 6 remaining bytes which can be used to put the data inside. If a single data is more than one byte, We used the "Union" technique to split it into the bytes and then reconstruct it in the other module. Here is an example of splitting a uint16 to two single bytes:

```
union TwoBytes
{
     uint16 u16;
```

```
    uint8 u8[2];
} twoBytes;

twoBytes.u16 = MicrophoneNorth.value;
t_msg.Data[0] = twoBytes.u8[0];
t_msg.Data[1] = twoBytes.u8[1];
```

We should remember that the flags need to be cleared at the end of each send session. If we don't reset the flags, the task is always running and won't wait for further flags settings.

5.4.2.2 Network Receive Task

Network receive task is always listening for new data from the network. So, when there is no data the task is waiting and calling canBus.available() to find out if there is new data available from the network.

Once there is new data available, It will receive the message. Then, We parse that incoming message. We know that the first byte is the Identifier number of the module which had sent the message and second byte is the command code.

Based on the command code, we assign the values to the shared data variables in the module library. If the data is more than one byte, We again reconstruct it with "Union" technique which we described before.

It's important to mention that when a new message arrives from a module the last seen time of them is updated.

5.4.2.3 Interval Task

We define an interval task to perform following tasks inside:

- Sending life beacon
- Checking for modules and components hardware connection
- Sending the module data periodically
- Checking for coordinator and choose if necessary

5.4.2.4 Game Perform Task

We consider the Game Perform Task as the main task for playing the games. In this task, First, we check the currently selected game and based on that each module understands their duty and behavior.

5.4.3 Decision Making

In this section, We explain the methods for selecting system coordinator and games.

5.4.3.1 Choosing Coordinator

We use highest module identification number as the coordinator. In the interval task, Each module compares their Id to the highest Id in the system.

5.4.3.2 Game Selection

The coordinator module will select the game based on two scenarios.

- Event-based: In this mode, the coordinator decides to change the game based on the external events like incoming sound or the distances from surroundings.
- Random based: While there are no external events the coordinator may want to change the current game in order to get outside attention. This change will be random and the coordinator selects current game between the available games and components.

Then, Each module will retrieve the selected game from the network and apply it.

5.4.4 Games

5.4.4.1 Game - Relax

This game is running when there is no external event has triggered for a while. The media player would play a relaxing song while the LEDs blinking slowly and the sensors are scanning for external events in order to change the game.

If there is no external event for a while after selection of this game, The coordinator may decide to randomly change the game.

5.4.4.2 Game - Happy

When coordinator receives clap event, It would switch the current game to Happy. In this game, the media player would play a happy song while LEDs blinking fast and the servo motor would go left and right in order to have a happy impression. The robot may change its location with a static pattern.

By clapping again we can change the game to another random one. Also the game will automatically change when it runs for 1 minute.

5.4.4.3 Game - Escape

The coordinator will monitor the changes on Infrared sensors and if someone passing by the robot, It will get away from its position to opposite direction. The media player would play Danger song and the LED of danger zone blink.

5.4.4.4 Game - Give me your hand

When modules are playing this game, the coordinator will monitor incoming sound angle. Based on this angle the servo motor would turn left or right while LEDs of that direction blink. The media player would play a sound in order to inform the direction.

6 CONCLUSION

In this chapter, we describe the final conclusion and suggestions for possible future work based on this paper.

6.1 Conclusion

In this paper, we have presented the design of a Networked modular robot which can perform various games. Different modules can communicate, synchronize and decide to perform a game together. This work has answered the following requirements.

- Is it possible to have a modular robot without the core to manage them?
- Is it possible to detect the existence of a module in the system?
- Is it possible to detect the existence of a hardware component like microphones, infrared sensors or LEDs in a module?
- Is it possible to implement multimedia capabilities with low-cost and good quality?
- Is it possible to detect the incoming sound angle or external events?

We have implemented complete hardware and software setup to answer above questions. The system consists of different modules which have specific responsibilities. In our setup, there is no core and the modules decide that make one among them as coordinator. The coordinator is responsible for selecting the current game for all modules and sending them the result. If the coordinator has removed from the system, others will decide a new one.

We described and implemented different techniques to check the existence of modules or hardware components. The modules share their collected data with others and with the help of their sensors and actuators may calculate incoming sound angle or detecting changes in the environment. Game selection is based on the existence of different modules and hardware components.

6.2 Future work

- The modules could be combined with hardware components in a smaller package and the 4 wire mechanical connector could be implemented in order to mount on the robot platform.
- Beside the above suggestion, each module could come as different level in robot platform.
- More modules with a wide range of components like other distance detectors, temperature sensors and etc could be designed.
- With the current configuration, Many more games could be implemented. It needs more creativity and research on children favorite games.
- Game selection methods could be studied based on favorites of the children.

BIBLIOGRAPHY

[1] Prof. Kai Arras. "Social Robotics Seminar SS2015." In: (2015). URL: http://srl.informatik.uni-freiburg.de/teachingdir/ss15/SemSS15-introduction.pdf (cit. on p. 1).

[2] Lina Becerra Paola Esquivel Kim Adams Adriana Rios. "USING ROBOTS TO ACCESS PLAY AT DIFFERENT DEVELOPMENTAL LEVELS FOR CHILDREN WITH SEVERE DISABILITIES: A PILOT STUDY." In: (2015). URL: https://www.resna.org/sites/default/files/conference/2015/robotics/adams.html (cit. on p. 1).

[3] Tarun Agarwal. "What is a Network in Embedded Systems? – Different Types of Networks." In: (). URL: https://www.elprocus.com/important-of-network-in-embedded-systems-for-beginners/ (cit. on p. 4).

[4] Dallas Semiconductor. "Fundamentals of RS–232 Serial Communications." In: (2017). URL: http://ecee.colorado.edu/~mcclurel/dan83.pdf (cit. on p. 5).

[5] National Instruments. "RS-232, RS-422, RS-485 Serial Communication General Concepts." In: (2016). URL: http://www.ni.com/white-paper/11390/en/ (cit. on p. 6).

[6] Texas Instruments. "RS-422 and RS-485 Standards Overview and System Configurations, Application Report." In: (2010). URL: http://focus.ti.com/lit/an/slla070d/slla070d.pdf (cit. on p. 7).

[7] NXP Semiconductors. "I2C-bus specification and user manual." In: (2014). URL: http://www.nxp.com/documents/user_manual/UM10204.pdf (cit. on p. 7).

[8] Vector Group. "Controller Area Network (CAN)." In: (2013). URL: https://vector.com/vi_controller_area_network_en.html (cit. on pp. 8, 22, 27).

[9] VLSI Solutions. "VS1053 - Ogg Vorbis / MP3 / AAC / WMA / FLAC / MIDI Audio Codec Chip." In: (2016). URL: http://www.vlsi.fi/en/products/vs1053.html (cit. on p. 11).

[10] Simplecortex by BRC-Electronics. "Introduction to Cocox OS." In: (2011). URL: http://www.brc-electronics.nl/coos (cit. on p. 16).

[11] Wikibooks Open Books. "Robotics/Types of Robots/Wheeled." In: (2016). URL: https://en.wikibooks.org/wiki/Robotics/Types_of_Robots/Wheeled (cit. on pp. 18, 19).

[12] Lammert Bies. "Introduction to RS485." In: (2015). URL: https://www.lammertbies.nl/comm/info/RS-485.html (cit. on p. 21).

[13] Paul Horowitz and Winfield Hill. *The Art of Electronics, 2nd edition*. Cambridge University Press, 1989. ISBN: 0-521-37095-7 (cit. on p. 31).

[14] Steven George Goodridge. "Multimedia Sensor Fusion for Intelligent Camera Control and Human-Computer Interaction." In: (1997). URL: http://www4.ncsu.edu/~kay/msf/sound.htm (cit. on p. 37).

[15] Hasan Khaddour. "A Comparison of Algorithms of Sound Source Localization Based on Time Delay Estimation." In: (2011). URL: http://www.elektrorevue.cz/file.php?id=200000713-385a639545 (cit. on p. 37).

[16] www.coocox.org. "CooCox CoOS User's Guide." In: (2009). URL: http://coocox.org/download/downloadfile/CoOS/PDF/CooCox_CoOS_User_Guide5%C3%94%C3%8229%C2%BA%C3%85%C2%B1%C2%B8%C2%B7%C3%9D.pdf (cit. on pp. 43, 45).

APPENDIX A: MP3 PLAYBACK SOURCE CODE

Listing A.1: Play MP3 file function

```
#include <VS1003_STM.h>
#include <SPI.h>
#include <SD.h>
SPIClass spiVS(1);
VS1003 player(9, 6, 7, 8, spiVS);
while (!SD.begin(25)) {
    Serial1.println("Card failed, or not present");//
        don't do anything more:
    delay(100);
}
Serial1.println("card initialized.");
player.begin();
player.modeSwitch(); //Change mode from MIDI to MP3
   decoding
player.setVolume(0x15);
void playFile(char *fileName) {
    uint8_t buff[BUFF_LEN];
    File dataFile;
    int bytesRead;
    int fileSize;
    int totalBytesRead = 0;

    dataFile = SD.open(fileName);

    if (!dataFile) {
        Serial1.println("Error opening file");
        return;
    }

    Serial1.print("Playing ");
    Serial1.println(fileName);
    fileSize = dataFile.size(); // need to know the
       size of the file

    // loop until end of file
    while (totalBytesRead != fileSize) {
```

```
        do {
35          bytesRead = dataFile.read(buff, BUFF_LEN)
                ;// try to read a buffers worth
            if (bytesRead == -1) // Oops. SD didnt
                manage to read anything ....
37          {
                delay(10);// Workaround for problem
                    in SD lib.
39          }
        } while (bytesRead == -1);// loop around if
            last read failed
41


43      player.playChunk(buff, bytesRead);
        totalBytesRead += bytesRead;// keep track of
            how much of the file we've played
45  }
    dataFile.close();
47 }
```

B

APPENDIX B: OGG RECORDING SOURCE CODE

Listing B.1: Record Ogg Sound function

```
#include <VS1003_STM.h>
#include <SPI.h>
#include <SD.h>
SPIClass spiVS(1);
VS1003 player(9, 6, 7, 8, spiVS);
File recording;  // the file we will save our
    recording to
#define RECBUFFSIZE 512
#define REC_BUTTON 3
uint8_t recording_buffer[RECBUFFSIZE];
#ifndef _BV
  #define _BV(x) (1<<(x))
#endif

// see if the card is present and can be initialized:
while (!SD.begin(25))
{
    Serial.println("Card failed, or not present");//
        don't do anything more:
    delay(100);
}
Serial.println("card initialized.");

player.begin();
player.setVolume(0x00);
Serial.println("Begin of Loop.");
// load plugin from SD card! We'll use mono 44.1KHz,
    high quality
if (! player.prepareRecordOgg("v44k1q05.img")) {
    Serial.println("Couldn't load plugin!");
    while (1);
}

uint16_t saveRecordedData(boolean isrecord) {
    uint16_t written = 0;

```

```
    // read how many words are waiting for us
35  uint16_t wordswaiting = player.
        recordedWordsWaiting();

37  // try to process 256 words (512 bytes) at a time
        , for best speed
    while (wordswaiting > 256) {
39    //Serial.print("Waiting: "); Serial.println(
          wordswaiting);
      // for example 128 bytes x 4 loops = 512 bytes
41    for (int x=0; x < 512/RECBUFFSIZE; x++) {
        // fill the buffer!
43      for (uint16_t addr=0; addr < RECBUFFSIZE;
            addr+=2) {
          uint16_t t = player.recordedReadWord();
45        //Serial.println(t);
          recording_buffer[addr] = t >> 8;
47        recording_buffer[addr+1] = t;
        }
49      if (! recording.write(recording_buffer,
            RECBUFFSIZE)) {
           Serial.print("Couldn't write "); Serial.
              println(RECBUFFSIZE);
51         while (1);
        }
53    }
      // flush 512 bytes at a time
55    recording.flush();
      written += 256;
57    wordswaiting -= 256;
    }
59
    wordswaiting = player.recordedWordsWaiting();
61  if (!isrecord) {
      Serial.print(wordswaiting); Serial.println("
          remaining");
63    // wrapping up the recording!
      uint16_t addr = 0;
65    for (int x=0; x < wordswaiting-1; x++,addr+=2)
          {
        // fill the buffer!
67      uint16_t t = player.recordedReadWord();
        recording_buffer[addr] = t >> 8;
69      recording_buffer[addr+1] = t;
      }
```

```
71        if (! recording.write(recording_buffer, (
              wordswaiting-1)*2)) {
              Serial.println("Couldn't write!");
73            while (1);
          }
75        written+=wordswaiting-1;
          recording.flush();
77        uint16_t t = player.recordedReadWord();
          recording_buffer[0] = t >> 8;
79        recording_buffer[1] = t;
          player.read_register(0xf);
81        if (! (player.read_register(0xf) & _BV(2))) {
             recording.write(recording_buffer, 2);
83           written+=2;
          } else {
85           recording.write(recording_buffer, 1);
             written+=1;
87        }
          recording.flush();
89     }

91     return written;
}
```

C

APPENDIX C: SOUND LOCALIZATION ALGORITHM IMPLEMENTATION

Listing C.1: Sound Localization Algorithm Implementation

```
float getSoundAngle() {
    int i;
    uint16 max_values_1 = 0;
    uint16 max_values_2 = 0;
    uint16 max_values_3 = 0;
    uint16 max_values_4 = 0;
    uint16 max_index_1 = -1;
    uint16 max_index_2 = -1;
    uint16 max_index_3 = -1;
    uint16 max_index_4 = -1;
    uint32 sum_values_1 = 0;
    uint32 sum_values_2 = 0;
    uint32 sum_values_3 = 0;
    uint32 sum_values_4 = 0;
    for (i = 0; i < ADC_SAMPLE_SIZE; i++) {
        if (adc_values_1[i] > max_values_1) {
            max_index_1 = i;
            max_values_1 = adc_values_1[i];
        }
        if (adc_values_2[i] > max_values_2) {
            max_index_2 = i;
            max_values_2 = adc_values_2[i];
        }
        if (adc_values_3[i] > max_values_3) {
            max_index_3 = i;
            max_values_3 = adc_values_3[i];
        }
        if (adc_values_4[i] > max_values_4) {
            max_index_4 = i;
            max_values_4 = adc_values_4[i];
        }
        sum_values_1 += adc_values_1[i];
        sum_values_2 += adc_values_2[i];
        sum_values_3 += adc_values_3[i];
        sum_values_4 += adc_values_4[i];
```

```
36      }
        if (max_values_1 - (sum_values_1 /
            ADC_SAMPLE_SIZE) < 20) return NULL;
38      int diff_1 = getDiffIndex(max_index_1,
            max_index_3, ADC_SAMPLE_SIZE);
        int diff_2 = getDiffIndex(max_index_2,
            max_index_4, ADC_SAMPLE_SIZE);
40      for (i = 0; i < ADC_SAMPLE_SIZE; i++) {
            Serial1.print(adc_values_1[i]);
42          Serial1.print(",");
        }
44      Serial1.println();
        for (i = 0; i < ADC_SAMPLE_SIZE; i++) {
46          Serial1.print(adc_values_3[i]);
            Serial1.print(",");
48      }
        return (float) atan2(diff_2, diff_1) * 180 / PI;
50  }

52  int16 getDiffIndex(uint16 index1, uint16 index2,
        uint16 totalReads) {
        int16 diff = index2 - index1;
54      if (diff > totalReads / 2) {
            diff = totalReads - diff;
56      }
        if (diff < -1 * (totalReads / 2)) {
58          diff = -1 * totalReads - diff;
        }
60      return diff;
    }
```

D

APPENDIX D: MODULE LIBRARY SOURCE CODE

Listing D.1: Module Library Header

```
#ifndef MODULE_LIBRARY_H
#define MODULE_LIBRARY_H
#include "motor.h"
#include <HardwareCAN.h>
#include <MapleCoOS116.h>

template <class T> class Component {
public:
    Component(T def);
    bool mine;
    volatile T value;
    uint8 pin;
    uint8 pin_support;
};
template <class T>
Component<T>::Component(T def) {
    mine = false;
    value = def;
    pin = -1;
    pin_support = -1;
}



class RobotModule {
public:
    static const uint8 CAN_MICROPHONE_DATA_1 = 1;
    static const uint8 CAN_MICROPHONE_DATA_2 = 2;
    static const uint8 CAN_SOUND_ANGLE_DATA = 3;
    static const uint8 CAN_IR_DATA_1 = 4;
    static const uint8 CAN_IR_DATA_2 = 5;
    static const uint8 CAN_LED_DATA = 6;
    static const uint8 CAN_WHEEL_MOTORS_DATA = 7;
    static const uint8 CAN_SERVO_MOTORS_DATA = 8;
    static const uint8 CAN_LIFE_DATA = 9;
    static const uint8 CAN_GAME_DATA = 10;
```

```
37
        static const uint8 MOTOR_FORWARD  = 1;
39      static const uint8 MOTOR_BACKWARD = 2;
        static const uint8 MOTOR_STOP = 3;
41
        static const uint8 MAX_MODULE_COUNT = 10;
43
        static const uint16 IntervalDelay = 1000;
45      static const uint16 GameSelectionDelay = 20000;

47      static const uint8 GAME_RELAX = 0;
        static const uint8 GAME_ESCAPE = 1;
49      static const uint8 GAME_HAPPY = 2;

51      static const uint8 TOTAL_GAMES = 3;

53      static uint8 SelectedGame;

55      static HardwareCAN canBus;
        static OS_FlagID CAN_Microphone_FlagId;
57      static OS_FlagID CAN_Sound_Angle_FlagId;
        static OS_FlagID CAN_IR_FlagId;
59      static OS_FlagID CAN_LED_FlagId;
        static OS_FlagID CAN_Wheel_Motors_FlagId;
61      static OS_FlagID CAN_Servo_Motors_FlagId;
        static OS_FlagID CAN_Interval_FlagId;
63      static OS_FlagID Player_FlagId;
        static OS_FlagID CAN_Game_FlagId;
65
        static volatile bool MicrophoneDataAvailable;
67      static volatile bool SoundAngleDataAvailable;
        static volatile bool IRDataAvailable;
69      static volatile bool LEDDataAvailable;
        static volatile bool WheelMotorsDataAvailable;
71      static volatile bool ServoDataAvailable;
        static volatile bool LifeDataAvailable;
73      static volatile bool GameDataAvailable;

75      static volatile bool IsPlayingMP3;

77      static uint8 CAN_ID;
        static bool IsCoordinator;
79
        //Microphones
81      static Component<int16> MicrophoneNorth;
```

```
    static Component<int16> MicrophoneSouth;
83  static Component<int16> MicrophoneWest;
    static Component<int16> MicrophoneEast;
85  //IR Sensors
    static Component<int16> IRNorth;
87  static Component<int16> IRSouth;
    static Component<int16> IRNorthEast;
89  static Component<int16> IRNorthWest;
    static Component<int16> IRSouthEast;
91  static Component<int16> IRSouthWest;
    //LEDs
93  static Component<int8> LEDNorthEast;
    static Component<int8> LEDNorthWest;
95  static Component<int8> LEDSouthEast;
    static Component<int8> LEDSouthWest;
97  // Motors
    static Motor MotorLeft;
99  static Motor MotorRight;
    //Sound Angle
101 static Component<float> IncomingSoundAngle;
    //MP3 Player
103 static Component<bool> Player;

105 static char *CurrentSound;
    static bool SpeakerLeftConnected;
107 static bool SpeakerRightConnected;

109 static char *WelcomeSound;
    static char *MusicSound;
111 static char *NorthWestDangerSound;
    static char *SouthWestDangerSound;
113 static char *NorthEastDangerSound;
    static char *SouthEastDangerSound;
115 static char *InfraredRemoved;
    static char *InfraredConnected;
117 static char *LEDRemoved;
    static char *LEDConnected;
119 static char *MicrophoneRemoved;
    static char *MicrophoneConnected;
121 static char *Despacito;

123 static OS_STK NetworkSendTaskStk[TASK_STK_SIZE];
    static OS_STK NetworkReceiveTaskStk[TASK_STK_SIZE
      ];
```

```
125     static OS_STK NetworkIntervalDataTaskStk[
            TASK_STK_SIZE];
        static OS_STK GamePerformTaskStk[TASK_STK_SIZE];
127
        static volatile bool ModulesInNetwork[
            MAX_MODULE_COUNT];
129     static volatile unsigned long ModulesLastSeen[
            MAX_MODULE_COUNT];

131     static void Init(int _can_id);

133     static void NetworkIntervalDataTask(void *pdata);

135     static void NetworkSendTask(void *pdata);

137     static void NetworkReceiveTask(void *pdata);

139     static void GamePerformTask(void *pdata);

141     static CAN_TX_MBX CANsend(CanMsg *pmsg);

143     static void PrintHex(uint8 b);

145     static bool isConnected(int32 oldValue, int32
            newValue);

147     static bool isDisconnected(int32 oldValue, int32
            newValue);
};
149
#endif
```

Listing D.2: Module Library Source code

```
#include "module.h"
2
#include <stdio.h>
4
HardwareCAN RobotModule::canBus(CAN1_BASE);
6


8 OS_FlagID RobotModule::CAN_Microphone_FlagId =
      CoCreateFlag(0,0);
  OS_FlagID RobotModule::CAN_Sound_Angle_FlagId =
      CoCreateFlag(0,0);
```

```
OS_FlagID RobotModule::CAN_IR_FlagId = CoCreateFlag
    (0,0);
OS_FlagID RobotModule::CAN_LED_FlagId = CoCreateFlag
    (0,0);
OS_FlagID RobotModule::CAN_Wheel_Motors_FlagId =
    CoCreateFlag(0,0);
OS_FlagID RobotModule::CAN_Servo_Motors_FlagId =
    CoCreateFlag(0,0);
OS_FlagID RobotModule::CAN_Interval_FlagId =
    CoCreateFlag(0,0);
OS_FlagID RobotModule::Player_FlagId = CoCreateFlag
    (0,0);
OS_FlagID RobotModule::CAN_Game_FlagId = CoCreateFlag
    (0,0);

uint8 RobotModule::CAN_ID = 0;
char * RobotModule::WelcomeSound = "welcome.mp3";
char * RobotModule::MusicSound = "music.mp3";
char * RobotModule::NorthWestDangerSound = "nwd.mp3";
char * RobotModule::SouthWestDangerSound = "swd.mp3";
char * RobotModule::NorthEastDangerSound = "ned.mp3";
char * RobotModule::SouthEastDangerSound = "sed.mp3";
char *RobotModule::InfraredRemoved = "infraredrem.mp3
    ";
char *RobotModule::InfraredConnected = "infraredcon.
    mp3";
char *RobotModule::LEDRemoved = "ledrem.mp3";
char *RobotModule::LEDConnected = "ledcon.mp3";
char *RobotModule::MicrophoneRemoved = "micrem.mp3";
char *RobotModule::MicrophoneConnected = "miccon.mp3"
    ;
char *RobotModule::Despacito = "dd.mp3";
char *RobotModule::CurrentSound = RobotModule::
    WelcomeSound;
Component<int16> RobotModule::MicrophoneNorth(-1);
Component<int16> RobotModule::MicrophoneSouth(-1);
Component<int16> RobotModule::MicrophoneWest(-1);
Component<int16> RobotModule::MicrophoneEast(-1);
//IR Sensors
Component<int16> RobotModule::IRNorth(-1);
Component<int16> RobotModule::IRSouth(-1);
Component<int16> RobotModule::IRNorthEast(-1);
Component<int16> RobotModule::IRNorthWest(-1);
Component<int16> RobotModule::IRSouthEast(-1);
Component<int16> RobotModule::IRSouthWest(-1);
```

```
//LEDs
Component<int8> RobotModule::LEDNorthEast(-1);
Component<int8> RobotModule::LEDNorthWest(-1);
Component<int8> RobotModule::LEDSouthEast(-1);
Component<int8> RobotModule::LEDSouthWest(-1);
//Motors
Motor RobotModule::MotorLeft(-1);
Motor RobotModule::MotorRight(-1);
//Player
Component<bool> RobotModule::Player(false);
//Coordination
bool RobotModule::IsCoordinator = false;

uint8 RobotModule::SelectedGame = -1;

volatile bool RobotModule::MicrophoneDataAvailable =
    false;
volatile bool RobotModule::SoundAngleDataAvailable =
    false;
volatile bool RobotModule::IRDataAvailable = false;
volatile bool RobotModule::LEDDataAvailable = false;
volatile bool RobotModule::WheelMotorsDataAvailable =
     false;
volatile bool RobotModule::ServoDataAvailable = false
    ;
volatile bool RobotModule::LifeDataAvailable = false;
volatile bool RobotModule::GameDataAvailable = false;
volatile bool RobotModule::IsPlayingMP3 = false;
Component<float > RobotModule::IncomingSoundAngle(0.0
    f);

OS_STK RobotModule::NetworkSendTaskStk[TASK_STK_SIZE]
     = {};
OS_STK RobotModule::NetworkReceiveTaskStk[
    TASK_STK_SIZE] = {};
OS_STK RobotModule::NetworkIntervalDataTaskStk[
    TASK_STK_SIZE] = {};
OS_STK RobotModule::GamePerformTaskStk[TASK_STK_SIZE]
     = {};

volatile bool RobotModule::ModulesInNetwork[
    MAX_MODULE_COUNT] = {};
volatile unsigned long RobotModule::ModulesLastSeen[
    MAX_MODULE_COUNT] = {};
```

```
78 void RobotModule::Init(int _can_id) {
       RobotModule::CAN_ID = _can_id;
80     canBus.map(CAN_GPIO_PB8_PB9);
       canBus.begin(CAN_SPEED_1000, CAN_MODE_NORMAL);
82     canBus.filter(0, 0, 0);
       canBus.set_irq_mode();
84 }

86 void RobotModule::NetworkIntervalDataTask(void *pdata
       ) {
       uint8 i = 0;
88     uint8 th = 15;
       int16 MicrophoneNorthOld = -1;
90     int16 MicrophoneSouthOld = -1;
       int16 MicrophoneWestOld = -1;
92     int16 MicrophoneEastOld = -1;
       int8 LEDNorthWestOld = -1;
94     int8 LEDNorthEastOld = -1;
       int8 LEDSouthWestOld = -1;
96     int8 LEDSouthEastOld = -1;
       for (;;) {
98         // Check for modules life and Coordinator
           for(i=0;i<MAX_MODULE_COUNT;i++) {
100            if(i != CAN_ID && ModulesInNetwork[i]) {
                   unsigned long now = millis();
102                if(ModulesLastSeen[i] + 5 *
                       IntervalDelay < now) {
                       ModulesInNetwork[i] = false;
104                }
               }
106        }
           uint8 coordinatorIndex = CAN_ID;
108        for(i=0;i<MAX_MODULE_COUNT;i++) {
               if(i != CAN_ID && ModulesInNetwork[i]) {
110                if(i > coordinatorIndex) {
                       coordinatorIndex = i;
112                }
               }
114        }
           if(coordinatorIndex == RobotModule::CAN_ID) {
116            RobotModule::IsCoordinator = true;
           } else {
118            RobotModule::IsCoordinator = false;
           }
120
```

```
122         if(IsCoordinator) {
                if(th == 20) {
124                 SelectedGame = rand() % TOTAL_GAMES;
                    th = -1;
126             }
                GameDataAvailable = true;
128 //            CoSetFlag(CAN_Game_FlagId);
            }
130         th++;

132         if(MicrophoneNorth.mine &&
            MicrophoneSouth.mine &&
134         MicrophoneWest.mine &&
            MicrophoneEast.mine ) {
136             //SoundAngleDataAvailable = true;
                MicrophoneDataAvailable = true;
138         }
            if(IRNorth.mine &&
140            IRSouth.mine &&
               IRNorthEast.mine &&
142            IRNorthWest.mine &&
               IRSouthEast.mine &&
144            IRSouthWest.mine) {
                IRDataAvailable = true;
146         }
            if(LEDNorthEast.mine &&
148            LEDNorthWest.mine &&
               LEDSouthEast.mine &&
150            LEDSouthWest.mine) {

152             int outState = digitalRead(LEDNorthEast.
                   pin);
                uint8 value;
154             if(outState == LOW) {
                    value = 0;
156             } else {
                    value = 1;
158             }
                int inState = digitalRead(LEDNorthEast.
                   pin_support);
160             if(inState == HIGH) {
                    LEDNorthEast.value = -1;
162             } else {
                    LEDNorthEast.value = value;
```

```
164             }

166             outState = digitalRead(LEDNorthWest.pin);
                if(outState == LOW) {
168                 value = 0;
                } else {
170                 value = 1;
                }
172             inState = digitalRead(LEDNorthWest.
                   pin_support);
                if(inState == HIGH) {
174                 LEDNorthWest.value = -1;
                } else {
176                 LEDNorthWest.value = value;
                }
178
                outState = digitalRead(LEDSouthEast.pin);
180             if(outState == LOW) {
                    value = 0;
182             } else {
                    value = 1;
184             }
                inState = digitalRead(LEDSouthEast.
                   pin_support);
186             if(inState == HIGH) {
                    LEDSouthEast.value = -1;
188             } else {
                    LEDSouthEast.value = value;
190             }

192             outState = digitalRead(LEDSouthWest.pin);
                if(outState == LOW) {
194                 value = 0;
                } else {
196                 value = 1;
                }
198             inState = digitalRead(LEDSouthWest.
                   pin_support);
                if(inState == HIGH) {
200                 LEDSouthWest.value = -1;
                } else {
202                 LEDSouthWest.value = value;
                }
204             Serial2.println("LED North East");
                Serial2.println(LEDNorthEast.value);
```

```
206             Serial2.println("LED North West");
                Serial2.println(LEDNorthWest.value);
208             Serial2.println("LED South East");
                Serial2.println(LEDSouthEast.value);
210             Serial2.println("LED South West");
                Serial2.println(LEDSouthWest.value);
212             LEDDataAvailable = true;
            }
214         if(Player.mine) {
                if(isConnected(MicrophoneEastOld,
                   MicrophoneEast.value) ||
216                     isConnected(MicrophoneNorthOld,
                           MicrophoneNorth.value) ||
                        isConnected(MicrophoneSouthOld,
                           MicrophoneSouth.value) ||
218                     isConnected(MicrophoneWestOld,
                           MicrophoneWest.value)) {
                    CurrentSound = MicrophoneConnected;
220                 CoSetFlag(RobotModule::Player_FlagId)
                       ;
                }
222             if(isDisconnected(MicrophoneEastOld,
                   MicrophoneEast.value) ||
                   isDisconnected(MicrophoneNorthOld,
                      MicrophoneNorth.value) ||
224                isDisconnected(MicrophoneSouthOld,
                      MicrophoneSouth.value) ||
                   isDisconnected(MicrophoneWestOld,
                      MicrophoneWest.value)) {
226                 CurrentSound = MicrophoneRemoved;
                    CoSetFlag(RobotModule::Player_FlagId)
                       ;
228             }
                MicrophoneEastOld = MicrophoneEast.value;
230             MicrophoneNorthOld = MicrophoneNorth.
                   value;
                MicrophoneSouthOld = MicrophoneSouth.
                   value;
232             MicrophoneWestOld = MicrophoneWest.value;

234             if(isConnected(LEDNorthEastOld,
                   LEDNorthEast.value) ||
                        isConnected(LEDNorthWestOld,
                           LEDNorthWest.value) ||
```

```
236                     isConnected(LEDSouthEastOld,
                            LEDSouthEast.value) ||
                        isConnected(LEDSouthWestOld,
                            LEDSouthWest.value)) {
238                 CurrentSound = LEDConnected;
                    CoSetFlag(RobotModule::Player_FlagId)
                        ;
240             }
                if(isDisconnected(LEDNorthEastOld,
                    LEDNorthEast.value) ||
242                 isDisconnected(LEDNorthWestOld,
                        LEDNorthWest.value) ||
                    isDisconnected(LEDSouthEastOld,
                        LEDSouthEast.value) ||
244                 isDisconnected(LEDSouthWestOld,
                        LEDSouthWest.value)) {
                    CurrentSound = LEDRemoved;
246                 CoSetFlag(RobotModule::Player_FlagId)
                        ;
                }
248
                LEDNorthEastOld = LEDNorthEast.value;
250             LEDNorthWestOld = LEDNorthWest.value;
                LEDSouthEastOld = LEDSouthEast.value;
252             LEDSouthWestOld = LEDSouthWest.value;
            }
254         ServoDataAvailable = true;
            WheelMotorsDataAvailable = true;
256         LifeDataAvailable = true;
            CoSetFlag(CAN_Interval_FlagId);
258         //Interval Delay
            CoTickDelay(IntervalDelay);
260     }
}
262
bool RobotModule::isConnected(int32 oldValue, int32
    newValue) {
264     return (oldValue == -1 && newValue != -1);
}
266
bool RobotModule::isDisconnected(int32 oldValue,
    int32 newValue) {
268     return (oldValue != -1 && newValue == -1);
}
270
```

```
void RobotModule::NetworkReceiveTask(void *pdata) {
    for (;;) {
//        Serial2.println("Setup Receive CAN");
        while(!canBus.available());
//        Serial2.println("Got the Data");
        CanMsg *r_msg = canBus.recv();
        if (r_msg && r_msg->DLC > 1) {
            uint8 len = r_msg->DLC;
            uint8 command = r_msg->Data[0];
            uint8 moduleId = r_msg->Data[1];
            union TwoBytes
            {
                uint16 u16;
                uint8 u8[2];
            } twoBytes;
            if(command == CAN_MICROPHONE_DATA_1) {
                twoBytes.u8[0] = r_msg->Data[2];
                twoBytes.u8[1] = r_msg->Data[3];
                MicrophoneNorth.value = twoBytes.u16;
                twoBytes.u8[0] = r_msg->Data[4];
                twoBytes.u8[1] = r_msg->Data[5];
                MicrophoneSouth.value = twoBytes.u16;
            } else if(command ==
               CAN_MICROPHONE_DATA_2) {
                twoBytes.u8[0] = r_msg->Data[2];
                twoBytes.u8[1] = r_msg->Data[3];
                MicrophoneWest.value = twoBytes.u16;
                twoBytes.u8[0] = r_msg->Data[4];
                twoBytes.u8[1] = r_msg->Data[5];
                MicrophoneEast.value = twoBytes.u16;
            } else if(command == CAN_SOUND_ANGLE_DATA
               ) {
                union FloatBytes {
                    float value;
                    unsigned char bytes[sizeof(float)
                       ];
                } floatBytes;
                for(uint8 i=0;i < sizeof(float);i++)
                   {
                    floatBytes.bytes[i] = r_msg->Data
                       [i+2];
                }
                IncomingSoundAngle.value = floatBytes
                   .value;
                Serial2.println("Sound Angle");
```

```
                Serial2.println(IncomingSoundAngle.
                    value);
            } else if(command == CAN_IR_DATA_1) {
                twoBytes.u8[0] = r_msg->Data[2];
                twoBytes.u8[1] = r_msg->Data[3];
                IRNorth.value = twoBytes.u16;
                twoBytes.u8[0] = r_msg->Data[4];
                twoBytes.u8[1] = r_msg->Data[5];
                IRSouth.value = twoBytes.u16;
                twoBytes.u8[0] = r_msg->Data[6];
                twoBytes.u8[1] = r_msg->Data[7];
                IRNorthEast.value = twoBytes.u16;
            } else if(command == CAN_IR_DATA_2) {
                twoBytes.u8[0] = r_msg->Data[2];
                twoBytes.u8[1] = r_msg->Data[3];
                IRNorthWest.value = twoBytes.u16;
                twoBytes.u8[0] = r_msg->Data[4];
                twoBytes.u8[1] = r_msg->Data[5];
                IRSouthEast.value = twoBytes.u16;
                twoBytes.u8[0] = r_msg->Data[6];
                twoBytes.u8[1] = r_msg->Data[7];
                IRSouthWest.value = twoBytes.u16;
            } else if(command == CAN_LED_DATA) {
                LEDNorthEast.value = r_msg->Data[2];
                LEDNorthWest.value = r_msg->Data[3];
                LEDSouthEast.value = r_msg->Data[4];
                LEDSouthWest.value = r_msg->Data[5];
            } else if(command ==
                CAN_WHEEL_MOTORS_DATA) {

            } else if(command ==
                CAN_SERVO_MOTORS_DATA) {

            } else if(command == CAN_LIFE_DATA) {
                moduleId = r_msg->Data[1];
            } else if(command == CAN_GAME_DATA) {
                SelectedGame = r_msg->Data[2];
            }
            ModulesInNetwork[moduleId] = true;
            ModulesLastSeen[moduleId] = millis();
            canBus.free();
        }
    }
}
```

```
352 void RobotModule::NetworkSendTask(void *pdata) {
    for (;;) {
354     StatusType err;
       OS_FlagID CANFlagId = 1 <<
          CAN_Microphone_FlagId | 1 <<
          CAN_Sound_Angle_FlagId
356                          | 1 << CAN_IR_FlagId | 1
                                << CAN_LED_FlagId | 1
                                <<
                                CAN_Wheel_Motors_FlagId

                             | 1 <<
                                CAN_Servo_Motors_FlagId
                                 | 1 <<
                                CAN_Interval_FlagId
358                          | 1 << CAN_Game_FlagId;
       CoWaitForMultipleFlags(CANFlagId,OPT_WAIT_ANY
          ,0,&err);
360    CAN_TX_MBX mbx;
       CanMsg t_msg;
362    t_msg.IDE = CAN_ID_EXT;
       t_msg.RTR = CAN_RTR_DATA;
364    if(GameDataAvailable) {
           t_msg.ID = 20;
366        t_msg.DLC = 3;
           t_msg.Data[0] = CAN_GAME_DATA;
368        t_msg.Data[1] = CAN_ID;
           t_msg.Data[2] = SelectedGame;
370        mbx = CANsend(&t_msg);
           CoClearFlag(CAN_Game_FlagId);
372        GameDataAvailable = false;
       }
374    if(LifeDataAvailable) {
           t_msg.ID = 10;
376        t_msg.DLC = 2;
           t_msg.Data[0] = CAN_LIFE_DATA;
378        t_msg.Data[1] = CAN_ID;
           mbx = CANsend(&t_msg);
380        LifeDataAvailable = false;
       }
382    if(MicrophoneDataAvailable) {
           t_msg.ID = 5;
384        t_msg.DLC = 6;
           t_msg.Data[0] = CAN_MICROPHONE_DATA_1;
386        t_msg.Data[1] = CAN_ID;
```

```
            t_msg.Data[2] = MicrophoneNorth.value & 0
                xFF;
388         t_msg.Data[3] = (MicrophoneNorth.value >>
                 8) & 0xFF;
            t_msg.Data[4] = MicrophoneSouth.value & 0
                xFF;
390         t_msg.Data[5] = (MicrophoneSouth.value >>
                 8) & 0xFF;
            mbx = CANsend(&t_msg);
392         t_msg.Data[0] = CAN_MICROPHONE_DATA_2;
            t_msg.Data[1] = CAN_ID;
394         t_msg.Data[2] = MicrophoneWest.value & 0
                xFF;
            t_msg.Data[3] = (MicrophoneWest.value >>
                8) & 0xFF;
396         t_msg.Data[4] = MicrophoneEast.value & 0
                xFF;
            t_msg.Data[5] = (MicrophoneEast.value >>
                8) & 0xFF;
398         mbx = CANsend(&t_msg);
            CoClearFlag(CAN_Microphone_FlagId);
400         MicrophoneDataAvailable = false;
            //Serial2.println("Send the Microphone
                Data CAN");
402     }
        if(IRDataAvailable) {
404         t_msg.DLC = 8;
            t_msg.Data[0] = CAN_IR_DATA_1;
406         t_msg.Data[1] = CAN_ID;
            t_msg.Data[2] = IRNorth.value & 0xFF;
408         t_msg.Data[3] = (IRNorth.value >> 8) & 0
                xFF;
            t_msg.Data[4] = IRSouth.value & 0xFF;
410         t_msg.Data[5] = (IRSouth.value >> 8) & 0
                xFF;
            t_msg.Data[6] = IRNorthEast.value & 0xFF;
412         t_msg.Data[7] = (IRNorthEast.value >> 8)
                & 0xFF;
            mbx = CANsend(&t_msg);
414         t_msg.Data[0] = CAN_IR_DATA_2;
            t_msg.Data[1] = CAN_ID;
416         t_msg.Data[2] = IRNorthWest.value & 0xFF;
            t_msg.Data[3] = (IRNorthWest.value >> 8)
                & 0xFF;
418         t_msg.Data[4] = IRSouthEast.value & 0xFF;
```

```
            t_msg.Data[5] = (IRSouthEast.value >> 8)
                & 0xFF;
            t_msg.Data[6] = IRSouthWest.value & 0xFF;
            t_msg.Data[7] = (IRSouthWest.value >> 8)
                & 0xFF;
            mbx = CANsend(&t_msg);
            CoClearFlag(CAN_IR_FlagId);
            IRDataAvailable = false;
            //Serial2.println("Send the IR Data CAN")
                ;
        }
        if(LEDDataAvailable) {
            t_msg.DLC = 6;
            t_msg.Data[0] = CAN_LED_DATA;
            t_msg.Data[1] = CAN_ID;
            t_msg.Data[2] = LEDNorthEast.value;
            t_msg.Data[3] = LEDNorthWest.value;
            t_msg.Data[4] = LEDSouthEast.value;
            t_msg.Data[5] = LEDSouthWest.value;
            CoClearFlag(CAN_LED_FlagId);
            LEDDataAvailable = false;
            mbx = CANsend(&t_msg);
            //Serial2.println("Send the LED Data CAN
                ");
        }
        if(SoundAngleDataAvailable) {
            union FloatBytes {
                float value;
                unsigned char bytes[sizeof(float)];
            } floatBytes;
            floatBytes.value = IncomingSoundAngle.
                value;
            t_msg.DLC = sizeof(float) + 2;
            t_msg.Data[0] = CAN_SOUND_ANGLE_DATA;
            t_msg.Data[1] = CAN_ID;
            for(uint8 i=0;i < sizeof(float);i++) {
                t_msg.Data[i+2] = floatBytes.bytes[i
                    ];
            }
            CoClearFlag(CAN_Sound_Angle_FlagId);
            SoundAngleDataAvailable = false;
            mbx = CANsend(&t_msg);
            //Serial2.println("Send the Sound Angle
                Data CAN");
        }
```

```
        CoClearFlag(CAN_Interval_FlagId);
458         CoTickDelay(50);
    }
460 }

462
void RobotModule::GamePerformTask(void *pdata) {
464     int oldValue_NE = IRNorthEast.value;
    int oldValue_SW = IRSouthWest.value;
466     int oldValue_NW = IRSouthEast.value;
    int oldValue_SE = IRNorthWest.value;
468     float oldAngle = IncomingSoundAngle.value;
    bool clapped = false;
470     while(1) {
//         Serial2.println("Selected Game");
472
        if(SelectedGame == GAME_RELAX) {
474             clapped = false;
            oldAngle = IncomingSoundAngle.value;
476             if (Player.mine && (!IsPlayingMP3 ||
                CurrentSound != MusicSound)) {
                 IsPlayingMP3 = false;
478                 CurrentSound = MusicSound;
                 CoSetFlag(RobotModule::Player_FlagId)
                     ;
480             }
            if (LEDNorthEast.mine) {
482                 digitalWrite(LEDNorthEast.pin, HIGH);
            }
484             if (LEDSouthWest.mine) {
                 digitalWrite(LEDSouthWest.pin, HIGH);
486             }
            if (LEDNorthWest.mine) {
488                 digitalWrite(LEDNorthWest.pin, LOW);
            }
490             if (LEDSouthEast.mine) {
                 digitalWrite(LEDSouthEast.pin, LOW);
492             }
            CoTickDelay(250);
494             if (LEDNorthEast.mine) {
                 digitalWrite(LEDNorthEast.pin, LOW);
496             }
            if (LEDSouthWest.mine) {
498                 digitalWrite(LEDSouthWest.pin, LOW);
            }
```

```
500             if (LEDNorthWest.mine) {
                    digitalWrite(LEDNorthWest.pin, HIGH);
502             }
                if (LEDSouthEast.mine) {
504                 digitalWrite(LEDSouthEast.pin, HIGH);
                }
506

508             if (abs(RobotModule::IRNorthEast.value -
                    oldValue_NE) > 250) {
                    SelectedGame = GAME_ESCAPE;
510             }
                oldValue_NE = RobotModule::IRNorthEast.
                    value;
512
                if (abs(RobotModule::IRSouthWest.value -
                    oldValue_SW) > 250) {
514                 SelectedGame = GAME_ESCAPE;
                }
516             oldValue_SW = RobotModule::IRSouthWest.
                    value;
                if (abs(RobotModule::IRSouthEast.value -
                    oldValue_SE) > 250) {
518                 SelectedGame = GAME_ESCAPE;
                }
520             oldValue_SE = RobotModule::RobotModule::
                    IRSouthEast.value;

522             if (abs(RobotModule::IRNorthWest.value -
                    oldValue_NW) > 250) {
                    SelectedGame = GAME_ESCAPE;
524             }
                oldValue_NW = RobotModule::IRNorthWest.
                    value;
526
            } else if(SelectedGame == GAME_HAPPY) {
528             if(!clapped) {
                    if (Player.mine && IsPlayingMP3) {
530                     IsPlayingMP3 = false;
                    }
532                 if (LEDNorthEast.mine) {
                        digitalWrite(LEDNorthEast.pin,
                            LOW);
534                 }
                    if (LEDSouthWest.mine) {
```

```
536                     digitalWrite(LEDSouthWest.pin,
                            LOW);
                    }
538                 if (LEDNorthWest.mine) {
                        digitalWrite(LEDNorthWest.pin,
                            LOW);
540                 }
                    if (LEDSouthEast.mine) {
542                     digitalWrite(LEDSouthEast.pin,
                            LOW);
                    }
544                 if(oldAngle != IncomingSoundAngle.
                        value) {
                        clapped = true;
546                 }
                } else {
548                 if (Player.mine && (!IsPlayingMP3 ||
                        CurrentSound != Despacito)) {
                        IsPlayingMP3 = false;
550                     CurrentSound = Despacito;
                        CoSetFlag(RobotModule::
                            Player_FlagId);
552                 }
                    if (LEDNorthEast.mine) {
554                     digitalWrite(LEDNorthEast.pin,
                            HIGH);
                    }
556                 if (LEDSouthWest.mine) {
                        digitalWrite(LEDSouthWest.pin,
                            HIGH);
558                 }
                    if (LEDNorthWest.mine) {
560                     digitalWrite(LEDNorthWest.pin,
                            HIGH);
                    }
562                 if (LEDSouthEast.mine) {
                        digitalWrite(LEDSouthEast.pin,
                            HIGH);
564                 }
                    CoTickDelay(1000);
566                 if (LEDNorthEast.mine) {
                        digitalWrite(LEDNorthEast.pin,
                            LOW);
568                 }
                    if (LEDSouthWest.mine) {
```

```
570                     digitalWrite(LEDSouthWest.pin,
                            LOW);
                    }
572                 if (LEDNorthWest.mine) {
                        digitalWrite(LEDNorthWest.pin,
                            LOW);
574                 }
                    if (LEDSouthEast.mine) {
576                     digitalWrite(LEDSouthEast.pin,
                            LOW);
                    }
578             }

580             if (abs(RobotModule::IRNorthEast.value -
                    oldValue_NE) > 250) {
                     SelectedGame = GAME_ESCAPE;
582             }
                oldValue_NE = RobotModule::IRNorthEast.
                    value;
584
                if (abs(RobotModule::IRSouthWest.value -
                    oldValue_SW) > 250) {
586                  SelectedGame = GAME_ESCAPE;
                }
588             oldValue_SW = RobotModule::IRSouthWest.
                    value;
                if (abs(RobotModule::IRSouthEast.value -
                    oldValue_SE) > 250) {
590                  SelectedGame = GAME_ESCAPE;
                }
592             oldValue_SE = RobotModule::RobotModule::
                    IRSouthEast.value;

594             if (abs(RobotModule::IRNorthWest.value -
                    oldValue_NW) > 250) {
                     SelectedGame = GAME_ESCAPE;
596             }
                oldValue_NW = RobotModule::IRNorthWest.
                    value;
598         } else if(SelectedGame == GAME_ESCAPE) {
                clapped = false;
600             oldAngle = IncomingSoundAngle.value;
                if (abs(RobotModule::IRNorthEast.value -
                    oldValue_NE) > 250) {
```

```
602                 if (MotorLeft.mine && MotorRight.mine
                        ) {
                        MotorLeft.setSpeed(20000);
604                     MotorRight.setSpeed(50000);
                        MotorLeft.setControl(
                            MOTOR_BACKWARD);
606                     MotorRight.setControl(
                            MOTOR_BACKWARD);
                    }
608                 if (LEDNorthEast.mine) {
                        digitalWrite(LEDNorthEast.pin,
                            HIGH);
610                 }
                    if (Player.mine) {
612                     IsPlayingMP3 = false;
                        CurrentSound =
                            NorthEastDangerSound;
614                     CoSetFlag(RobotModule::
                            Player_FlagId);
                    }
616                 CoTickDelay(1000);
                    if (MotorLeft.mine && MotorRight.mine
                        ) {
618                     MotorLeft.setControl(MOTOR_STOP);
                        MotorRight.setControl(MOTOR_STOP)
                            ;
620                 }
                    if (LEDNorthEast.mine) {
622                     digitalWrite(LEDNorthEast.pin,
                            LOW);
                    }
624             }
                oldValue_NE = RobotModule::IRNorthEast.
                    value;
626
                if (abs(RobotModule::IRSouthWest.value -
                    oldValue_SW) > 250) {
628                 if (MotorLeft.mine && MotorRight.mine
                        ) {
                        MotorLeft.setSpeed(50000);
630                     MotorRight.setSpeed(20000);
                        MotorLeft.setControl(
                            MOTOR_FORWARD);
632                     MotorRight.setControl(
                            MOTOR_FORWARD);
```

```
            }
            if (LEDSouthWest.mine) {
                digitalWrite(LEDSouthWest.pin,
                    HIGH);
            }
            if (Player.mine) {
                IsPlayingMP3 = false;
                CurrentSound =
                    SouthWestDangerSound;
                CoSetFlag(RobotModule::
                    Player_FlagId);
            }
            CoTickDelay(1000);
            if (MotorLeft.mine && MotorRight.mine
                ) {
                MotorLeft.setControl(MOTOR_STOP);
                MotorRight.setControl(MOTOR_STOP)
                    ;
            }
            if (LEDSouthWest.mine) {
                digitalWrite(LEDSouthWest.pin,
                    LOW);
            }
        }
        oldValue_SW = RobotModule::IRSouthWest.
            value;

        if (abs(RobotModule::IRSouthEast.value -
            oldValue_SE) > 250) {
            if (MotorLeft.mine && MotorRight.mine
                ) {
                MotorLeft.setSpeed(20000);
                MotorRight.setSpeed(50000);
                MotorLeft.setControl(
                    MOTOR_FORWARD);
                MotorRight.setControl(
                    MOTOR_FORWARD);
            }
            if (LEDSouthEast.mine) {
                digitalWrite(LEDSouthEast.pin,
                    HIGH);
            }
            if (Player.mine) {
                IsPlayingMP3 = false;
```

```
                CurrentSound =
                    SouthEastDangerSound;
666             CoSetFlag(RobotModule::
                    Player_FlagId);
            }
668         CoTickDelay(1000);
            if (MotorLeft.mine && MotorRight.mine
                ) {
670             MotorLeft.setControl(MOTOR_STOP);
                MotorRight.setControl(MOTOR_STOP)
                    ;
672         }
            if (LEDSouthEast.mine) {
674             digitalWrite(LEDSouthEast.pin,
                    LOW);
            }
676     }
        oldValue_SE = RobotModule::RobotModule::
            IRSouthEast.value;
678
        if (abs(RobotModule::IRNorthWest.value -
            oldValue_NW) > 250) {
680         if (MotorLeft.mine && MotorRight.mine
                ) {
                MotorLeft.setSpeed(50000);
682             MotorRight.setSpeed(20000);
                MotorLeft.setControl(
                    MOTOR_BACKWARD);
684             MotorRight.setControl(
                    MOTOR_BACKWARD);
            }
686         if (LEDNorthWest.mine) {
                digitalWrite(LEDNorthWest.pin,
                    HIGH);
688         }
            if (Player.mine) {
690             IsPlayingMP3 = false;
                CurrentSound =
                    NorthWestDangerSound;
692             CoSetFlag(RobotModule::
                    Player_FlagId);
            }
694         CoTickDelay(1000);
            if (MotorLeft.mine && MotorRight.mine
                ) {
```

```
696                     MotorLeft.setControl(MOTOR_STOP);
                        MotorRight.setControl(MOTOR_STOP)
                            ;
698                 }
                    if (LEDNorthWest.mine) {
700                     digitalWrite(LEDNorthWest.pin,
                            LOW);
                    }
702             }
                oldValue_NW = RobotModule::IRNorthWest.
                    value;
704         }
            CoTickDelay(250);
706     }
}
708
void RobotModule::PrintHex(uint8 b)
710 {
    static char digits[] = "0123456789ABCDEF";
712     Serial2.print(digits[b >> 4]);
    Serial2.print(digits[b & 0xf]);
714 }

716 CAN_TX_MBX RobotModule::CANsend(CanMsg *pmsg)
{
718     CAN_TX_MBX mbx;

720     do
            mbx = canBus.send(pmsg) ;
722     while(mbx == CAN_TX_NO_MBX) ;
    return mbx ;
724 }
```

E

APPENDIX E: MOTOR LIBRARY SOURCE CODE

Listing E.1: Motor Library Header

```
#ifndef MOTOR_LIBRARY_H
#define MOTOR_LIBRARY_H
#include <HardwareCAN.h>
#include <MapleCoOS116.h>
class Motor {
public:
    Motor(uint16 def);
    void setSpeed(uint16 speed);
    void setControl(uint8 dir);
    bool mine;
    volatile uint16 speed;
    volatile uint8 dir;
    uint8 pin_pwm;
    uint8 pin_in_1;
    uint8 pin_in_2;
    uint8 pin_standby;
};

#endif
```

Listing E.2: Motor Library Source code

```
#include "motor.h"
#include "module.h"

Motor::Motor(uint16 def) {
    mine = false;
    speed = def;
    pin_pwm = def;
    pin_in_1 = def;
    pin_in_2 = def;
    pin_standby = def;
    dir = 0;
};
```

```
void Motor::setSpeed(uint16 speed) {
    this->speed = speed;
    pwmWrite(this->pin_pwm, speed);
}

void Motor::setControl(uint8 dir) {
    this->dir = dir;
    if(dir == RobotModule::MOTOR_FORWARD) {
        digitalWrite(pin_in_1, HIGH);
        digitalWrite(pin_in_2, LOW);
    } else if(dir == RobotModule::MOTOR_BACKWARD){
        digitalWrite(pin_in_1, LOW);
        digitalWrite(pin_in_2, HIGH);
    } else {
        digitalWrite(pin_in_1, LOW);
        digitalWrite(pin_in_2, LOW);
    }
}
```