

# AUDIO COMMUNICATION FOR MULTI-ROBOT SYSTEMS

by

Pooya Karimian

B.Sc., Sharif University of Technology, 2003

A THESIS SUBMITTED IN PARTIAL FULFILLMENT  
OF THE REQUIREMENTS FOR THE DEGREE OF  
MASTER OF SCIENCE  
in the School  
of  
Computing Science

© Pooya Karimian 2007  
SIMON FRASER UNIVERSITY  
2007

All rights reserved. This work may not be  
reproduced in whole or in part, by photocopy  
or other means, without the permission of the author.

## APPROVAL

**Name:** Pooya Karimian  
**Degree:** Master of Science  
**Title of thesis:** Audio Communication for Multi-Robot Systems

**Examining Committee:** Dr. Anoop Sarkar, Professor  
School of Computing Science  
Chair

---

Dr. Richard Vaughan, Professor  
School of Computing Science  
Senior Supervisor

---

Dr. Greg Mori, Professor  
School of Computing Science  
Supervisor

---

Dr. Mohamed Hefeeda, Professor  
School of Computing Science  
Examiner

**Date Approved:** \_\_\_\_\_

# Abstract

Interaction through communication is an important aspect of multi-robot systems. Audio communication, while common among animals and well studied in biology, is not well explored as a multi-robot technology. In this thesis we study the use of audio messages as a means of communication among mobile robots. We examine the properties of audio compared to other communication media, and show how these can be exploited.

To guide the design of a multi-robot system, a simple audio propagation model integrated into a robot simulator is developed. This simulator shows how acoustic communication improves the team performance in a prototypical search task.

We also introduce a physical network layer that uses audio as the transmission medium and implements a broadcast method related to the CSMA protocol. We then describe a distributed mutual exclusion algorithm suitable for use over audio, and demonstrate it in both simulation and real-world.

**Keywords:** Robotics, Autonomous Robots, Intelligent Control Systems, Distributed Systems, Audio Communication.

*To my parents*

# Acknowledgments

I would like to thank Dr. Richard Vaughan. He is a wonderful supervisor. He provided encouragement, support, and lots of good ideas from the very first semester I was here at the Simon Fraser University.

This work wouldn't be possible without the facilities provided by the Autonomy Lab and the help of my colleagues at the lab. I would like to thank Jens Wawerla for his help especially in the Chatterbox project.

Thanks to all members of the thesis committee for their interest and time. Dr. Greg Mori has been an invaluable source of knowledge both during his course and when I was his research assistant.

Dr. Torsten Möller guided me to finish the course project discussed in Section 2.5 in a short time and with good results. Dr. Tamara Smyth and Dr. Daniel Weiskopf provided me with helpful comments on that project.

The Open-Source software was an essential part of my project. Thanks to the Player and Stage projects and the open community behind the Gumstix boards. I am thankful to Jesús Arias for releasing his RTTY software on which my Nava module is based.

Finally, I wish to thank my family: my parents for their support even from far away on the other side of the world and my sister, Roya, for her encouragement.

# Contents

<b>Approval</b>	<b>ii</b>
<b>Abstract</b>	<b>iii</b>
<b>Dedication</b>	<b>iv</b>
<b>Acknowledgments</b>	<b>v</b>
<b>Contents</b>	<b>vi</b>
<b>List of Tables</b>	<b>ix</b>
<b>List of Figures</b>	<b>x</b>
<b>List of Programs</b>	<b>xiv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Goal . . . . .	1
1.2 Audio in Human and Animals . . . . .	2
1.3 Audio in Technology . . . . .	2
1.4 Communication in Robotics . . . . .	3
1.5 Why Audio? . . . . .	4
1.6 Contributions . . . . .	6
1.7 Thesis Outline . . . . .	7
<b>2 Modeling Audio Signals</b>	<b>9</b>
2.1 Audio Propagation . . . . .	9

2.2	Shortest Path Model . . . . .	10
2.2.1	Pros . . . . .	11
2.2.2	Cons . . . . .	12
2.3	Implementation . . . . .	12
2.3.1	Complexity Analysis . . . . .	15
2.4	Stage model . . . . .	19
2.5	Complex Model . . . . .	23
<b>3</b>	<b>Sounds Good: Evaluation of Audio Communication</b>	<b>28</b>
3.1	Task Definition . . . . .	29
3.2	System . . . . .	30
3.3	Implementation . . . . .	33
3.3.1	Frontier Based Exploration . . . . .	34
3.3.2	Local Map . . . . .	35
3.3.3	Using Audio Information . . . . .	35
3.3.4	Path Planning in a Local Map . . . . .	38
3.4	Experiment . . . . .	40
3.5	Results . . . . .	42
3.6	Discussion . . . . .	45
<b>4</b>	<b>Going to the Real World</b>	<b>46</b>
4.1	Sounds Good in the Real World . . . . .	46
4.2	Audio Communication in Robotics . . . . .	47
4.3	Concurrent Computing . . . . .	47
4.3.1	Mutual Exclusion . . . . .	48
4.3.2	Local Mutual Exclusion . . . . .	48
4.3.3	Distributed Coordination . . . . .	49
4.4	Final Choice . . . . .	49
4.5	Message Passing Communication . . . . .	49
<b>5</b>	<b>Nava: Audio Communication Layer</b>	<b>50</b>
5.1	RTTY Communication . . . . .	50
5.1.1	Modulation . . . . .	51
5.1.2	Demodulation . . . . .	52

5.2	Broadcast over Audio . . . . .	53
5.2.1	Network Layer . . . . .	53
5.2.2	Collisions . . . . .	53
5.2.3	Unreliable Broadcast . . . . .	54
5.2.4	Implementation . . . . .	54
5.3	Player Driver . . . . .	57
<b>6</b>	<b>Mutual Exclusion for Robots</b>	<b>58</b>
6.1	Mutual Exclusion . . . . .	58
6.2	Mutual Exclusion for Robots . . . . .	58
6.2.1	Physical Object . . . . .	58
6.2.2	Lock Assignment Authority . . . . .	59
6.2.3	Token Passing in a Ring . . . . .	60
6.2.4	Logical Clocks . . . . .	61
6.3	Mutual Exclusion over Audio . . . . .	64
6.3.1	Our Algorithm . . . . .	65
6.3.2	Analysis . . . . .	66
<b>7</b>	<b>Local Mutual Exclusion Demonstration</b>	<b>69</b>
7.1	Implementation . . . . .	69
7.2	Application: Charging . . . . .	71
7.2.1	Simulation . . . . .	75
7.2.2	Real-World Demonstration . . . . .	78
<b>8</b>	<b>Conclusions and Future Work</b>	<b>84</b>
8.1	Summary . . . . .	84
8.2	Future Work . . . . .	85
8.2.1	Hybrid Communication . . . . .	85
8.2.2	Bio-inspired Communication . . . . .	85
8.2.3	Modern Network Protocols on Audio . . . . .	86
8.2.4	Sound Signature . . . . .	86
8.3	Final Word . . . . .	87
	<b>Bibliography</b>	<b>88</b>

# List of Tables

3.1	Mean and standard deviation of time to finish an experiment using different types of audio sensors and on different starting configurations. The numbers are in seconds. Each mean and deviation is calculated using 20 different experiments. . . . .	40
3.2	Two-tailed student T-test results. Each row shows the comparison of a method pair on one configuration. Column “gain” is the performance gain percentage of method 2 over method 1, and is calculated by dividing the means minus one. Column “Different” shows whether according to the t-Test the results were statistically different with a 95% confidence or not. . . . .	44
6.1	Actions and events for agents requesting the lock in the audio-based mutual exclusion method in comparison to Ricart-Agrawala algorithm. . . . .	66
7.1	The request pair and the time lock is granted for the three robots of the simulation trial depicted in Figure 7.5(b). . . . .	76
7.2	The request pair and the time lock is granted for the three robots of the real-world trial depicted in Figure 7.10(b). . . . .	82

# List of Figures

1.1	System architecture of a multi-robot system running (a) in simulation with robot controller programs using the simulator to model the virtual robots and audio communication between them (b) in real-world with robots equipped with a physical audio message transmission modem. . . . .	6
2.1	Audio propagation in an environment can be complex and at the same time exhibit regular properties that can be used by robotic systems. . . . .	10
2.2	The simple model simulates audio propagation as the shortest path between the sound source and the destination. This is the path the direct and the most powerful sound will take. . . . .	11
2.3	The shortest-path audio model models the large difference in the received signal intensity in the scenarios depicted in (a) and (b). In this figure, solid lines are walls, circles are the sender and receiver robots locations and dotted lines are shortest audio paths. . . . .	12
2.4	These are the steps to build a visibility graph (a) of a map and its set of polygonal obstacles. (b) The obstacle vertices form the graph nodes. (c) & (d) An edge is added for every pair vertices that are mutually visible without hitting an obstacle. . . . .	13
2.5	A sample set of masks in which their center point cannot be a diffraction point and their corresponding integer representation . . . . .	15
2.6	The steps to find the shortest path between two nodes in a bitmap using diffraction points and visibility graph . . . . .	16

2.7	An example of a bitmap and a set of diffraction points (small green dots near the walls). These three robots (hexagons) are sending audio signals and the lines show the calculated shortest audio paths. Note the difference in the number of diffraction points of different map types in (a) and (b). . . . .	17
2.8	The audio propagation model calculates the shortest-path distance between the virtual robots in the simulated world and dispatch messages between robot controller programs that are connected as clients to Player. The audio model can be implemented in two ways: (a) The initial version connected a client to Player. (b) The new model integrated into Stage. . . . .	21
2.9	Ray tracing in Stage using a quad-tree matrix for speed-up. . . . .	22
2.10	Player/Stage modeling the audio message paths between four robots. . . . .	22
2.11	Diffraction of audio waves around obstacle edges matches the shortest-path simplification of audio model. (a) Waves passing through a narrow slit. (b) Diffraction behind a wall. (c) Diffraction around an obstacle. . . . .	24
2.12	Two wooden blocks placed in a circular ripple tank with a slit between them, creating circular waves. Beneath the ripple tank was a sheet of white paper, where the wave patterns appeared due to a light source above the ripple tank. ©Armed Blowfish. Used by permission under the BSD license. . . . .	25
2.13	Huygens-Fresnel principle analyzes how waves are diffracted using a sum of small secondary waves along the advancing wave front. Compare this Huygens-Fresnel analyzed model with a real photo of wave diffraction shown in Figure 2.12. ©Arne Nordmann. Used by permission. . . . .	26
2.14	A complex audio propagation model, modeling the direct sound, reflection and diffraction of audio waves. Such a model can provide high accuracy but will be computationally expensive. . . . .	27
3.1	A schematic of the examined resource transportation task for a single robot. The robot starts by searching for the source, loading a virtual resource, searching for the sink and then unloading. From the start to when the unloading finishes two completed jobs will be counted. . . . .	29
3.2	Prototype of the Chatterbox, a small robot, running Linux and equipped with different types of sensors and emitters. . . . .	31
3.3	Occupancy grid and frontier based exploration. . . . .	34

3.4	(a) <i>Local map</i> : an occupancy grid built by a robot over a period of time. In this image, <i>white</i> shows empty area, <i>black</i> shows known obstacles (enlarged by the robot size), and <i>gray</i> shows unknown area. (b) Traversibility map: the darker the cell color, the harder going to that cell is (c) Potential field map: with zero at the target frontier and growing for neighbor cells (d) The path the robot takes to reach the target by following the steepest gradient in the potential field. . . . .	39
3.5	Randomly generated initial configurations in a partial hospital floor plan (“hospital_section” in Stage). Each map is $34 \times 14$ square meters. The robots are the small circles each $15\text{cm}$ in diameter. The markers, shown as boxes, are resource locations. (a) Initial configuration #01 (b) Initial configuration #09. . . . .	41
3.6	The mean and 95% confidence interval time (in seconds) to (a) finish each job in one of the initial configurations (configuration number 9) (b) finish all 20 jobs in all configurations. The configurations are reordered by the time of ”no audio” method for the sake of clarity. . . . .	43
5.1	Signal levels for two 8N1 bytes sent using RS-232 standard. There are one start bit, eight bits of data, no parity and one stop bit. . . . .	51
5.2	Block diagram for demodulation in RTTY program. ©Jesús Arias. Used by permission. . . . .	52
5.3	The state machine implementing the Nava audio communication layer . . . .	55
5.4	Nava packet format . . . . .	56
6.1	Single server mutual exclusion: a single central authority grants lock to the requesting clients in order of their request times. . . . .	59
6.2	Achieving mutual exclusion with token passing. The node which has the token can grab the lock. . . . .	60
6.3	Lamport logical clocks use happened-before relation between events to time-stamp them. . . . .	62
6.4	The Ricart-Agrawala algorithm for mutual exclusion uses messages that are time-stamped with Lamport clock and the node id: $(T_i, i)$ . (a) $n_2$ and $n_3$ request for lock. (b) $n_1$ replies to both. $n_2$ replies to $n_3$ but $n_2$ defers the reply and grabs the lock. . . . .	63

7.1	The format of four-byte audio message of mutual exclusion experiment. . . .	69
7.2	The state machine describing the possible states of the local mutual exclusion method. The text under the line inside the states is the action that is performed when entering a state. There are two separate timers to Wanted and Silent states. . . . .	74
7.3	System architecture implementing the charging application. The robot controller programs are similar software running on each robot running both the navigation behaviors plus the mutual exclusion algorithm (a) in simulation (b) in real-world. . . . .	75
7.4	Charging application using mutual exclusion in simulation. (a) Two robots communicating to access the charger. (b) A robot is charging while two other robots are waiting for the access. . . . .	76
7.5	The transmitted messages for mutual exclusion algorithm logged from a simulation run similar to Figure 7.4. ↓ specifies a sent message in $(Type, T_i, i)$ format. ↑ is a receive event. The numbers in brackets are logical clocks. Thin lines are when the robot is requesting the lock and thick is when it holds the lock. Dashes lines are when the robot is silent. (a) Only one robot is requesting the lock. (b) Three robots negotiate to get the lock. . . . .	77
7.6	iRobot Create Programmable Robot. . . . .	78
7.7	Home Base IR beams . . . . .	79
7.8	Gumstix, Wifistix, Roboaudio-TH, Microphone and Speaker. . . . .	80
7.9	iRobot Create robots communicate to decide which one gets access to the charger. . . . .	80
7.10	The transmitted messages for mutual exclusion algorithm logged from the real experiment shown in Figure 7.9. ↓ specifies a sent message in $(Type, T_i, i)$ format. ↑ is a receive event. × is receiving noise or a corrupted message. The numbers in brackets are logical clocks. Thin lines are when the robot is requesting the lock and thick is when it holds the lock. Dashes lines are when the robot is silent. (a) Only one robot is requesting the lock. (b) Three robots negotiate to get the lock. . . . .	81

# List of Programs

2.1	Pseudo-code implementation of the shortest-path audio model. . . . .	15
7.1	The variables and functions needed for the implementation of the local mutual algorithm shown in Program 7.2. . . . .	72
7.2	The pseudo-code implementation of the local mutual exclusion algorithm. The global variables and the functions need by this algorithm are defined in Program 7.1 . . . . .	73

# Chapter 1

## Introduction

### 1.1 Goal

The homogeneous and autonomous agents of a multi-robot system can perform complicated tasks through interacting and communicating with each other. Audio, as one of the communication media used by animals and humans, has been utilized in robotic applications before. But in multi-robot systems, compared to other communication methods, audio communication is not much studied. Some properties of audio make it worthwhile to research. For example, the relatively short transmission range of audio leads to scalability for large numbers of robots. Also the interaction of sound waves with the environment can provide information about surroundings to an autonomous robot.

By showing how well audio communication fits in a multi-robot system, the goals of this thesis are:

- Study the viability of using audio communication for mobile robots.
- Study the complex properties of sound waves and show how they can be exploited in a simple manner.
- Discuss the drawbacks and the benefits of using the sound for communication in comparison to the more common methods already in use.
- Show how a simple propagation model can be used to kick-start the research in this field.
- Develop a model of audio-based communication and share it with other researchers.

- Develop an audio communication protocol using the concepts currently used in computer networks.
- Demonstrate the use of audio communication in simulated and real world tasks.

## 1.2 Audio in Human and Animals

Audio is defined as sound signals with periodic vibrations at human-audible frequencies. These frequencies for a normal healthy human are between 20 and 20,000 Hertz. Audio waves are mechanical longitudinal waves that propagate through different media such as air and water. They are generated by movement of a part in the sound source and are sensed by the vibration they cause in the sensor.

Audio frequencies sensible by humans and species of animals are not the same and audio signaling is extensively exploited by them for different means. The most obvious example is how humans use speech to communicate with each other as an important part of their lives. We spend enormous amounts of time listening to music and studying music as a hobby. The impact of sound and music on human feelings and behavior is well-known.

Many non-human mammals, birds and insects use sound in spectacular ways. Owls use inter-aural time differences to localize the audio source with high precision and use it to hunt prey [27]. Bats use *echolocation*, by emitting high-pitched sounds and listening to the echoes, to gather information about the objects around them [41]. Dolphins recognize individuals and address each other by whistling [22]. Other animals use audio to communicate with each other or to solve territorial disputes [33].

## 1.3 Audio in Technology

These days many electronic devices are equipped with audio emitters and sensors. Radios, cell phones, and most computers have sound capabilities. To sense and emit sound signals on a computer with a sound card, microphones and speakers are used. These emitters and sensors are generally designed to cover all or part of the human-audible frequencies.

In robotics, audio sensors are occasionally used but are not as popular as other sensor types. They have been used both for passively sensing the environment and as a means of communication.

As a simple and easily understandable communication mechanism, robots can use audio to interact with humans. By use of simple beepers and buzzers as common debugging tools, humans can perceive internal state of the robot controller over several meters and even without line of sight.

Social, entertainment and service robots use speech synthesizers to make human understandable sounds [47]. Speech recognizers are used to let the humans control the robots just by speaking to them [4].

## 1.4 Communication in Robotics

Sometimes the goal of an intelligent system is defined as a point where a human communicating with an intelligent system could not distinguish the system from a real human [50]. This suggests the importance of studying the use of the same communication medium as the one humans use for intelligent agents' communication.

The use of audio as a robot-to-robot communication medium is something that is not well studied before. The most common means of communication between robots is through wireless data links. Communicating with the use of radio frequencies has the advantages of being robust, fast and long-range. Nowadays, wireless communications modules installed on cellular phones and handheld devices are cheap, compact and power efficient enough to be used in every robot. They use standard protocols such as Bluetooth and IEEE 802.11. Their spread spectrum capabilities let them be used in usually large numbers in the same environment.

Another class of sensor used for communication in robotics is line-of-sight methods. Line-of-sight communication can be implemented using infrared signaling. *Infrared Data Association (IrDA)* provides fast and high-bandwidth data transfers over short distances with a direct line-of-sight. Infrared communication is fast enough to be used for video transmission. Visible light and laser have also been used. Researchers at NASA proposed an optical data link to the Mars Telecommunications Orbiter using line-of-sight laser communication [5].

The use of audio waves in communication has been tried out a few times before. Its unreliability and complex propagation behaviors have often ruled it out in favor of other communication media. But there are some properties of sound that make it unique and of great use in sensor network and so in distributed robotic systems. For example, Girod in

his PhD research [15] developed a system of acoustic sensor arrays that use a combination of wireless and audio communication to estimate mutual distance.

It is through interaction and communication between homogeneous multi-agents that they can distribute a task among themselves [45]. The transmission of the information between robots allows the organization of behaviors and management of resources. For example, by broadcasting an alarm call to a group, robots can coordinate in a task. And by interacting, distant agents can distribute the needed actions or establish territories.

## 1.5 Why Audio?

There are some interesting properties of sound which may make it attractive as an alternative or complementary medium for robot-robot communication. **Locality** of audio signals is one of them. When receiving an audio signal we know that the source is somewhere near us. Also having information about the emitting power from the sound source and having the ability to measure the intensity of the received sound lets us estimate this distance quantitatively.

Unlike light and infrared-based systems, in audio signals there is **no need for line-of-sight**. Audio propagates around obstacles and reaches the listeners as long as they are near enough to the source.

Audio signals form an **intensity gradient** as they propagate away from their source. This gradient starts from a powerful signal near the sound source and weakens with the square of the distance the farther it traverses. Having this gradient means that a robot with the ability to detect sound level can estimate the relative distance from the source and the direction of the sound, provided that it has an estimate of the emission intensity. In many environments, such as an office building, the intensity gradient closely follows the traversable space for a robot. Further, the steepest intensity gradient generally takes the shortest path from source to the robot.

Huang [19] showed sound-based servoing for mobile robots to localize a sound source. Østergaard [36] showed that even with a single microphone an audio alarm signal has a detectable gradient which can be used to track down the path toward the sound source. They used these audio signals to help solve a multiple-robot-multiple-task allocation problem.

When audio propagates through the environment it interacts with obstacles surrounding the empty spaces it passes through. The waves propagate around obstacles by diffraction and by reflection from surfaces. The energy of the reflected waves is dependent on the

material they are reflected from. This interaction of sound waves and the robot-traversable environment means that useful environmental information can be obtained from a received audio signal in addition to information encoded into the signal by its producer.

Most or all of the properties mentioned for audio can be said to be common among all types of media that use waves for communication. But the following points make audio an interesting choice for robot communication in our opinion:

- Robot-scale physical interaction of audio with the environment: Infrared is mostly a line-of-sight only sensor. Wireless waves easily transmit through some obstacles with a hard to distinguish difference between open-space and transmissive obstacles. Sound waves can be considered to be somewhere between infrared light and WiFi signals in terms of their physical interaction with the environment.
- A biologically inspired way of communication: The audio communication between the robots can be observed by humans. It can also be used to interact with humans and animals and at the same time with other robots.
- Easy access to directional sensors: Conventional WiFi antennae that come with most of the electronic devices are omni-directional. Directional microphones are standard and cheap sensors available in many devices. Directional sensors can be used to sense the steepest gradient direction.
- Availability: Wireless communication under water is hard. Acoustic signaling is known to be a better choice for underwater communication [44]. Currently the WiFi communication modules are expensive compared to sound devices. Sound sensors may already be available because of other reasons for example for interacting with humans.

Still our goal is not to compete with wireless communication. WiFi, as a fast and reliable way of communication, is getting more and more popular in robotics. Cheap wireless modules are now available and more robots are being equipped with this type of communication. But we believe that audio is still an under-studied and attractive way of communication that might be useful in multi-robot systems.

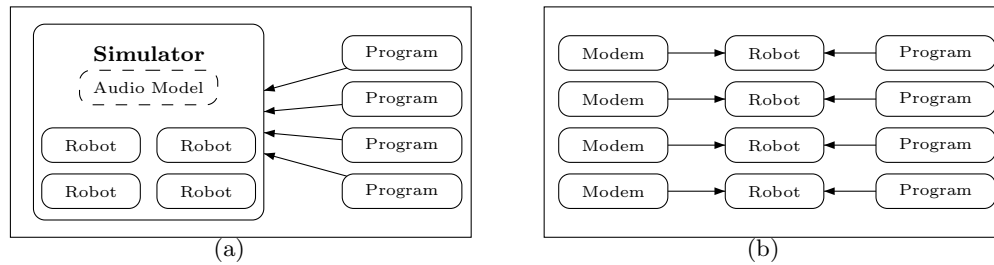


Figure 1.1: System architecture of a multi-robot system running (a) in simulation with robot controller programs using the simulator to model the virtual robots and audio communication between them (b) in real-world with robots equipped with a physical audio message transmission modem.

## 1.6 Contributions

A summary of our contributions during this work include:

- Implementing a simple and practical audio signal model for simulation of audio-based communication among multi-robots.
- Suggesting that audio communication among robots increase the team performance even if using simple audio sensors. This hypothesis is tested in simulation using the mentioned audio model and a prototypical task.
- Developing a network module implementing a modified Carrier Sense-Multiple Access (CSMA) protocol for audio communication among robots in the real-world.
- Proposing a novel distributed algorithm for achieving mutual exclusion locally using audio signaling. The method is demonstrated to work in simulation and real-world experiments.

Our system is designed so that the implemented real-world network module can be simulated with the audio model. This means that the same robot controller program can be used in both simulation and real-world. Figure 1.1 illustrates a general schematic of our multi-robot system architecture.

## 1.7 Thesis Outline

As we will discuss later, this work is just a starting point in the study of audio communication for multi-robot systems. There are multiple paths that could have been taken to follow this research and there are many parameters and implementation aspects that could have been changed for each experiment. Though, throughout this work, we tried to study the most important aspects of this area and generate reusable modules and code that can later be used for more other similar studies.

This is the outline of this document:

### **Chapter 2: Modeling Audio Signals**

First we start by developing a simple model of audio propagation that can help us to advance the research.

### **Chapter 3: Sounds Good: Evaluation of Audio Communication**

Having a working simulator for the physical world and propagation of sound in that world, we take on a generic prototypical resource transportation task and show how audio can be used to solve this class of applications.

### **Chapter 4: Going to the Real World**

In this chapter, we discuss the large possibilities of using audio in real-world experiments. We discuss how implementing a completely new problem while using the old tools from the previous simulation experiments, gives us a new insight.

### **Chapter 5: Nava: Audio Communication Layer**

“Nava” is an implementation of CSMA network communication layer over audio waves. It provides broadcast based communication using small data packets and carrier information to robots while trying to avoid message collisions.

### **Chapter 6: Mutual Exclusion for Robots**

Mutual exclusion is selected for being a very interesting and useful problem in distributed systems and a spatial version of it is implemented using audio communication.

### **Chapter 7: Local Mutual Exclusion Demonstration**

An experiment with self-charging robots and chargers spread around the environment demonstrates how local mutual exclusion is used in a distributed resource management application.

### **Chapter 8: Conclusions and Future Work**

Finally this concludes this thesis.

## Chapter 2

# Modeling Audio Signals

To study the use of audio and as a preparation for building a multi-robot system, our first step is to simulate the physical world and the behavior of the controller code.

Robot simulators can already model the physical world to a useful extent. They can simulate robot movements in response to controller commands and the interaction of objects and obstacles in the environment with robotic sensors such as infrared and laser range finders and cameras. But to the best of our knowledge, none of the available robot simulators can model audio in the way we need it. Having the ability to model audio propagation is our initial attempt in the study of audio communication.

### 2.1 Audio Propagation

Audio waves in an office-like environment have a very complex propagating pattern. The reason is that the sound can be partially reflected when it hits obstacles on its way. It can also be partially absorbed by different materials if they are thick and solid enough and it can also transmit through thin matter. The amount of the reflection and the absorption of the original signal are largely dependent on the hit material properties, the audio frequency and the amplitude of the wave [13]. In Figure 2.1 you can see how an audio signal emitted from a single source can take multiple paths to get to another point.

This complex behavior of audio waves makes it very difficult to build a physically accurate model of the audio propagation. Depending on the implementation such a model may also be very computationally expensive and time consuming.

One goal for having a simulator beforehand is to speed-up the design process compared

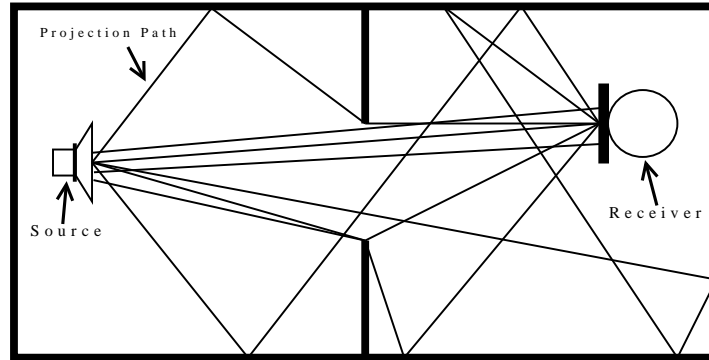


Figure 2.1: Audio propagation in an environment can be complex and at the same time exhibit regular properties that can be used by robotic systems.

to using trial-and-error experiments in the real-world. This makes the move to the physical world much easier. Even a not so accurate model that is fast and realistic enough can satisfy our simulation requirements.

## 2.2 Shortest Path Model

We take a pragmatic approach to simulate audio propagation from an audio source to multiple destinations. This simple model is similar to those used in computer games [6]. In a computer game the simulation needs to be realistic enough to the ears of the player, while at the same time it should be computationally feasible to run in real-time. In our case even being able to run it faster than real-time is a positive point because then the simulator can be sped up to run faster and this will make the results available much sooner.

Our simulator models the audio propagation by the simplifying assumption that the sound traverses the shortest path from the speaker to the microphone and that the received signal intensity is only a function of the length of the traversed path. This means that the reflections or multiple paths are not modeled and that the audio is assumed not to transmit through the solid walls.

The shortest path between the sound source and the destination shows the path to the first direct sound. The direct sound path is the route that the most powerful transmission will take to get to the destination. All other signals that are reflected from the walls and then reach the target will take a longer path. A longer transmission path and the fact that in each reflection, part of the signal gets absorbed by the hit material; means that most

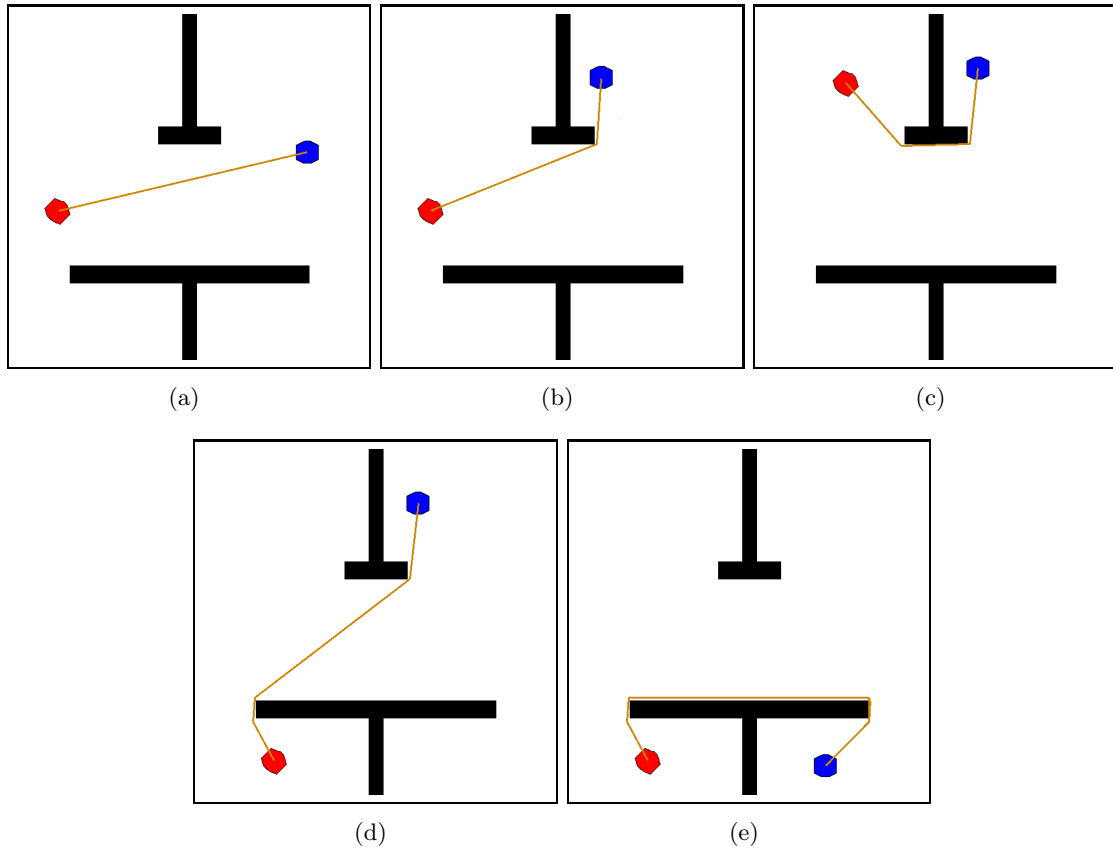


Figure 2.2: The simple model simulates audio propagation as the shortest path between the sound source and the destination. This is the path the direct and the most powerful sound will take.

of the time all the echoed sounds will be less powerful than the direct-shortest path one. Different scenarios are shown schematically in Figure 2.2.

### 2.2.1 Pros

Although a simple model, this approach still exhibits some of the useful features of the real audio transmission. This includes locality, directionality and the gradient of the audio. Locality and gradient are modeled by measuring the distance the wave traveled over the shortest path. The sound direction at the destination is calculated from the vector formed by the last piece in the shortest-path from the source to destination point.

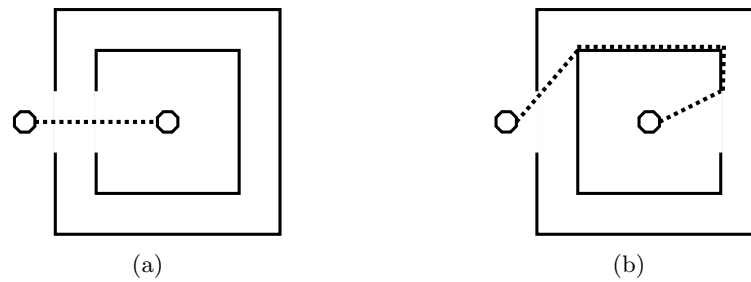


Figure 2.3: The shortest-path audio model models the large difference in the received signal intensity in the scenarios depicted in (a) and (b). In this figure, solid lines are walls, circles are the sender and receiver robots locations and dotted lines are shortest audio paths.

See Figure 2.3 for a sample configuration in which this model simulates the difference. Also in Section 2.5 we will discuss how the diffraction property of sound waves matches this model.

### 2.2.2 Cons

When using this simple model to test a robot controller, there are some drawbacks that should be noted. In the real world, audio reflects, echoes and may take multiple paths to get to the target. Also in the real world sometimes the sound passes through thin obstacles. We do not model other wave properties such the phase either. A controller that is only designed to handle the direct sound may not work as expected in the real-world. These situations should be handled by a smart controller design.

A solution to the above problem can be the careful use of the audio information in the controller. Also the fusion of the audio information to the data from other robot sensors can lead to a better sense of the real world. This information about the surrounding environment can single out the direct sound from its reflections. This will be addressed with more detail in Section 2.4 where the minimal simulation approach is discussed and in Chapter 3 where a sample controller that uses audio is developed.

## 2.3 Implementation

Modeling only by calculating the shortest path is much faster than modeling complex audio propagation. But still running it faster than real-time and over large maps and with large number of audio source and destinations, requires careful implementation.

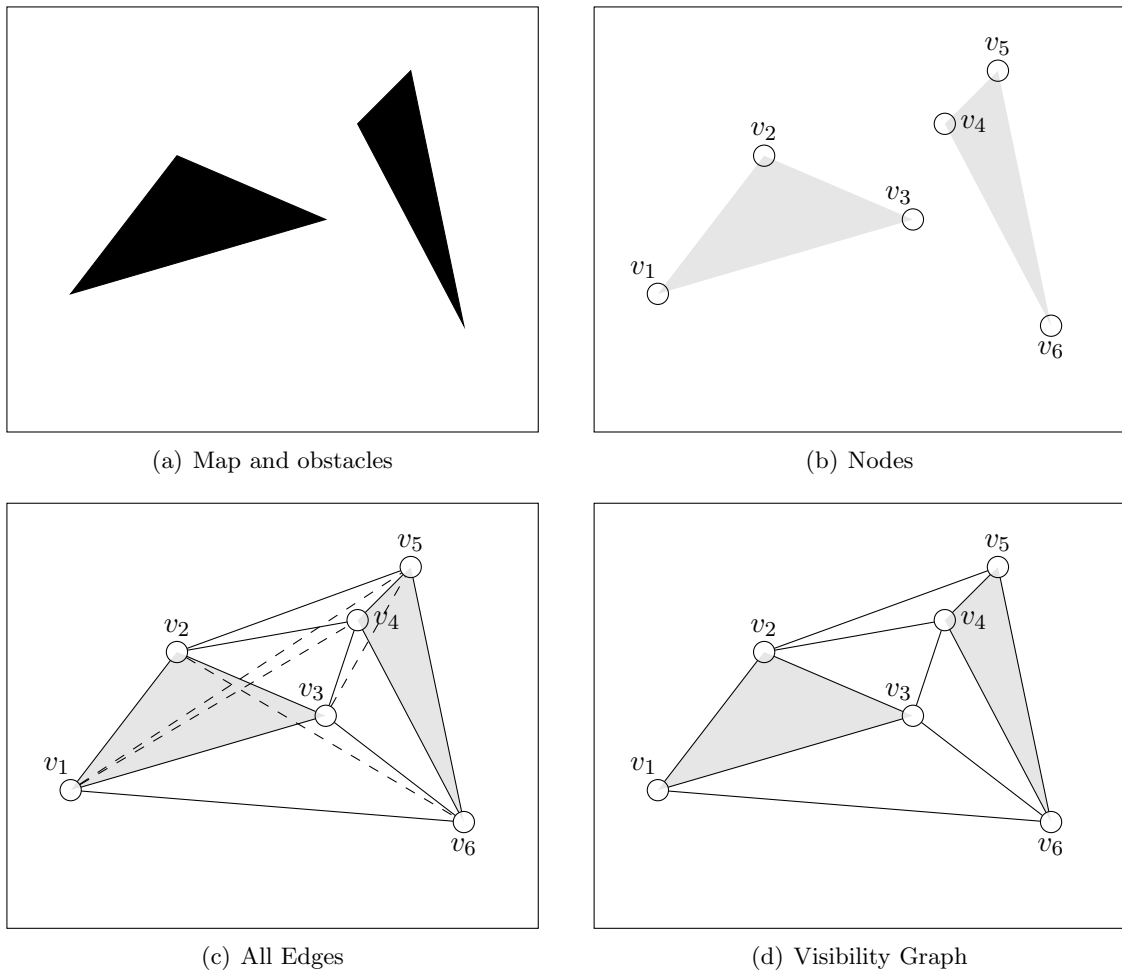


Figure 2.4: These are the steps to build a visibility graph (a) of a map and its set of polygonal obstacles. (b) The obstacle vertices form the graph nodes. (c) & (d) An edge is added for every pair vertices that are mutually visible without hitting an obstacle.

To find the shortest path between two points on a map that does not hit obstacles, we used a computational geometry method. This method is based on a search for shortest paths on the *visibility graph* [10]. The visibility graph of a map  $M$  is defined as:

For a map  $M$  in which obstacles are defined as a set of polygonal obstacles  $S$ , the nodes of visibility graph are the vertices of  $S$ , and there is an edge, called a visibility edge, between vertices  $v$  and  $w$  if these vertices are mutually visible.

To find the shortest-path between the two points  $p_{source}$  and  $p_{destination}$  on a map  $M$ , first the visibility graph  $G$  of this map is built. Then the points  $p_{source}$  and  $p_{destination}$  are added as the new nodes to the graph  $G$  and then the visibility edges between these new nodes and the old nodes are added respectively. Figure 2.4 shows the steps to build a visibility graph of an obstacle map.

Again, this means that there will be an edge between two nodes if these two nodes are mutually visible to each other. The value of all the edges in the graph is set to the Euclidean distance between the corresponding points of the two vertices on the map  $M$ .

After adding the points to the visibility graph, by running the Dijkstra algorithm [11] starting from the audio source node  $p_{source}$ , the shortest distance and the shortest path to audio destination node  $p_{destination}$  are found. If there is more than one destination point, a single run of Dijkstra algorithm will calculate all the shortest distances and shortest paths too.

In our simulator, instead of a polygon map, our input is a 2D floor-plan map of the environment in a bitmap format. But for generating the visibility graph, obstacles should be defined as a set of polygons. To do that, we find the set of potential polygon corner points from the map. These corner points are the set of points on the obstacle boundaries where the audio can diffract and the shortest path can potentially change its direction. We call these points: *diffraction points*. As seen in Figure 2.2 these points are mostly on the obstacle corners and convex parts of the obstacles.

To find these points from a bitmap, a  $3 \times 3$  mask is used. The bitmap is scanned by this mask and is matched against a predefined set of masks. This set has a list of bitmaps that cannot be diffraction points. For example the center point in the mask of Figure 2.5(a) cannot be a point where audio changes direction; the shortest path should either not pass from that point or it should continue its way in parallel to the wall on top.

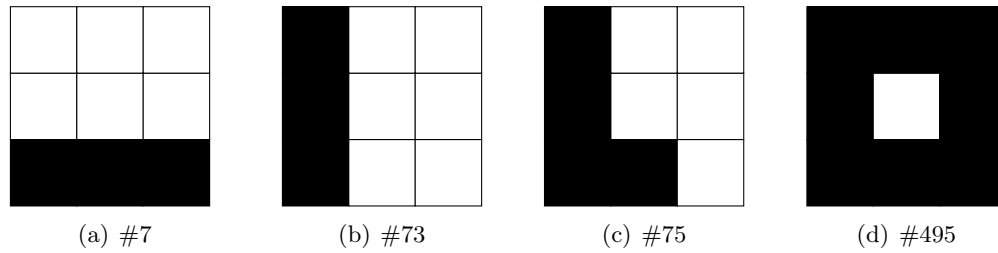


Figure 2.5: A sample set of masks in which their center point cannot be a diffraction point and their corresponding integer representation

1. Find the diffraction points in the map
2. Calculate the visibility graph of all diffraction points
3. For each source and target
  - (a) Add source and target node to the graph
  - (b) Add the corresponding edges between these two new nodes and all the diffraction points
  - (c) Calculate the shortest distance using the Dijkstra algorithm

Program 2.1: Pseudo-code implementation of the shortest-path audio model.

If the  $3 \times 3$  neighborhood around a point is not found in the set, that point will be marked as a potential diffraction point. Each one of these  $3 \times 3$  masks can be simply indexed with a 9-bit integer. See Figure 2.5 for a sample set of these masks.

A pseudo-code of how the shortest path audio model is implemented is shown in Program 2.1. A visual representation of these steps can be seen in Figure 2.6.

### 2.3.1 Complexity Analysis

The running time of the audio model algorithm largely depends on the size and the type of the obstacle map and the number of the robots in the world. Figure 2.7 shows the diffraction points and the shortest path found on two different bitmaps and on different source and target positions.

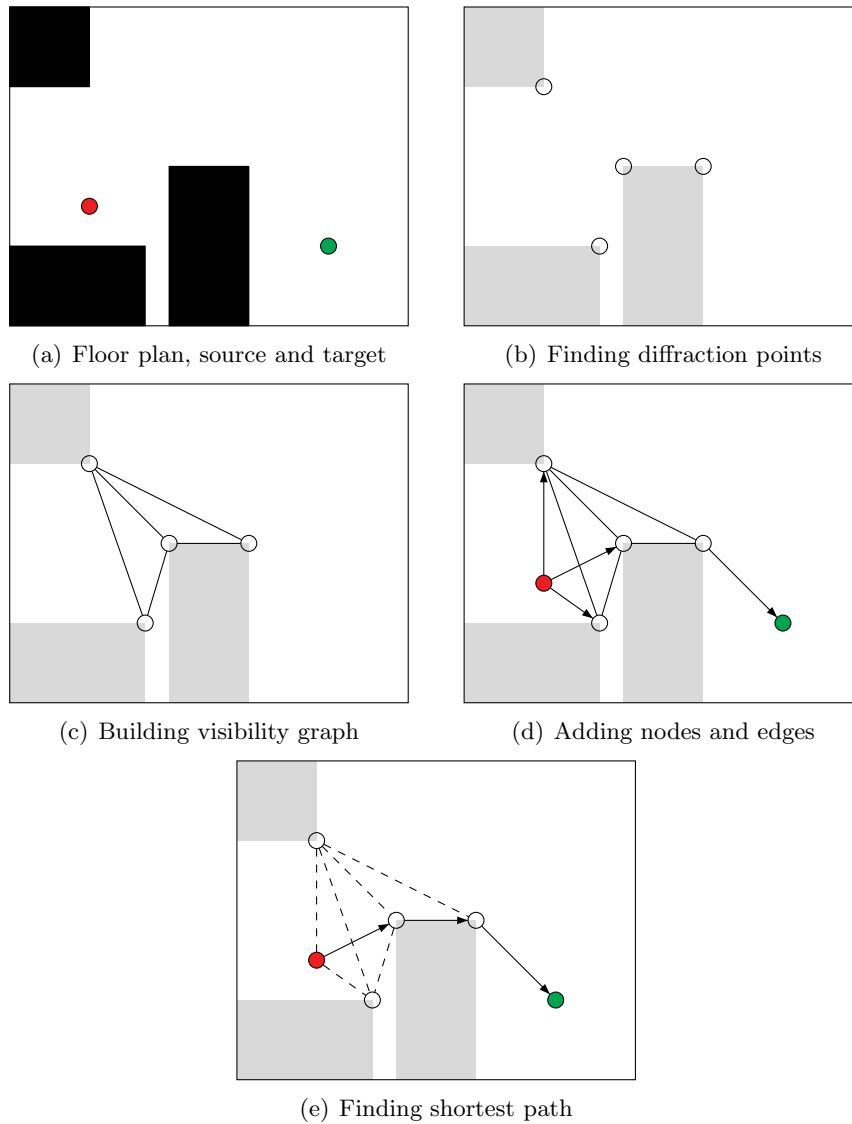


Figure 2.6: The steps to find the shortest path between two nodes in a bitmap using diffraction points and visibility graph

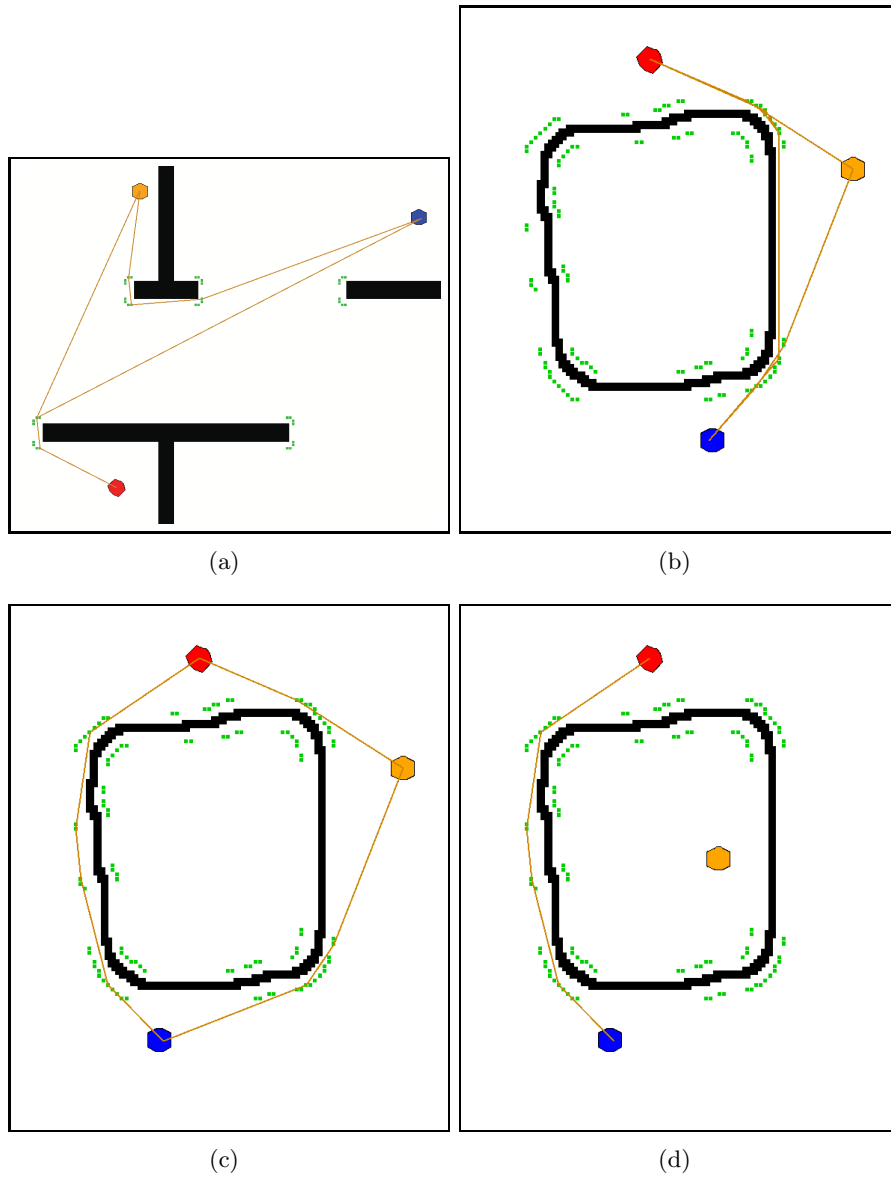


Figure 2.7: An example of a bitmap and a set of diffraction points (small green dots near the walls). These three robots (hexagons) are sending audio signals and the lines show the calculated shortest audio paths. Note the difference in the number of diffraction points of different map types in (a) and (b).

To find the diffraction points in a map, the bitmap is scanned against a set of masks. The time for this step is linear to the number of pixels in the bitmap. The number of pixels in the bitmap is derived from the map size and the map resolution. Assuming that the map has  $l \times m$  pixels, the runtime of this step will be of  $O(lm)$ .

The number of diffraction points found in a map is dependent on the map type. In an office-like environment with rectangular walls, the number of diffraction points is usually very small. They are only placed on the corners and not along the straight walls. In contrast, lots of points on the convex side of a curved surface can be diffraction points. In its worst case, on the perimeter of a circular obstacle, there are infinite points where audio can change its direction. Since we use a bitmap presentation of the map, this number is limited but still can slow down the process if the map has a high resolution. Compare the number of diffraction points in Figure 2.7(a) and Figure 2.7(b).

Building a visibility graph from the diffraction points needs a loop over every pair of points and then calculating whether this point pair are visible to each other or not. Checking for visibility between two points on a bitmap might need a  $O(l)$  calculation where  $l$  is the map size but our simulator provides us with a quad-tree implementation for quicker lookup. If  $d$  is the number of diffraction points found, the time order of this step will be smaller than  $O(d^2l)$ . Refer to Section 2.4 about the simulator we use.

Finding diffraction points and building the initial visibility graph is only done once in the initialization of the simulator. As long as the number of the diffraction points found is not very large, the initialization time will be reasonable. For example in an experiment with near 3000 points, the visibility graph was built in around 90 seconds. In most experiments where the number of the points is under 1000, the initialization is done in less than 10 seconds. To increase the speed, number of diffraction points can be decreased by using more office-like maps or by lowering the map resolution.

After the initialization to calculate the sound paths, in each simulation step all the transmitting and receiving nodes are added to the visibility graph. Then the corresponding edges are added to the graph and the Dijkstra algorithm is run for each sound source. If  $n$  nodes are added to the graph, finding the visibility of nodes to the existing nodes in the graph and adding the edges is of  $O(ndl)$  (each node should be compared against each diffraction point for visibility). We use a binary heap in our Dijkstra algorithm implementation but we have to run Dijkstra one time for each sound source.

If  $m$  of the  $n$  added nodes are sound sources, the order of finding shortest-path is

$O(m(e + d + n)\log(d + n))$  where  $e$  is the number of edges in the final visibility graph. Normally the number of nodes ( $n$ ) is smaller than number of diffraction points ( $d$ ) and that is much smaller than number of edges ( $e$ ). In the worst case, number of edges in graph ( $e$ ) can be as large as  $(d + n)^2$  (As if all nodes are visible to each other).

We can say that the largest part of the order of this algorithm usually is  $O(md^2\log(d))$ . This again shows the importance of making the number of diffraction points small. Also to decrease the number of edges in the graph, we set a maximum hearing range, thus the visibility only needs to be checked between vertices that are in the hearing range of each other.

If in an experiment the number of sound sources ( $m$ ) was a comparable number to the number of diffraction points ( $d$ ) it might be better to use Floyd-Warshall algorithm [12] instead of running Dijkstra algorithm  $m$  times. Floyd-Warshall calculates the shortest-distance in  $O(V^3)$  where  $V$  is the number of vertices in the graph. But in most of the cases that this simulator is going to be used for,  $m \ll d \Rightarrow O(md^2\log(d)) < O(d^3)$ .

To speed-up the simulation and to avoid duplicate calculations a cache structure is also implemented. The cache stores the recently calculated paths for point pairs. Most cache hits happen when there are static nodes in the world.

## 2.4 Stage model

The Player project<sup>1</sup> provides free software tools for robot and sensor applications [14]. The Player robot server is probably the most widely used robotic control interface in the world. It provides an abstraction layer between the robot controller code and the drivers talking to the robotic hardware [52]. This abstraction lets the real devices to be replaced with simulated ones.

“Player is a device server that provides a powerful, flexible interface to a variety of sensors and actuators (e.g., robots). Because Player uses a TCP socket-based client/server model, robot control programs can be written in any programming language and can execute on any computer with network connectivity to the robot. In addition, Player supports multiple concurrent client

---

<sup>1</sup><http://playerstage.sourceforge.net/>

connections to devices, creating new possibilities for distributed and collaborative sensing and control.”<sup>2</sup>

Stage is the most famous simulation engine for Player. It is a two-dimensional simulator that can simulate the interaction of multiple robots with the environment and with each other at the same time. There are various sensor and actuator models included with Stage. These include range finders like lasers and sonar, cameras and grippers. Stage can also model rechargeable energy storages like batteries on a robot.

“Stage is a scalable multiple robot simulator; it simulates a population of mobile robots moving in and sensing a two-dimensional bitmapped environment, controlled through Player. Stage provides virtual Player robots which interact with simulated rather than physical devices. Various sensor models are provided, including sonar, scanning laser rangefinder, pan-tilt-zoom camera with color blob detection and odometry.”<sup>2</sup>

Both Player and Stage are released as open-source software under GNU General Public License<sup>3</sup>, thus making it easy and free for the researchers to use, distribute and modify them. Because of the availability of the Player and Stage and their ease of use, we decided to develop our controller code based on this platform. Stage was providing us with all the needed simulation functionalities that we needed with the exception of the audio propagation model.

In the usual configuration, each robot controller program connects over a TCP network connection to the Player server and then subscribes to the different sensors or sends commands to the actuators. These sensors and actuators can be real robot hardware used through the abstraction provided by Player or they can be simulated devices provided by Stage. By adding the Player provided abstraction to the system architecture of Figure 1.1(a) and thus achieving Figure 2.8(b) we can use the same controller program in both simulation and real-world.

In the first attempt, we implemented the audio model as a controller client connecting to the Player server. The controller code was based on the Playernav utility written by Brian Gerkey, included with Player distribution. The client was subscribing to the position

---

<sup>2</sup><http://playerstage.sourceforge.net/index.php?src=faq>

<sup>3</sup><http://www.gnu.org/copyleft/gpl.html>

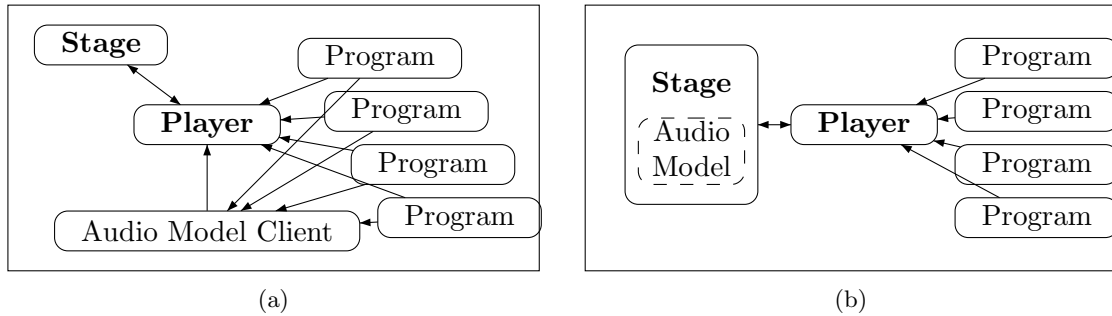


Figure 2.8: The audio propagation model calculates the shortest-path distance between the virtual robots in the simulated world and dispatch messages between robot controller programs that are connected as clients to Player. The audio model can be implemented in two ways: (a) The initial version connected a client to Player. (b) The new model integrated into Stage.

devices of all the robots and getting their position information. It had a TCP/IP server to which each robot could connect and send and receive audio messages from other robots. See Figure 2.8(a).

We later added the audio propagation model to Stage itself. Figure 2.8(b) shows the new architecture. Stage uses some internal structures for fast calculation of ray tracing. It uses a quad-tree to store obstacles as rectangles. This way of storing the obstacles lets the Stage to calculate the direct visibility between two points much quicker. Instead of going over all bitmap pixels between these two points, Stage just jumps over empty areas that have no obstacles. See Figure 2.9. The ability to access these internal structures for the fast construction of the visibility graph was one of the main reasons to include the audio model in Stage. Integrating our propagation model with Stage also makes it easier for other researchers to use it along with Stage in other experiments. Figure 2.10 shows Stage modeling the audio message paths between four robots.

Stage takes a minimal approach for simulation of real-world physics. It provides simple and computationally cheap model of devices with *good enough fidelity*. It does not try to gain great fidelity or emulation of noise in the real-world which can be hard to achieve and computationally expensive. This model encourages the robust control techniques proposed by Jakobi [21]. The simple models in Stage provide a one-way validation environment for robot controllers.

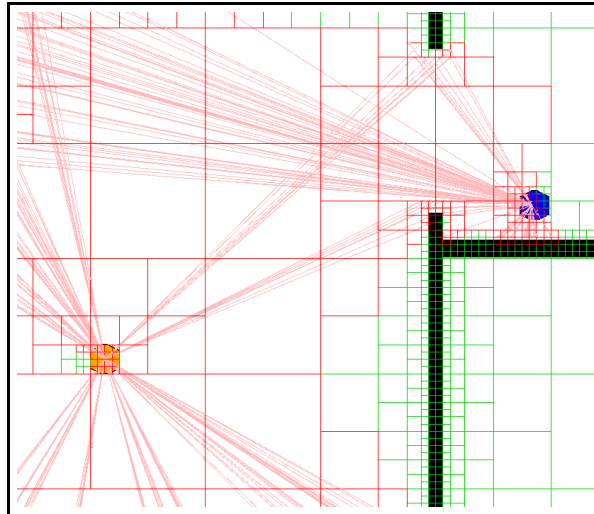


Figure 2.9: Ray tracing in Stage using a quad-tree matrix for speed-up.

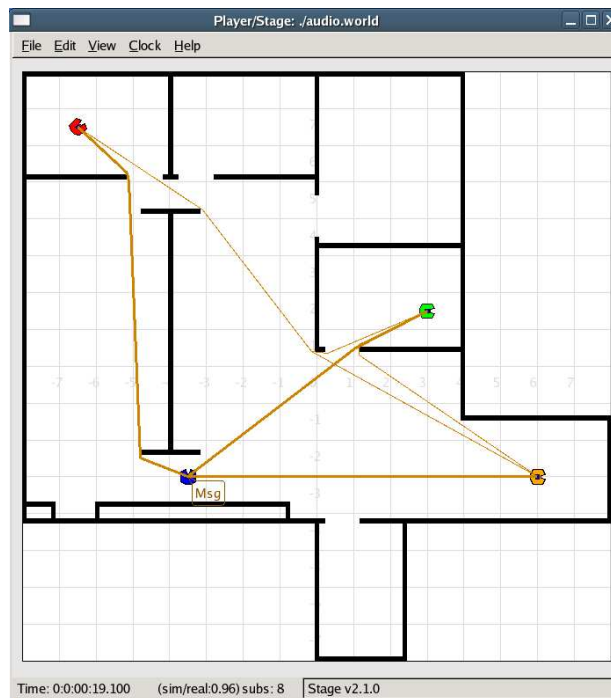


Figure 2.10: Player/Stage modeling the audio message paths between four robots.

If a robot controller does not work in Stage simulation, it is likely to have problems in the real-world too. The reverse argument is not guaranteed to be true: a working controller in simulation does not necessarily work in the real-world. But in practice if the controllers are designed intelligently and tested in Stage, the migration to the real-world in most cases would be easy and with small amount of changes in the code. This good-enough accuracy and cheap computational price with mostly linearly scalable models in Stage makes it an easy to scale simulator for large multi-robot experiments.

Our model of audio propagation follows the same design principles of the Stage simulator. It is a simple model which does not provide accurate simulation but tries to be fast and provide validation facility for testing controller code. The audio model implementation does not scale linearly but it is a polynomial time algorithm which is much faster than modeling a high-fidelity simulation.

The code for the latest development release of the Stage that includes the audio model can be downloaded from the Player/Stage project on SourceForge.net<sup>4</sup>.

## 2.5 Complex Model

Audio waves in reality have a very complex behavior. As discussed before, in our audio model we decided to use a simple model because of its speed and ease of implementation. But researchers have also attempted to model the propagation of sound waves and all its interactions with the environment.

A physically based sound propagation model can handle all these interactions of sound:

- **Intensity** drop while traversing: follows the inverse square law.
- **Absorption** by the hit object: dependent on the material type and audio frequency.
- **Reflection**: follows the law of reflection and dependent on the material type and audio frequency.
- **Interference** of waves: follows superposition law.
- **Refraction**: when there is a change in the medium properties.
- **Diffraction**: bending and spreading out around the obstacle edges.

---

<sup>4</sup>[http://sourceforge.net/cvs/?group\\_id=42445](http://sourceforge.net/cvs/?group_id=42445)

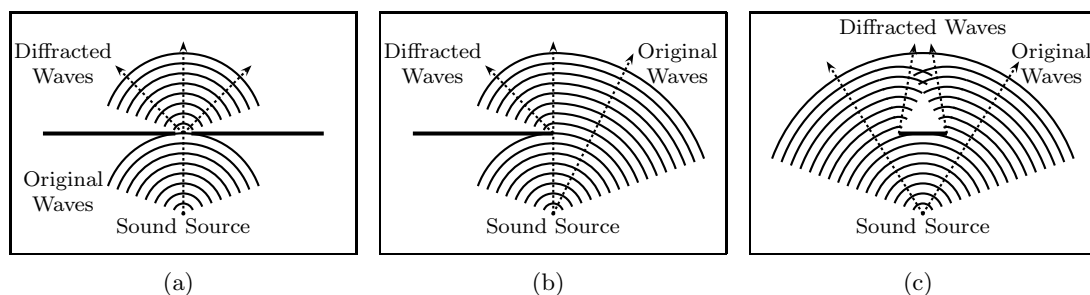


Figure 2.11: Diffraction of audio waves around obstacle edges matches the shortest-path simplification of audio model. (a) Waves passing through a narrow slit. (b) Diffraction behind a wall. (c) Diffraction around an obstacle.

Accurately modeling reverberant sounds allows the prediction of the acoustic properties of the environment [8]. Sample usage for such a model include sound synthesis, modeling of auditoriums, sound generation in computer games, learning the cues for localizing the sound sources as well as its use in robotics applications.

Such a complex model can also be used to validate the already developed simple model against the physical world and to tune the parameters of this simple model to be as near as possible to the real world. Different scenarios of the wave diffraction shown in Figure 2.11 show how the simplified calculation of the shortest-path between the sound source and the target is not that far from the reality of wave behavior.

There are different methods to develop a physically based sound propagation model [13]. Karimian (the author of this thesis) under supervision of Dr. Torsten Möller has developed such a model. “Using Computer Graphics Techniques to Model Acoustics”<sup>5</sup> was done as a course project for Fall 2005 Simon Fraser University’s CMPT770 Advanced Computer Graphics course.

This implementation is based on a 3D graphics rendering technique named Photon Mapping [23]. In photon mapping instead of calculating all the possible reverberation paths, a random sampling of the problem space using photons, as small light packets, is used to estimate the real solution. In similar approaches, Phonon Tracing [3] or Sonel Mapping [24], developed for sound modeling as the first step, the space of the problem is sampled randomly with a large number of small sound packets (a.k.a. sonels or phonons) shot from the source.

<sup>5</sup><http://www.sfu.ca/~pkarimia/courses/cmpt770graphics/proj/>

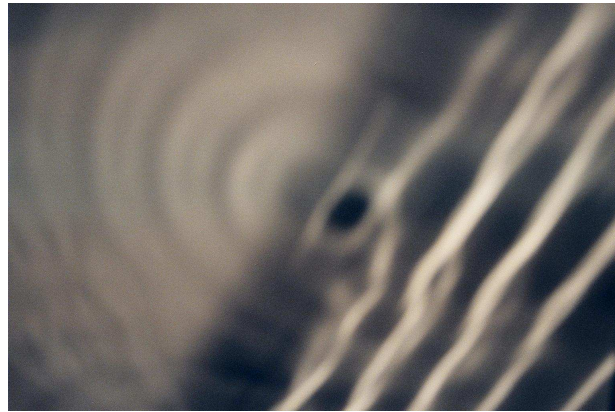


Figure 2.12: Two wooden blocks placed in a circular ripple tank with a slit between them, creating circular waves. Beneath the ripple tank was a sheet of white paper, where the wave patterns appeared due to a light source above the ripple tank. ©Armed Blowfish. Used by permission under the BSD license.<sup>7</sup>

These small packets will propagate through the space according to the wave propagation properties. The probability of a packet reflecting or getting absorbed is calculated from the hit material’s preset attributes. In the second step, each propagated packet now residing somewhere in the world will be treated as a small source of sound itself and an estimation of the real-world propagation will be calculated.

Most graphics rendering methods assume that the light will only propagate along a straight line unless it hits an object. But diffraction of the sound waves makes them different from light. In the photo shown in Figure 2.12 a *ripple tank* is used to model how waves diffract when going through a narrow slit. To model diffraction property of waves researchers have tried different models such as the *Uniform Theory of Diffraction* [49] or *Huygens Fresnel principle* [25].

The “Huygens Fresnel principle” of waves analyzes wave propagation by modeling it as a sum of small secondary waves (a.k.a. wavelets) that are along the advancing wave front. Each of these points along the wave will itself be regarded as a new source of wave [17]. This principle can model how sound waves diffract. See Figure 2.13.

---

<sup>7</sup>Copyright ©Armed Blowfish, all rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, and this list of conditions; Redistributions in binary form must reproduce the above copyright notice, and this list of conditions in the documentation and/or other materials provided with the distribution; Neither the name of Armed Blowfish nor the names of other contributors may be used to endorse or promote products derived from this software without specific prior written permission.

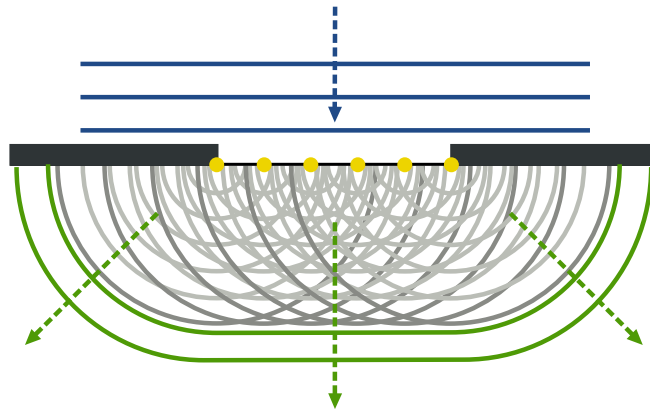


Figure 2.13: Huygens-Fresnel principle analyzes how waves are diffracted using a sum of small secondary waves along the advancing wave front. Compare this Huygens-Fresnel analyzed model with a real photo of wave diffraction shown in Figure 2.12. ©Arne Nordmann. Used by permission.

The Huygens Fresnel principle is suitable for use in a Photon mapping method. Each sound packet in the rendering algorithm will be treated as a Huygens wavelet and its propagation probability is sampled according to the Huygens Fresnel principle. Figure 2.14 shows a sample output of that project modeling an audio source near two vertical walls. There you can see the effect of the reflection and the diffraction of the sound waves in the interaction with objects.

The time complexity of physical modeling of sound propagation is exponential but an approximation algorithm, like the method described shortly above, can bring it down to polynomial time. But still the number of calculations and the complicated rendering algorithm makes it too slow for a real-time simulation. There are attempts to solve this problem by making use of the computational power of the graphics accelerators or physics cards which are the hardware based acceleration expansion cards for personal computers. However, for the near future we are restricted to less realistic, but fast and approximate models, such as the one described in Section 2.2.

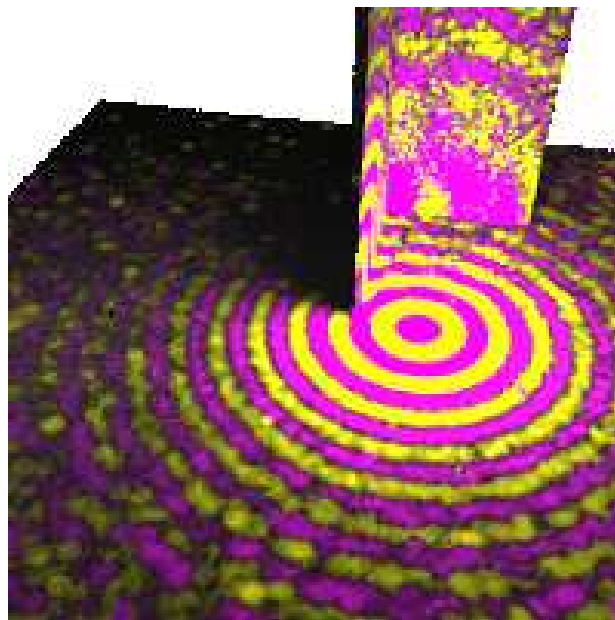


Figure 2.14: A complex audio propagation model, modeling the direct sound, reflection and diffraction of audio waves. Such a model can provide high accuracy but will be computationally expensive.

## Chapter 3

# Sounds Good: Evaluation of Audio Communication

The work described in this chapter is published as “Sounds Good: Simulation and Evaluation of Audio Communication for Multi-Robot Exploration” that was presented at IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS’06) in Beijing, China [26].

Having a fast simulator equipped with an audio propagation model lets us study audio communication in multi-robot tasks. Emulating robotic tasks that use audio messaging allow us to gather more information and get an insight into the usefulness of this type of communication.

To guide the design of a multi-robot system, in the “Sounds Good” experiment, we evaluated two different types/designs of audio direction sensor:

- **Omni-directional** sensor with high accuracy in detecting sound direction.
- **Bi-directional** sensor with a one bit direction resolution.

The questions that motivated the work in this experiment were simple:

- Can audio communication, even utilized in a simple way, enhance the performance of a group of robots?
- Do our robots need accurate sound localization to get significant benefits from audio signaling?

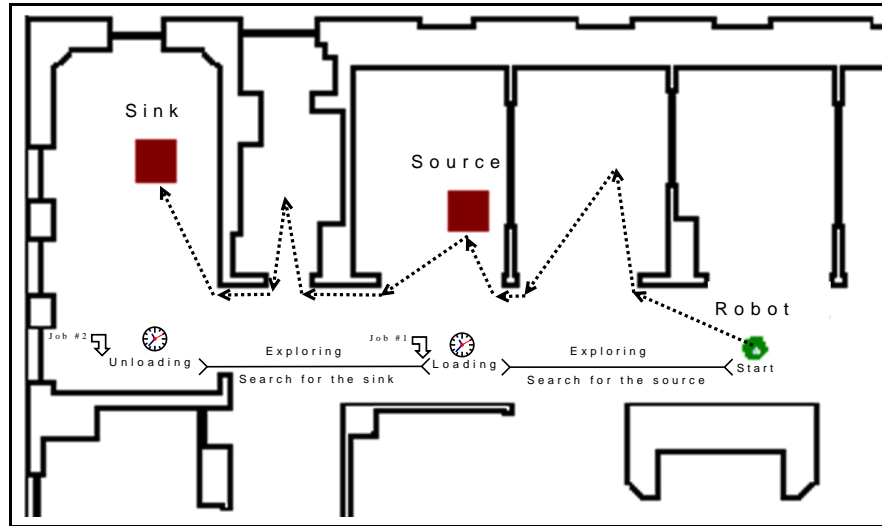


Figure 3.1: A schematic of the examined resource transportation task for a single robot. The robot starts by searching for the source, loading a virtual resource, searching for the sink and then unloading. From the start to when the unloading finishes two completed jobs will be counted.

### 3.1 Task Definition

The first step we took to study the usefulness of audio in practical robot applications was to define a task and then try to show the use of the audio communication in that duty. The task definition should be general and prototypical and it should be functionally similar and applicable to other problems and applications.

As a motivating example, we examine a general *resource transportation task* which at the same time requires robots to explore the world. The exploration is to find and then go to the two initially unknown locations corresponding to a source and a sink of some notional resource. After finishing the loading at the source; the robots then move to the sink for unloading. A schematic of this task is shown in Figure 3.1.

This importance of this task is that it is functionally similar to the various exploration and transportation scenarios that have been previously studied. Vaughan in [53] showed how a team of real-robots cooperate with each other to robustly transport resource between two locations in an unknown environment. In that work, the robots share information with each other through the direct modification of the environment inspired by the trail-laying of ants.

Audio communication can be regarded as a way of modifying the environment. But in comparison to other methods such as trail laying or radio frequency communication, audio messaging is bio-inspired, easier to implement, temporary and harmless to the environment. It can be sensed when the robot is in the proximity of the sound source. But it can only be sensed as long as the source is still emitting sound.

In our defined task, a completed *job* is defined as finding one marker and spending a fixed amount of time there (30 seconds in our case) working (loading/unloading) and then changing the goal to another marker. The *metric of the success* is the time taken for the entire team to complete a fixed number of these jobs: trips between the source and the sink (20 trips in our experiment).

A group measure like the one that is selected will show how the whole team performs rather than evaluating individual robots. Selecting the time to finish a fixed number of jobs as a measure rather than counting the number of the jobs finished in a fixed amount of time provides a better resolution. In the latter case, the time can be over while a job is near finishing and thus this benchmark will not include the partial job.

With such a definition for the success measure, the amount of work done in unit time can be increased by adding additional robots. If the robots act independently performance increases linearly with the number of robots added until interference between robots becomes significant. If the robots are not independent but instead actively cooperate by sharing information, we can expect to improve performance further [54].

In these experiments we examine the effect of robots generating audio signals to announce the proximity of a target on the overall system performance.

## 3.2 System

At Simon Fraser University's Autonomy Lab<sup>1</sup>, we build life-like machines. Our goal is to increase the autonomy of robots and other machines.

Our interest is in designing systems of many small, low-cost robots. In this experiment, we assume our robots to be similar autonomous agents with little computational power and memory. As the robots are individually autonomous, having shared memory or a map of the world will not be trivial for them. The robots' initial positions are random and the

---

<sup>1</sup><http://autonomy.cs.sfu.ca/>

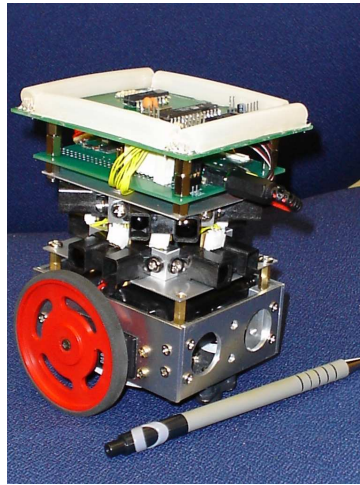


Figure 3.2: Prototype of the Chatterbox, a small robot, running Linux and equipped with different types of sensors and emitters.

world will be large compared to the size of the robots and it is not practical for these small indoor robots to have an accurate global localization. Even relative positioning is hard to achieve because of not having a perfect odometer.

At the Autonomy Lab, we are working on the “Chatterbox” project which is building forty small robots to study long-duration autonomous robot systems. These robots run Linux on Gumstix<sup>2</sup> single-board computers. At the time of this experiment, the design of the swarm was to build small two-wheeled robots, each 15cm in diameter (a little larger than a CD) and the same height and a maximum speed of 30 cm/second. They will work in large, office-like environments which is too large for the robot to store a complete world map. Figure 3.2 shows a prototype of this robot.

As the robots were not ready at the time of this project and our goal was to boost the design step; this experiment is done in simulation. The simulations used only the same sensors and actuators that we planned to make available on the real robots.

The simulated sensors are configured to match the real devices as closely as possible given the limitations of Stage. For avoiding the obstacles and to build a map of the robot surroundings, eight infrared sensors with a range of 1.5 meters, similar to the Sharp GP2D12 ranger device, are used.

---

<sup>2</sup><http://www.gumstix.com/>

Microphones are low-cost audio sensors and we tried to take advantage of the power of these sensors. In addition to the microphones, our robots are equipped with some other low-cost sensors: a single loud-speaker, infrared rangers, and a low-resolution CCD camera with a range of five meters.

The camera and a simple hardware-based image blob finder are used to identify the markers showing the position of the source and the sink locations. This camera is assumed to be similar to the design of the CMUCam [39]. The camera can be substituted with any other fiducial-type sensor with the well-known ability to guide the robot to a nearby line of sight object.

Two configurations of the simulated audio sensor were tested: *omni-directional*, i.e. giving high-resolution information about the direction to a sound source, and *bi-directional*, giving only one bit of direction data. The maximum audio receiving range was set to 15 meters.

To design and evaluate our system we used the Player/Stage [14] robotics package. Robot controllers are written as clients to the Player robot interface server which provided device-independent abstraction layer over robot hardware. The robots' hardware, movements and interactions with obstacles are simulated in Stage and it generates the appropriate sensor data. In our system, all the sensors and robot parameters in Player and Stage are set to model the real-world scenario as closely as the software will allow.

At the time of this experiment, the audio model had not yet been integrated into the standard Stage distribution. The audio model was implemented as client software connected to Player. The audio client obtains map and robot position data from Player, and acts as a communication proxy between the robots. Robots emit *sound* by making a request to the audio client. Our audio model client calculates the shortest distance between the transmitting robot and all receiving robots. The intensity of the received signal is determined by the distance traveled. If any of the robots receive a sound above a minimum threshold, the audio client transfers the sound data including the received intensity and direction to the receiving robot.

But since these experiments were done, as discussed in Section 2.4, the model is now moved into the development branch of Stage. The availability of audio propagation simulation in Stage plus the provided physical audio module in Player (See Chapter 5) makes it much easier to use the same controller code used in simulation in the real-world.

### 3.3 Implementation

The source and sink are two arbitrary, distinct world locations. The source and sink, respectively, provide and consume units of some abstract resource to and from the robots. The robots transport this virtual resource from source to the sink. This models robots transporting widgets around a factory or mail around an office, for example. In order for the robots to be able to find them, these locations are marked with optical fiducials (here after markers), visible in the on-board camera only over short distances with line of sight. Robots must find the source and sink locations and travel between them as quickly as possible. Without global localization, the robots need to explore the world to find the markers that indicate source and sink. On reaching a marker, a robot stops there for a short, fixed amount of time intended to model the robot doing some work at that location, such as grasping an object. After this time is up, the robot seeks the other location marker. This way of implementing the system permits marker locations to change arbitrarily over time.

Given a complete map of the environment and perfect localization of robot and all resource locations (hereafter *targets*) we can apply standard planning techniques to achieve near-optimal performance. In dynamic indoor environments this kind of world knowledge is costly or impossible to obtain. An alternative trivial solution is for the robot to wander randomly until it bumps into a target. This method will give poor expected performance in large environments, but it has the attractive feature that it does not make any assumptions about, or require any knowledge of, the environment. Better-performing single-robot solutions require more sophisticated search strategies that make certain assumptions about the world. For example by remembering the location of previously-seen targets we can find them again quickly, assuming that they do not move: an assumption that may not hold in dynamic environments. Any implemented system must select a search strategy that trades off performance, assumptions and world knowledge. A reasonably-performing, scalable system based on local maps is proposed below.

Without having a global map of the environment and/or any prior information about the location of the source and sink, the robots have to explore the world to find the markers. As a way of communicating with other robots and feeding them with information, while exploring the world each robots will periodically announce by audio the markers it saw in the last 10 seconds.

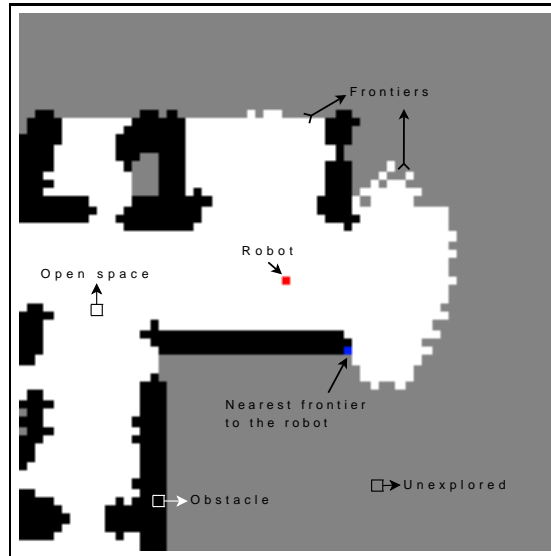


Figure 3.3: Occupancy grid and frontier based exploration.

### 3.3.1 Frontier Based Exploration

There are many different approaches that can be taken for exploration to find markers. One suitable method is *frontier-based searching* proposed by Yamauchi [58]. In frontier-based searching each robot uses an occupancy grid with three states: empty, obstacle, and unknown/unexplored for each cell to store the global map. Initially, the entire world is unexplored, but as the robot moves, the occupancy grid will be filled using the sensor readings. Frontiers in this occupancy grid are defined as those empty cells that have an unknown cell in their 8-connected neighborhood. Each robot moves towards the nearest frontier and gradually it explores all the traversable areas. Selecting the nearest frontier is a greedy strategy to minimize the traveling cost. The exploration is complete when there are no more accessible frontiers. The frontier-based approach guarantees that the whole traversable area of the map will be explored. Figure 3.3 shows an occupancy grid around a robot and the nearest frontier to the robot.

Yamauchi in [59] also proposed the same algorithm for multi-robots by using a shared occupancy grid between all the robots. In the Sounds Good experiment we use an adaptation of the original single robot approach. It is described below.

### 3.3.2 Local Map

Our constraints require that the robot has no *a priori* global map, has no means to globally localize itself, and has conventional odometry with unbounded error growth. But producing a global map during the exploration of a large world has a high computational cost and needs a large memory for storing the information. We wish to avoid the memory and computational cost, yet still perform an effective exploration.

Our approach is to maintain a short-range occupancy grid of robot’s current neighborhood, centered at the robot. This is a fixed size occupancy grid that we call a *local map*. As the robot moves, the local map is updated continuously from sensor data. Because the local map only contains the information about the neighbor cells, some of those cells may “fall off” the edge of the local map as the robot moves, and are lost.

To explore the world we use frontier-based exploration but on a local map instead of a global occupancy grid. We assume the state of all cells outside the map is to be *unknown*. This means that all empty cells on the map border are frontiers and thus can be selected as a potential robot target. This guarantees that, unless the robot is stuck in a closed wall, it will eventually traverse to the map borders and thus moving the local map to cover the unexplored areas outside the local map.

The local map uses constant memory, unlike the global map, which uses memory proportional to the area explored. But unlike the original frontier-based searching, using a local map has the disadvantage that long term cycles in robot position are not detectable. A robot avoids visiting a previous cell as long as that cell is covered in the map and marked as visited. But in a local map information far from the robot will be lost. We later try to avoid this problem by added randomness to the exploration and robot movements.

We modify the original frontier-based method so that each cell value in the map expires after a fixed amount of time and reverts to *unknown*. This is to cope with the dynamic elements of the world such as other robots, which may look like obstacles to the sensors. This would also take care of possible sudden errors in odometry such as wheel slips.

### 3.3.3 Using Audio Information

We aim to discover whether audio signaling can be used to improve performance of a robot system searching for the markers. To be feasible for real-world implementation in the short term, we allow only very simple audio messages, representing single values from a pre-set

range. This will be something similar to robots using *Dual-Tone Multi-Frequency (DTMF)* code, as used by a touch-tone telephone, to talk to each other. When a marker is seen and while it remains in view, the robot generates a DTMF tone identifying the marker. By continuing to announce the marker for a short period of time after the marker is no longer in view, it allows other robots to continue to receive location information, thereby increasing their chance of finding the markers. In our experiments this time is set to 10 seconds.

In addition to receiving the marker number, other robots in the audible range will know audio volume and the direction from which the sound arrived. This is feasible in the real world: Valin [51] showed how microphone arrays can be used to detect the angle with high precision. However, a far simpler configuration is to have only two microphones and the direction can be simplified to two states: depending on the microphone placement, this could be front or rear, left or right.

In the original frontier based search, the target frontier point selection is based on its distance to the robot. This is a simple greedy approach that can be replaced by any scoring function. In the original algorithm case, the nearer the frontier is the higher the score it will get. Other sources of information can be added to this scoring function to optimize for other criteria rather than achieving minimum travel [35].

In our problem the goal is to minimize the time to find the marker locations. To do this we add the information we received through audio messages to our selection scoring. To select the goal point on the local map, a cost function which selects a frontier cell is used. Cell selection can be based on multiple weighted factors including distance to that cell, a random weighting (to add stochasticity to help avoiding the loops), or the information extracted from the audio messages.

To use this information from the audio sensor, the messages received are stored in a queue each with an arrival time-stamp. Each cell can now be scored based on the received messages and this score is subtracted from the distance cost for scoring frontiers. These are the factors for scoring each cell based on one message:

- Difference of cell direction compared to message direction (-1.0 to 1.0)
- Message age (0.0 to 1.0)
- Message intensity (0.0 and 1.0)

In our implementation, for a cell  $c = (c_x, c_y)$ , and the set  $M$  of messages, where each message is  $m = (m_{data}, m_{\theta}, m_{level}, m_{age})$ , the cost function is:

$$f(c, M) = |c| + G - w \prod_{m \in M} (\Theta(m_{dir} - c_{dir})\Omega(m_{age})\Phi(m_{level}))$$

In which,  $|c| = \sqrt{c_x + c_y}$  is the cell distance from robot.  $G$  is a small Gaussian random value with a mean of zero.  $w$  is a fixed weight. For  $0 < x < 2\pi$ ,  $\Theta(x) = \frac{\pi - 2x}{\pi}$ , which is the direction difference factor. This is an approximation for  $\Theta(x) = \cos(x)$ .  $\Omega(m_{age})$  is the message age factor and  $\Phi(m_{level})$  is the intensity factor. Messages are discarded from the queue after three minutes.

This method of adding and subtracting different terms from the scoring function of the frontier-based searching is a searching algorithm that is aware of robot's surrounding environment. It lets us easily compare the effect of using different types of information in the scoring function. It uses all the information for decision making but at the same time it only ranks the unexplored area frontiers and not the obstacles nor the previously explored areas. Although this may not be clearly seen in the simulation experiment where we assume the audio only traverses the shortest path through the open areas. In the real-world, a single sound can be heard from different directions because of reflection or transmitting through the materials. A frontier based exploration will never try to explore through a wall just because it heard a sound coming from that direction. Instead it will be more affected by the direction of the strongest message that corresponds to the shortest path wave.

This method, although being an easy way to add audio information into the search, has some problems. If there are two messages that are from two opposing angles the robot will take the frontier that is in between these two. This is not always the best choice. But still in most cases this simple scoring function will perform reasonably.

Of other possible expansions to this system is a bias in favor of the current direction of the robot to prevent the robot from making cyclic decisions. It is also possible to generate repelling sounds as well as attracting sounds. This means that a robot can signal others not to go near it, causing the robots to spread throughout the map. One sample scenario is when a robot fails to see a marker for a long period of time it can start generating a repelling sound. This time-based approach, depending on the map configuration and the markers' positions, has its own drawbacks. For example a robot can start repelling other robots from its current position while the marker is hidden somewhere near it. A better

but harder-to-implement method is to generate the repelling sound based on the amount of area already explored near each robot. However, none of these methods are implemented in our experiments.

### 3.3.4 Path Planning in a Local Map

For obstacle avoidance and planning we combine the local map with a potential field path-planning method due to Batavia [2]. Using the collected environment information stored in the local map, to avoid the robot hitting the obstacles, the obstacles will be grown by the size of the robot.

From this new map a *traversability map* is built. The traversability map is the result of applying a distance transform to the obstacles in the local map. The distance transform operator numbers each cell with its distance from the nearest obstacle, so a non-empty cell is numbered as zero; all its empty neighbors will be one and so on. In our implementation, a city-block (“Manhattan”) distance metric is used, thus assuming travel is only possible parallel to the X and Y axes.

Occupancy grid cells marked “unknown” are handled as empty cells. An exponential function on the value of a cell in the traversability map shows the cost of moving to that cell. This forces the robot to maintain a suitable distance from obstacles while not totally blocking narrow corridors and doorways.

A wave-front transform is then used to generate a robot-guiding potential field from the local map and traversability map. The field is represented by a bitmap in which the value of each cell indicates the cost of moving from the goal to that point. It is implemented by a flood fill starting from the target cell, valued 1, and numbering all other cells with their minimum travel cost from the target. The cost function is the city-block distance plus the risk of getting near to an obstacle. This risk cost is taken directly from the corresponding cell in the traversability map.

By always moving from a cell into the lowest-valued adjacent cell, the robot takes the optimal path to the target. If implemented using FIFO queues, both steps of this algorithm scale  $O(n)$ , where  $n$  is the number of cells in the map - a fixed value in our implementation.

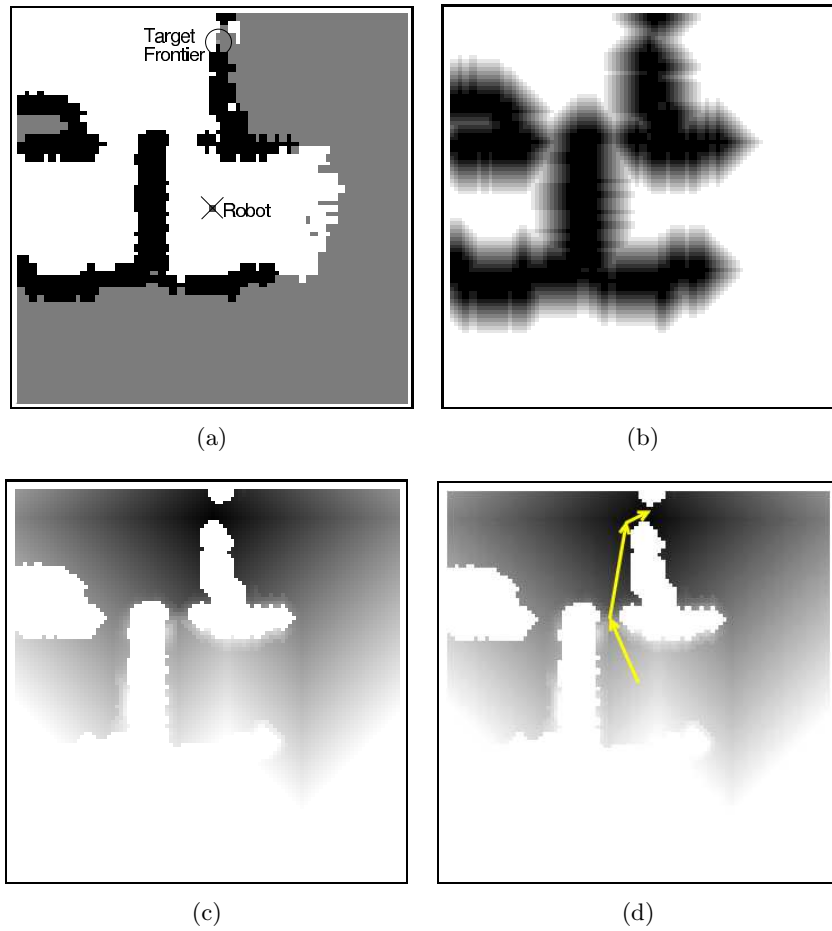


Figure 3.4: (a) *Local map*: an occupancy grid built by a robot over a period of time. In this image, *white* shows empty area, *black* shows known obstacles (enlarged by the robot size), and *gray* shows unknown area. (b) *Traversability map*: the darker the cell color, the harder going to that cell is (c) *Potential field map*: with zero at the target frontier and growing for neighbor cells (d) The path the robot takes to reach the target by following the steepest gradient in the potential field.

Config#	No Audio		Bi-directional		Omni-Directional	
	$\mu$	$\sigma$	$\mu$	$\sigma$	$\mu$	$\sigma$
#01	11631.8	2642.9	7427.4	2480.8	6669.9	1622.5
#02	7046.9	2310.7	1679.4	571.9	1724.3	492.1
#03	6575.8	2656.3	1598.2	1017.6	1782.8	1266.4
#04	2912.1	726.3	2097.9	814.2	1851.7	795.3
#05	3227.6	684.0	1290.3	478.1	1050.3	305.7
#06	4525.9	713.5	636.5	63.8	715.4	636.9
#07	5731.6	1335.1	4144.1	858.1	4459.3	1084.1
#08	4345.9	919.9	601.4	74.7	542.6	42.9
#09	9925.8	3444.0	4360.8	1550.5	3431.7	1063.5
#10	546.3	192.8	295.3	29.4	278.1	34.0

Table 3.1: Mean and standard deviation of time to finish an experiment using different types of audio sensors and on different starting configurations. The numbers are in seconds. Each mean and deviation is calculated using 20 different experiments.

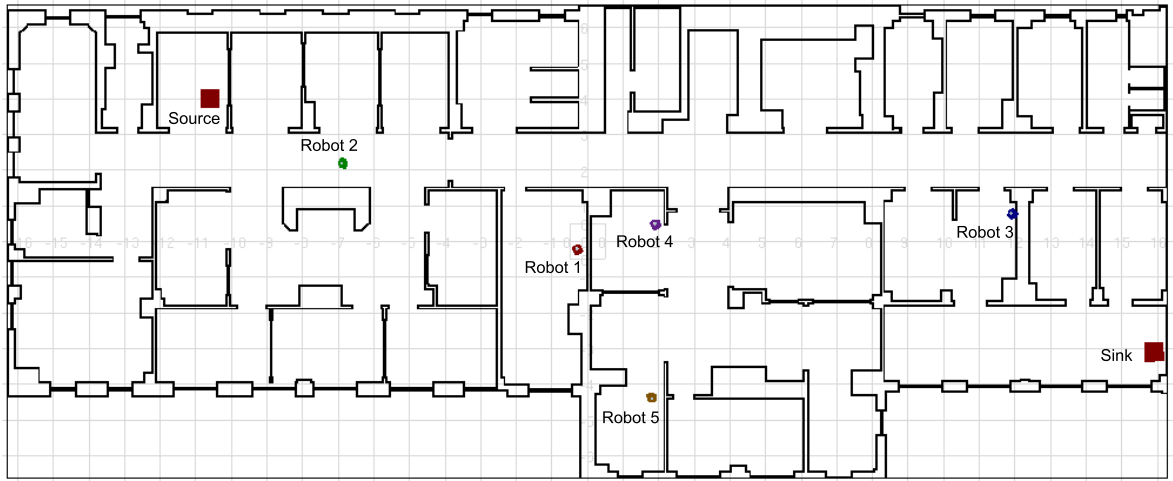
### 3.4 Experiment

The environment map for this experiment is the “hospital section” map distributed with Stage, which is derived from a CAD drawing of a real hospital. It is a general office-like environment with rooms and corridors and a total size of 34 by 14 meters. The map is large compared to the robot’s size and sensor ranges (it is  $227 \times 93$  times a  $15 \times 15$ cm robot size).

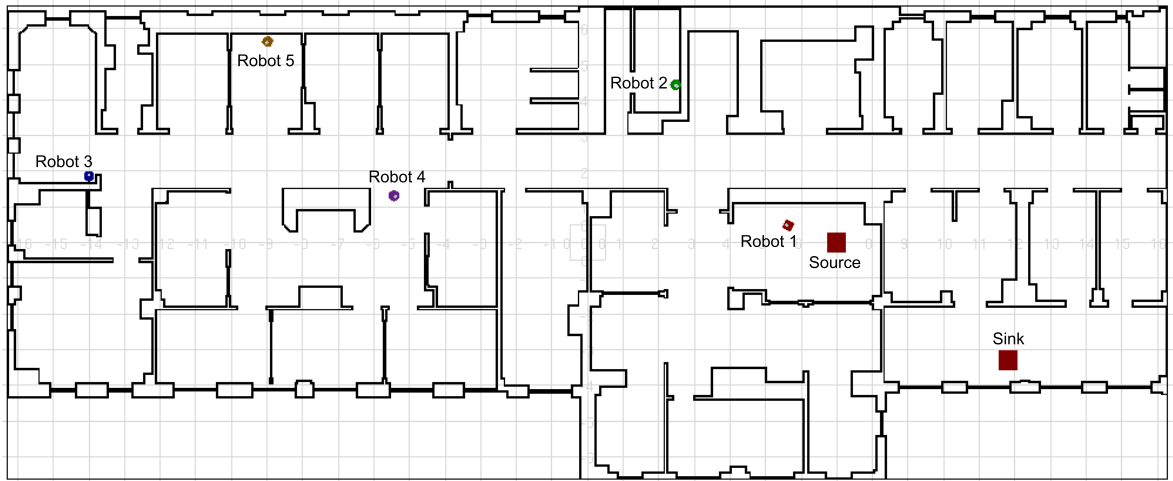
A *starting configuration* is a list of starting (*position, angle*) tuples for robots and the position of the two markers (working areas). A *valid starting configuration* is a starting configuration in which no object is placed over an obstacle and all markers are reachable. Ten different valid starting configurations are randomly generated.

The time for a complete *job* is defined as finding one marker and spending 30 seconds there working (loading/unloading) and then changing the goal to another marker. In each experiment the time for completing a total of 20 jobs by 5 robots is measured. It is possible that different robots will complete different number of jobs. This means that if one robot becomes stuck somewhere, the other robots can still continue to work.

We ran 20 experiments of 3 different methods over 10 different starting configurations, for a total of 600 simulation trials. Table 3.1 summarizes the results for all the starting configurations and for three different audio configurations: 1) no audio sensor, 2) bi-directional microphone and 3) omni-directional sensor. Figure 3.6(a) shows the mean and the 95%



(a)



(b)

Figure 3.5: Randomly generated initial configurations in a partial hospital floor plan (“hospital\_section” in Stage). Each map is  $34 \times 14$  square meters. The robots are the small circles each  $15\text{cm}$  in diameter. The markers, shown as boxes, are resource locations. (a) Initial configuration #01 (b) Initial configuration #09.

confidence interval of time to finish each job in one of the initial configurations (Configuration number #09, shown in Figure 3.5(b)). The chart shown in Figure 3.6(b) shows the time to finish the total 20 jobs for all initial configurations. It can be seen that the most important factor in the total time to finish the job is the initial configuration and especially the position of the two working areas. In configuration #01, shown in Figure 3.5(a), the two markers are far apart and the job-completion times for all methods seen in the chart shown in Figure 3.6(b) are large. However in some configurations like configuration #09 shown in Figure 3.5(b), differences in mean completion times between the three sensing methods are visible.

### 3.5 Results

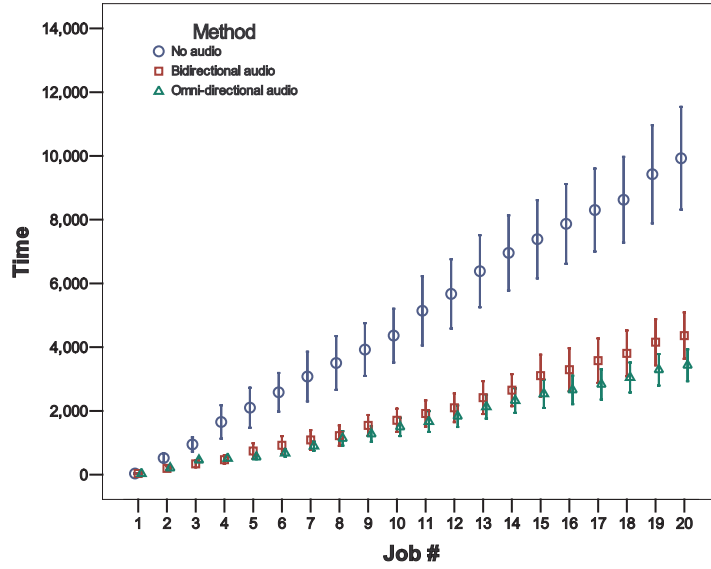
To analyze these differences we ran *t-tests* between every pair of methods on each map to determine which pairs have significantly different means. From the results depicted in Table 3.2, we found that:

- For every map, there is a statistically significant difference between no audio and any method which uses attracting audio.
- Only two of the configurations using omni-directional sensors were statistically different from bi-directional ones
- Of these two, the differences were inconsistent.

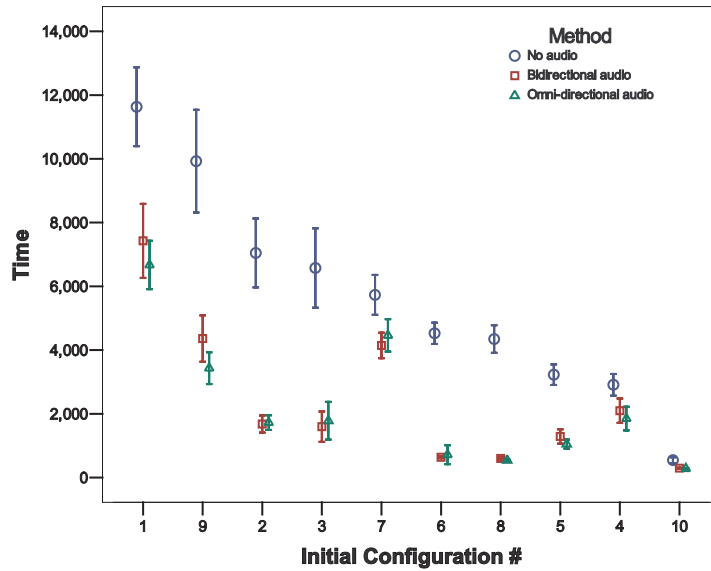
Given these results, we conclude that:

1. Using attracting audio can affect the performance significantly.
2. As there is no statistically significant difference between an omni-directional and a bi-directional microphone and the bi-directional sensor is cheaper and requires less signal processing, the use of a simple bi-directional microphone can be recommended.

It should be noted that the size of the performance gain given by using audio is likely to be sensitive to the implementation parameters and environment properties. It also seems likely that the 1-bit audio direction sensor would not perform so well in less constrained environments.



(a)



(b)

Figure 3.6: The mean and 95% confidence interval time (in seconds) to (a) finish each job in one of the initial configurations (configuration number 9) (b) finish all 20 jobs in all configurations. The configurations are reordered by the time of "no audio" method for the sake of clarity.

Method 1	Method 2	Config #	Perf. gain %	t-Test	Significance %	Different?
No audio	Bi-dir	#01	56.6	5.2	0.001	Yes
No audio	Bi-dir	#02	319.6	10.1	0.000	Yes
No audio	Bi-dir	#03	311.4	7.8	0.000	Yes
No audio	Bi-dir	#04	38.8	3.3	0.190	Yes
No audio	Bi-dir	#05	150.1	10.4	0.000	Yes
No audio	Bi-dir	#06	611.0	24.3	0.000	Yes
No audio	Bi-dir	#07	38.3	4.5	0.007	Yes
No audio	Bi-dir	#08	622.7	18.1	0.000	Yes
No audio	Bi-dir	#09	127.6	6.6	0.000	Yes
No audio	Bi-dir	#10	85.0	5.8	0.000	Yes
No audio	Omni-dir	#01	74.4	7.2	0.000	Yes
No audio	Omni-dir	#02	308.7	10.1	0.000	Yes
No audio	Omni-dir	#03	268.8	7.3	0.000	Yes
No audio	Omni-dir	#04	57.3	4.4	0.008	Yes
No audio	Omni-dir	#05	207.3	13.0	0.000	Yes
No audio	Omni-dir	#06	532.6	17.8	0.000	Yes
No audio	Omni-dir	#07	28.5	3.3	0.206	Yes
No audio	Omni-dir	#08	700.9	18.5	0.000	Yes
No audio	Omni-dir	#09	189.2	8.1	0.000	Yes
No audio	Omni-dir	#10	96.5	6.1	0.000	Yes
Bi-dir	Omni-dir	#01	11.4	1.1	26.026	No
Bi-dir	Omni-dir	#02	-2.6	-0.3	79.157	No
Bi-dir	Omni-dir	#03	-10.4	-0.5	61.417	No
Bi-dir	Omni-dir	#04	13.3	1.0	33.947	No
Bi-dir	Omni-dir	#05	22.8	1.9	6.624	No
Bi-dir	Omni-dir	#06	-11.0	-0.6	58.490	No
Bi-dir	Omni-dir	#07	-7.1	-1.0	31.448	No
Bi-dir	Omni-dir	#08	10.8	3.0	0.417	Yes
Bi-dir	Omni-dir	#09	27.1	2.2	3.321	Yes
Bi-dir	Omni-dir	#10	6.2	1.7	9.562	No

Table 3.2: Two-tailed student T-test results. Each row shows the comparison of a method pair on one configuration. Column “gain” is the performance gain percentage of method 2 over method 1, and is calculated by dividing the means minus one. Column “Different” shows whether according to the t-Test the results were statistically different with a 95% confidence or not.

### 3.6 Discussion

We have shown that using audio communication can increase the performance of a realistic group task significantly. This can be done even by using simple robots, each equipped with a speaker and two microphones as a bi-directional sensor, i.e. using one bit of audio signal direction information. In this task and its implementation there are many different aspects which could usefully be studied. There are several implementation parameters which can be changed, potentially affecting the overall performance of the system. Nonetheless, we believe that these simulation results are a useful predictor of the gross behavior of this system in the real world.

Due to its physical interaction with the robot's environment, audio promises to be a very interesting sensor modality. Future work could explore using frequency information in ambient and transmitted audio signals, for example to characterize locations by their "sound". As an example, our lab, offices and hallway have very different ambient sound and sound-reflection properties: each can be easily recognized by sound alone. This could be useful for localization.

## Chapter 4

# Going to the Real World

The Sounds Good experiment showed how audio could be used to enhance the performance of a multi-robot system. It was a proof-of-concept and preparation for the extension of this type of signaling into real-world applications. In the following sections we will discuss some possible scenarios in which this type of communication can be used.

### 4.1 Sounds Good in the Real World

After Sounds Good, the obvious extension would be to repeat the same experiment in the real world and on real robots. In that experiment we assumed that our robots are equipped with directional audio sensors. Our simulation showed that even a simple bidirectional sensor configuration can be good enough to increase the performance significantly. Running this experiment on real robots will be a validation of the simulation results.

The exploration/transportation experiment assumed that the audio sensor can distinguish a few message types and their direction information. This has been shown to be possible either by using microphone arrays [51] to achieve high precision, or even by using a simple two microphone configuration. PeanutBot<sup>1</sup>, the audio homing robot built in Cornell University, uses three microphones to detect the direction of the sound source and move to that direction. McDowell in [32] showed how acoustic sensors were used in relative positioning for navigation of a robot team in an environment where other positioning systems such as GPS are ineffective.

---

<sup>1</sup>[http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/s2007/ai45\\_hkc2\\_sbs43/ai45\\_hkc2\\_sbs43/](http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/s2007/ai45_hkc2_sbs43/ai45_hkc2_sbs43/)

Extracting audio direction information is one of the engineering efforts that are needed to move the Sounds Good experiment from simulation to the real world. Instead we can use the tools that we already developed in the previous step and use them in a totally different and new application that would be more significant.

## 4.2 Audio Communication in Robotics

There have been several studies of audio communication in robotics. Some example applications that can be changed to use audio communication in a multi-robot configuration are as follows.

Girod in [15] used acoustic signaling to derive a relative localization method for sensor nodes spread in an environment. By combining the radio frequency communication and sound transmission in a sensor network system, he obtained accurate distance information between the elements.

Østergaard in [36] generated locally-sensible sound gradient from several fixed emergency alarms. A team of robots each equipped with a single directional microphone solved a multi-robot-multiple-task allocation problem. They also showed the effectiveness of using multiple sound frequencies in a noisy environment.

Another possible use of audio is as an added piece of information in self organizing systems. Holland in [18] and Melhuish in [34] demonstrated use of a biologically inspired chorusing mechanism in controlling the size of the robot clusters or to form traveling groups of fixed size.

## 4.3 Concurrent Computing

Another group of applications which can be fitted to use audio communication are the algorithms used in concurrent programming and distributed systems. To coordinate among processes or to implement a concurrent program in a distributed system, message passing or shared memory should be used [9]. In *distributed robotic systems* where robots are mobile agents, communication plays an important role.

### 4.3.1 Mutual Exclusion

One of the classical problems in a distributed system is to achieve mutual exclusion among multiple homogenous agents without the existence of a central coordinator. A solution to mutual exclusion can be used to solve many other distributed system problems.

For tasks in a physical world that are shared among multiple robots, avoiding conflicts and resource allocation is an important problem. Passing through a narrow corridor, getting access to a single charger or selecting a leader are examples of these kinds of problems.

A popular and easy solution to this class of problems is to solve it through robot's interaction with physical objects. For example, the possession of a unique object can be used to select which robot gets the exclusive access to a lock. This physical fulfillment of mutual exclusion sometimes can be very expensive and hard to achieve. Robots might need to fight physically to get the object and they should be equipped with grippers and other means of picking up the object. Another solution to that problem is to incorporate communication and use a distributed algorithm for making the decision of who gets the lock.

### 4.3.2 Local Mutual Exclusion

One of the properties of the audio based communication is its locality with a reasonable scale compared to the robot sizes. This property cannot be easily achieved in other types of communication media. Using locality of audio, some of the classical distributed algorithms can be implemented spatially. For example in case of mutual exclusion and by using audio communication robots can achieve a *local mutual exclusion*. This means that they can guarantee mutual exclusion over the hearing range of the robots.

An autonomous Multi-Robot system is a good example of a distributed system. Like any other distributed system if multiple agents want to access a shared resource at the same time, locks need to be put in place to avoid race conditions. An implementation of local mutual exclusion for robots can be used in many applications. A general application is when there are resources spread around the environment at different locations. Even if identifying and uniquely labeling these resources is not possible for robots, they can use a local mutual exclusion algorithm to get access to a local resource. Charging with multiple charging stations spread in the environment is a good sample usage scenario.

The same communication based mutual exclusion algorithm can be used to avoid physical fights and conflicts in many other cases. Entering a narrow corridor, selecting a leader, forming territories are other examples that can benefit from audio-based local mutual exclusion.

### 4.3.3 Distributed Coordination

Other distributed methods that are used for coordination and agreement include election, multi-cast communication and consensus. The ideas used in the solution to the mutual exclusion problem are also helpful in solving the other problems.

## 4.4 Final Choice

Because of the novelty of implementing local mutual exclusion and the fact that a biologically inspired communication medium can be used to solve this distributed system problem, we decided to focus on that.

## 4.5 Message Passing Communication

In the Sounds Good experiment, the audio messages passed between robots were simple messages with a tiny amount of information. But the robots had the signal processing ability to detect the intensity and direction of the audio messages. In the selected application, which is to implement an audio-based local mutual exclusion, the messages are more complex and contain more information; thus letting us study the more complicated distributed algorithms on a multi-robot system. We also chose not to use the direction information or the audio intensity (as a direct function of source-destination distance) information. This will decrease the complexity involved with the selection and calibration of sensors and also avoid dedicating significant computation power for signal processing.

This audio-based message passing system is implemented as a reusable module. In the next chapter we will discuss the design of this network communication layer for transmission of data over audio.

## Chapter 5

# Nava: Audio Communication Layer

*In the Persian language **navā** means melody, tune; a motive; a mode and also a song. It is also the name of a musical scale in Iranian traditional music [16].*

*Nava* is an implementation of an audio based broadcast network layer. It encodes small packets of data and broadcasts them using a Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) method. It does not have the ability to detect collisions with perfect reliability but it tries to avoid them. There is a checksum in each packet to detect corruptions due to noise and possible collisions. It also reports information about the quality of the received data and the existence of carrier frequencies to the higher layers.

### 5.1 RTTY Communication

The Nava audio communication layer is mostly based on a package named RTTY version 2.1 written by Jesús Arias<sup>1</sup> [1]. It is licensed under the GNU General Public License<sup>2</sup>. The original RTTY software is for generation and detection of Radio-Tele-TYpe (RTTY) signals using a normal sound card. It supports three types of decoding frequency modulated signals: Morse code, Frequency Shift Keying (FSK) and High-Level Data Link Control (HDLC) protocol.

In fact, many cheap modems available these days also use a simple technology similar to RTTY. The hardware part of these modems only provides interfacing with the phone line

---

<sup>1</sup><http://www.ele.uva.es/~jesus/rtty/>

<sup>2</sup><http://www.gnu.org/copyleft/gpl.html>

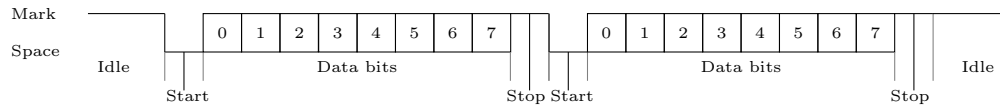


Figure 5.1: Signal levels for two 8N1 bytes sent using RS-232 standard. There are one start bit, eight bits of data, no parity and one stop bit.

and the ability to transmit and receive analog data over wires. Other than the digital-to-analog and analog-to-digital conversions most of the modulation and demodulation is done in software.

### 5.1.1 Modulation

In Frequency Shift Keying, two fixed frequencies are used for zero and one bits. They are called *MARK* and *SPACE* frequencies. The FSK modulator unit in RTTY sends the byte stream, byte by byte, using RS-232 standard. For example in 8N1 format, each byte is sent as a start-bit which is always zero, 8 bits of data and a stop bit that is always one. Zeros and ones are sent with *MARK* and *SPACE* frequencies and an amplitude envelope is used for smooth transition of frequencies between zero and one. See Figure 5.1.

FSK encoding combined with a Universal Asynchronous Receiver/Transmitter (UART) can be easily used to send and receive streams of bytes between two points. Sometimes to have full-duplex communication over a shared medium, four frequencies are used instead of two. Each side will be assigned to two unique frequencies for zeros and ones. This way there will be no conflict when both sides are communicating simultaneously.

There are other possible methods for a broadcast based communication using audio. For example, each robot can use one or more separate unique frequency for communication. These frequencies can be pre-assigned or selected randomly. By assigning a separate frequency to each robot, the robots can avoid collisions when communicating.

The problem is that in practice the number of possible frequencies is not unlimited. Because of the definition of audio communication, we are limited to audio frequency range. Also the number of frequencies that can be used is limited to a maximum number because of our constant sampling rate and the fixed bandwidth assigned to each channel. This makes this method non-scalable unless the underlying protocol can support collisions. If collisions

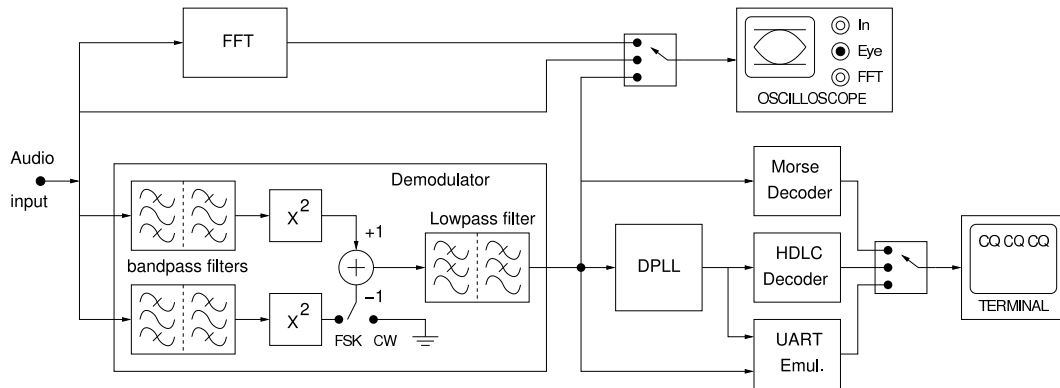


Figure 5.2: Block diagram for demodulation in RTTY program. ©Jesús Arias. Used by permission.

can be handled then this method will have fewer collisions than FSK but instead it will be more computationally expensive than FSK. In FSK, it is enough to be able to detect the two main frequencies but detection of multiple frequencies needs more complex operations such as Discrete Fourier Transform.

To humans, the FSK generated signal sounds similar to normal computer modems or Fax machines. But if sent with a high bit per second rate and in small chunks of data they are heard as short chirps.

### 5.1.2 Demodulation

Figure 5.2 shows the demodulator section of RTTY. In order to decode the FSK encoded signal, RTTY first uses the sound card to digitize the input signal. This signal is passed through two parallel band-pass filters. Each band-pass filter is tuned to one of the MARK and SPACE frequencies. The output power (RMS) of the MARK filter is then subtracted from the output power of the SPACE filter to achieve the original transmitted stream. To extract the actual bits and bytes, a digital phase-locked loop (DPLL) and an emulated UART is used.

## 5.2 Broadcast over Audio

### 5.2.1 Network Layer

The RTTY software was designed to extract the data mostly when there is a single transmitting agent in the environment. To build a shared medium broadcast based system that allows communication between multi-robots, we used ideas similar to those used in wireless networks.

For communication between multiple nodes on a shared medium, “Carrier Sense - Multiple Access” is a good choice. In the CSMA protocol, each node has the ability to sense the existence of the carrier signal. The existence of the carrier on the shared medium means that another node is already sending data and thus, the current node transmitter should avoid sending data. In audio communication using FSK protocol, the detection of any of the MARK or SPACE frequencies is equivalent to sensing the carrier.

### 5.2.2 Collisions

The problem with pure CSMA is when two nodes want to transmit at the same time. In that case both of the nodes will listen on the shared medium for the existence of the carrier and then it is possible that both start transmitting at the same time. There are methods for detection of collisions. For example in a wired network, a collision can be detected by measuring the signal on the wire. If the voltage is higher than the voltage applied to the wire by a single node, a collision is assumed. This class of methods is referred to as *CSMA-CD*.

Due to the nature of some media such as air in audio communication, collision detection is either not possible or very hard to do. In audio communication a node cannot listen while transmitting because the node’s own transmitted signal will obscure any other signal present in the air. The reason is that the sound intensity at any given distance from the sound source follows the *inverse-square law*. Because of this, in close proximity of the source, the sound is much more powerful. This is similar to wireless networks such as Ethernet 802.11.

Wang in [56] proposed CSMA/CD-W protocol for multi-robot systems. CSMA/CD-W uses radio frequency for communication and assumes that there should always be a difference in transmission start time. In CSMA/CD-W collisions are detected by immediately checking for the existence of carrier frequencies after the transmission was done. But the ability to check for the carrier that fast after the transmission end is not easy to achieve and probably

needs hardware support. Especially for audio, a received sound after transmission end can be because of unavoidable distance between speaker and microphone, the echo from robot body or floor, or the delay caused by internal data buffers.

Another possible solution is to try to avoid collisions if possible. This class of protocols is called *CSMA-CA* methods. A simple way of avoiding collisions for the transmitter is to listen for the carrier for a random amount of time and only sending the data when the media seems to be free for that period. This is the method that is used in our implementation.

### 5.2.3 Unreliable Broadcast

Without having a collision detection method, we can only achieve an unreliable network transport layer. Data corruption and data loss may happen because of collisions. Data corruption can be prevented by adding checksum of data to each packet. Data loss can be avoided by expecting the receiving nodes to send acknowledgement of the receipt of each packet.

Because of the nature of audio communication, our system assumptions, the complexity of handling acknowledge messages and the technical difficulty of message routing between nodes, we ended up with an unreliable broadcast network. The collision avoidance tries to keep the network reliable but it does not guarantee it. As a work-around the network layer also provides the information about carrier existence, incomplete packets and noise data to the application layer. Later in Chapter 6 and Chapter 7 we will discuss how the carrier sense information will be used to implement applications on an unreliable network.

### 5.2.4 Implementation

The Nava audio communication layer is implemented as a state based model. The state machine describing the communication layer is shown in Figure 5.3. The states in that machine are as follows:

#### Calibration State

This is the initial state. It sets all the parameters to their initial values. One of these parameters is a threshold on the demodulator's output level. This threshold separates silence from carrier existence.

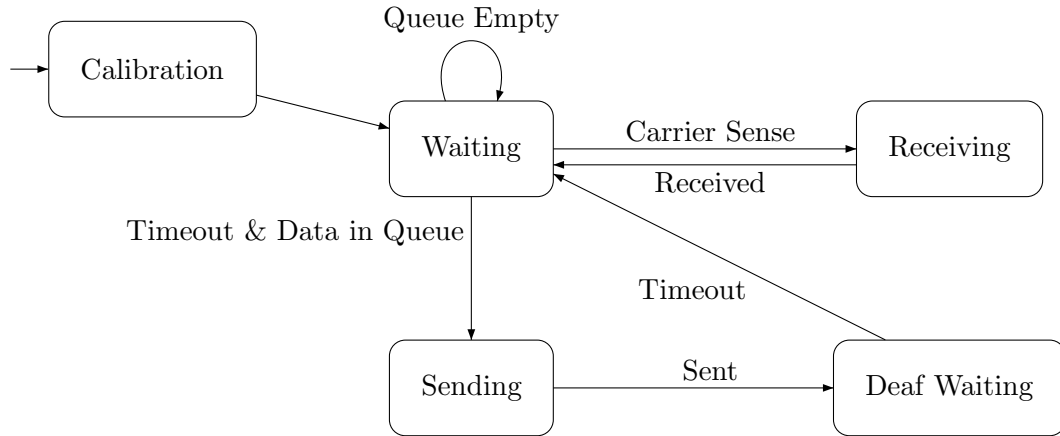


Figure 5.3: The state machine implementing the Nava audio communication layer

The threshold can be either a pre-assigned value or it can be calibrated on startup. In the latter case the calibration state listens for four seconds of silence and sets the threshold based on that.

After the calibration is done, the system goes to the waiting state.

### Waiting State

In this state, Nava listens on the air for the existence of carrier. Carrier is detected by measuring the output power of the demodulator over the equivalent of  $\frac{1}{16}$  of a single bit transmission time. The value is compared against the silence threshold to detect the carrier existence. The time to search for carrier should be long enough to compensate for the possible noise in the environment and on the other hand it should be short enough not to miss a large portion of a bit. All the timing units in our implementation are based on this  $\frac{1}{16}$  of a bit time. This means that if the bit rate is increased all the system will perform faster accordingly.

If the carrier is detected, the system immediately goes to the receiving state. If no carrier is detected after a random time and if there is data in the queue ready to be sent, the system goes to the sending state.

### Sending State

The Nava network layer sends data in packets. Before sending a packet, first a *break signal* is sent. A “break” is a fixed level signal for longer than a character length. In our network

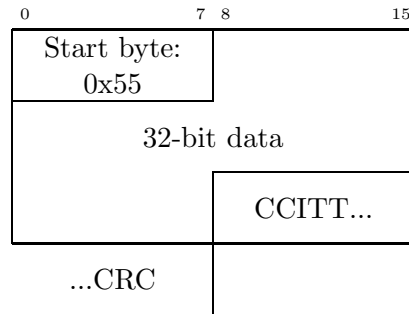


Figure 5.4: Nava packet format

layer, 20 bits of zero and then 20 bits of one form a break signal. This break level will make it easier for the receiving node to synchronize to the data stream.

In Nava, a data packet is a 7 byte packet. It contains one header byte with a fixed value (0x55), four bytes of data, and two trailing bytes of checksum. The checksum is in the standard CCITT format [46]. An illustration of packet format can be seen in Figure 5.4.

The bytes are send in 8N1 format, which means first there is one start bit, then eight bits of data and then on stop bit. As there is already a checksum at the end of the packet there is not need for parity bits. The time for sending each byte is equal to the time for sending 10 bits of data.

The total time to send a packet is the time to send 40 bits of break signal plus 7 bytes of data which equals to the time to send  $40 + 10 \times 7 = 110$  bits. At 300bps each packet will take near 367ms to be sent.

After a packet is sent, the system goes to the Deaf waiting state.

### Deaf Waiting State

Every time an audio message transmission through the speaker is finished, an echo of it can be heard by the system after a short while. This is due to multiple factors causing a delay in the system. Internal buffers used in our system and the unavoidable physical distance between the speaker and the microphone and the echo from the robot body are some of these factors. To avoid this to be interpreted as a wrong carrier signal, right after sending a packet, the system goes to a deaf state.

In the “deaf waiting” state the audio signals will be discarded for a short fixed period of time. This fixed time is set to be shortest possible time that is larger than the echo time. After this period was over, the system goes back to the waiting state.

The fixed waiting time in this state should be smaller the time to send a packet. This will guarantee that an existing carrier will not be missed and thus keeping the integrity of carrier sense that is needed for the experiment that will be discussed in Chapter 7.

### Receiving State

The system enters the receiving state when it senses the carrier in the waiting state. After entering the receiving state the system immediately starts decoding the received data and parsing the data packet. Other than decoding the data bytes, the UART module in RTTY also checks the integrity of bytes receive based on the start bit and stop bit values.

If the first received byte is not the header byte (0x55), the UART found an error in receiving the data or the checksum does not match the data, detected noise is reported. Reporting the noise information allows the higher application layer to know of carrier existence even in the case when no meaningful data is received.

The received data can be one of the following:

- Data packet
- Noise
  - **Zero byte noise:** The UART could not decode any single byte.
  - **One byte noise:** The start byte is wrong.
  - **Seven byte noise:** Wrong checksum or corrupted data inside the packet.

Nava also reports the quality information which might be helpful in getting the information about the environment and to get an approximate node distance measure. The quality is calculated from the sum of the absolute values of the MARK and SPACE filters.

## 5.3 Player Driver

Nava is implemented as a plug-in driver for Player which can be used on any computer running Linux with a sound card. It uses small amount of CPU power because of the simple demodulation algorithm that it uses. It can be accessed from any robot controller through the Player's Opaque interface. The behavior of this communication module can be simulated using the audio model that was discussed in Chapter 2. The audio model can simulate message collisions as well.

## Chapter 6

# Mutual Exclusion for Robots

### 6.1 Mutual Exclusion

Mutual exclusion algorithms are used to avoid multiple agents accessing a single shared resource at the same time. A mutual exclusion algorithm should guarantee that the capacity of the protected resource is never exceeded. It should avoid starvation where a process never gets to access the resource thus satisfying fairness. Deadlocks should also be avoided in which two or more agents wait for the others to release a resource before releasing their captured resource.

On a single computer running multi-threaded processes, mutual exclusion can be achieved using a shared semaphore between threads. By atomic increments and decrements of this semaphore each thread can get safe access to the protected resource [42].

### 6.2 Mutual Exclusion for Robots

The following sections describe some of the methods that can be used for mutual exclusion in multi-robot systems. In these algorithms we assume the robots to be autonomous and running software on-board. We also assume that all robots are honest agents working towards the success of the group.

#### 6.2.1 Physical Object

One way to achieve mutual exclusion in robotics is to satisfy it through physical properties of objects. For example the possession of a single object can decide which robot gets the

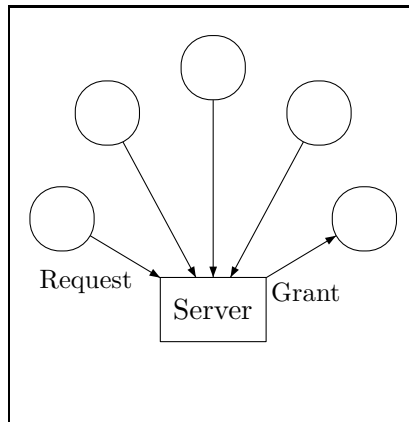


Figure 6.1: Single server mutual exclusion: a single central authority grants lock to the requesting clients in order of their request times.

lock. The robots have to compete with each other to get that object.

An applicable field for this method is in robotic soccer and its competitions the RoboCup<sup>1</sup>. In a soccer game, the robot that carries the ball can take the leadership of the team. A similar approach can be implemented using other physical attributes such as docking on a home base that only fits one single robot. This method is dependent on the application and sometimes may be costly because of its need for extra mechanical equipment such as grippers.

### 6.2.2 Lock Assignment Authority

One solution to the distributed mutual exclusion problem is to have a single authority that manages other agents' access to the shared resource. The authority will use the same concept of a single computer lock such as a semaphore to store the current state and to authorize a single agent in case the lock was available. See Figure 6.1 for an illustration of this method.

For the central authority, an implementation for the server could be based on knowing the current lock owner and also the order in which the lock requests from the requesting nodes were received. The server then grants the clients in the same order whenever the lock is available. This is similar to Lamport's bakery algorithm [28], in which each thread

---

<sup>1</sup><http://www.robocup.org>

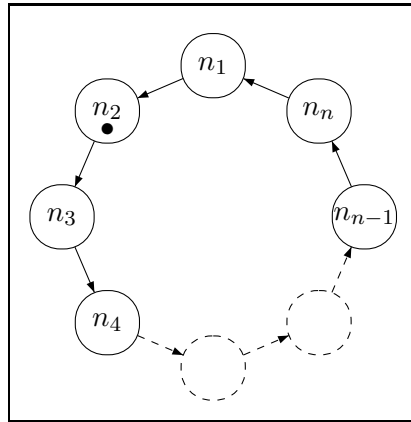


Figure 6.2: Achieving mutual exclusion with token passing. The node which has the token can grab the lock.

is assigned a number at entrance like it is done in a real bakery. This sequence number can also be replaced with a time stamp of sufficient resolution.

The problem with this method is that one of the agents, that take the role of the authority, should be different from others. If all the agents are similar they all need to first negotiate and select one agent as the authority. This election problem itself is a distributed systems problem.

### 6.2.3 Token Passing in a Ring

Passing a token in a ring is one of the simplest ways to achieve mutual exclusion. It needs all the agents to be ordered and each agent knowing its next agent in the ring order. The possession of a virtual token specifies who can access the lock. If the node that has the token does not need the lock, it releases it to the next node.

The need for all agents to be ordered in a ring and each agent knowing the next one makes this method hard to implement in a robotic system. In case scalability while running is needed, there should be mechanisms in place to add and remove nodes from the system and inform the affected agents about the new configuration. The other major problem is the creation of the token and the guarantee of it being the only token in the system. If all robots are homogenous the token creation needs to be done through an election algorithm.

Still this algorithm can be simplified to be useful in some applications, for example the

first token can simply be created by a human when the experiment starts or through other similar means.

#### 6.2.4 Logical Clocks

In a distributed system with nodes being able to communicate with each other, a desired distributed mutual exclusion algorithm would be a system where the nodes pass messages to each other in case of a lock request. Then each node individually decides whether it can access the lock or not. This individual decision should be guaranteed to satisfy mutual exclusion properties and should not conflict with other nodes' decisions.

If each agent in the distributed system can follow the same decision process as a central authority and make the same decision then the mutual exclusion can be achieved without a need for the central server. The central authority needed to identify each node. It also needed a numbering system to sequence the client requests based on the request arrival time.

For the robots each separately following the decision process of a virtual authority, identifying other distributed agents is straightforward from the message headers. For assigning sequence numbers to requests, Lamport logical clocks [29] can be used.

#### Lamport Logical Clock

A single process running on one machine can order events based on its own clock. But in a distributed system, where the events are the messages passing between agents, ordering can be done with messages labeled with the time-stamp of the transmitter. The clock on all agents should also be synchronized between all agents. But synchronizing clocks perfectly is not possible [29]. There is also the delay between when the message is sent and when it is received which is not always a fixed value. This causes ambiguity even if the clocks were synchronized.

Lamport proposed logical clocks to quantize the concept of an event happening before other events in a distributed system. The idea is based on the facts that [9]:

- The order of the messages from a single agent is the same order that agent generates them and numbers them.
- For messages passed between agents, the event of sending message *happened before* the event of receiving the message.

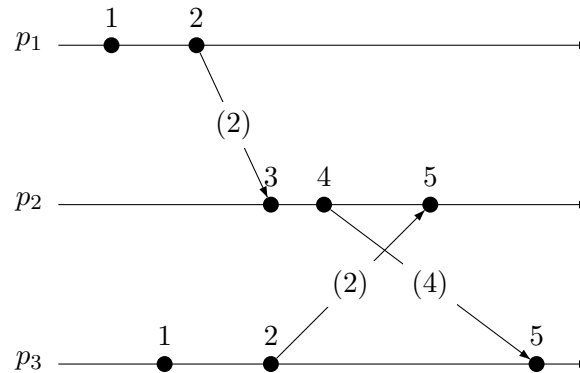


Figure 6.3: Lamport logical clocks use happened-before relation between events to time-stamp them.

This “happened before” relation, often denoted by:  $\rightarrow$ , provides us with a partial ordering of the events. Happened before can be described with these rules:

- If two events  $a$  and  $b$  are both from a single agent and  $a$  is generated before  $b$ , then  $a \rightarrow b$ .
- If  $a$  is the event of a sending a message from an agent and  $b$  is the event of receiving the same message by another agent, then  $a \rightarrow b$ .
- The happened-before relation of events is transitive. If  $a \rightarrow b$  and  $b \rightarrow c$ , then  $a \rightarrow c$ .

Each agent in the distributed system keeps its own logical clock and sets it based on the happened-before relation:

1. Each agent increments the logical clock before any event happening in that agent.
2. Each message sent from an agent is time-stamped with its logical clock.
3. On receiving a message, the agent updates its own clock to the maximum of their own value and the time-stamp of the received message. According to the first rule the clock will also be incremented by one.

See Figure 6.3 to see how messages are time-stamped and how the logical clock is updated by each agent.

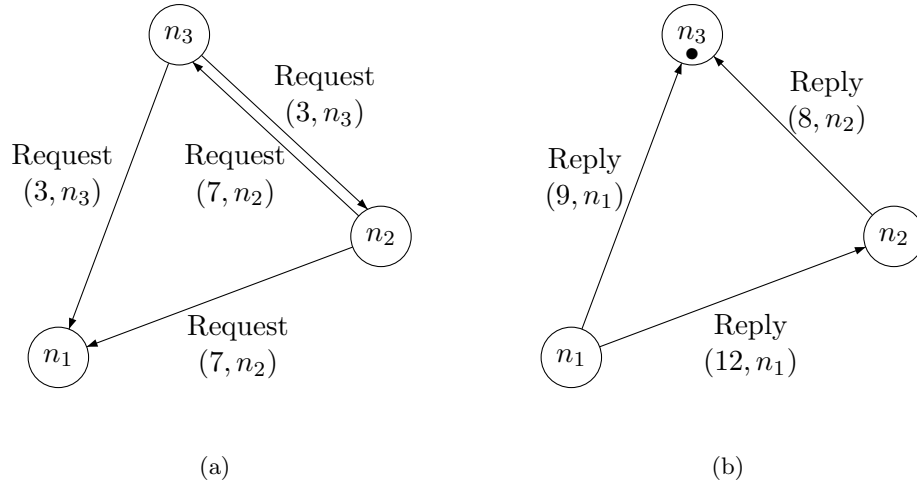


Figure 6.4: The Ricart-Agrawala algorithm for mutual exclusion uses messages that are time-stamped with Lamport clock and the node id:  $(T_i, i)$ . (a)  $n_2$  and  $n_3$  request for lock. (b)  $n_1$  replies to both.  $n_2$  replies to  $n_3$  but  $n_2$  defers the reply and grabs the lock.

Logical clocks only provide a partial-order relation. To achieve total-order among the events, each message can be labeled with both a time-stamp and the agent’s identifier. This identifier can be used to break the symmetry in case of a tie of logical time stamps. As we said before, each agent’s identifier is guaranteed to be unique and there is a total order relation between them. The  $(T_i, i)$  pair of “time stamp - node id” is now a total order.

We say that  $(T_i, i) < (T_j, j)$ , if and only if

$$\left\{ \begin{array}{l} T_i < T_j, \text{ or} \\ T_i = T_j \text{ and } i < j \end{array} \right.$$

### Ricart-Agrawala algorithm

Ricart in [38] proposed a mutual exclusion algorithm for computer networks based on Lamport logical clocks and a total order. The Ricart-Agrawala algorithm works by each agent requesting for access to the lock and then waiting for all other agents in the system to approve the request. Each agent in the system can be in one of the following three states of the lock being **available**, **requesting** the lock, or lock **held**.

An agent requests the lock by broadcasting a *request message*, time-stamped with the

logical time  $T_i$  and its unique identifier  $i$ . Upon reception of this request, all other agents update their logical clock and then take one of the following actions:

- If the agent does not already have the lock (“available” state), it immediately sends a *reply message*.
- If the agent already has the lock (“held” state), it will defer the request until when it releases the lock.
- If the agent is requesting the lock too (“requesting” state), it compares the request time and id pair with its own time and id:
  - If  $(T_i, i) < (T_j, j)$ , it sends a *replay message*.
  - If  $(T_i, i) > (T_j, j)$ , it defers the request until its own request is satisfied.

The requesting agent will get the lock when it received replies from all other agents in the system. Because of that, each agent should have the knowledge of the total number of agents in the system. Figure 6.4 shows a sample scenario of distributed nodes solving mutual exclusion.

### 6.3 Mutual Exclusion over Audio

Because of the nature of audio its range is limited and it can be blocked by obstacles. That is the reason why mutual exclusion over audio is only guaranteed over the audio range.

But the network layer we implemented in Chapter 5 is still unreliable over that range. The communication module tries to avoid collisions but it does not guarantee it. Also because of the noise in the environment messages may get corrupted.

To solve this we added some assumptions to our system and also used the features provided by the communication layer. These assumptions are

- The mutual exclusion is only guaranteed over the minimum range of all speakers and microphones in the system.
- As implied by the previous assumption, we assume that there are no faulty microphone or speaker.
- The robots do not move if they are requesting the lock or already hold the lock.

The communication module provides us with carrier existence information. Carrier detection in that module is defined as the existence of audio frequencies in the communication range. Because of our assumptions and this feature we can detect all the transmissions from other robots even if the message was corrupt.

We designed our system and set our assumptions so that our communication module can capture all of the messages: either as a correctly received packet that is checked for integrity using a checksum or as an incomplete message or as a noise through carrier detection mechanism.

### 6.3.1 Our Algorithm

In Ricart-Agrawala algorithm the idea was to get an acknowledgement from every other robot before holding the lock. Implementing that algorithm over an unreliable network where the number of robots can also change during the experiment is not easy. It requires implementing buffers and a system of acknowledge messages to make the communication reliable first. And also a system of keeping track of the number of robots through other means.

Here we suggest an approach which in contrast to Ricart-Agrawala, holding and requesting the lock is shown by sending messages and the requesting robot will wait until there is no other robot sending messages before holding the lock. The basic idea is that the robot will acquire the lock only after it does not hear anything for a fixed amount of time. It is the goal of the requesting robot to transmit its request to other robots and convince them to stay silent.

Robots will use the same method of achieving a total order to decide which agent acquires the lock. If a robot decided that it is still not its turn to get the lock, it stays silent; letting another robot to hold the lock. When a robot is requesting the lock, it first checks to see if there is any lock held message or request message received recently. If there is a lock held message received it will defer its requests until a further time. If there is another lock request message, the robot compares the  $(T_i, i)$  pair attached to the message data in relation to its own Logical clock. If it found out that the other robot has priority over itself, this robot will defer its request long enough to let the robot get the lock. If none of the above cases were true, the robot starts send the request messages. If the environment stayed quiet for a predefined amount of time, the robot will acquire the lock.

Because we are using an unreliable network, the messages may be received as corrupted.

Action / Event	Ricart-Agrawala	Audio-Based
Request by	Broadcasting	Broadcasting
Grab lock when	All others acknowledged	Others were silent
Prevent others by	Deferring the reply	Announcing frequently
Grant the lock by	Replying to the requests	Staying silent

Table 6.1: Actions and events for agents requesting the lock in the audio-based mutual exclusion method in comparison to Ricart-Agrawala algorithm.

Though, we assumed that the carrier detection feature of the communication module detects every message transmission even if not received successfully. It still might detect noises in the environment as carrier too.

If carrier existence is sensed during the request period but the data could not be decoded into a meaningful packet, the robot will restart measuring the time before holding the lock again. The reason is that the received noise could be the message from another robot that is prior to the first robot. It could also be that the robot itself has priority so it will continue to broadcast request messages.

### 6.3.2 Analysis

#### Mutual Exclusion Requirements

A valid implementation of mutual exclusion algorithm has some requirements. **Safety** of the lock, meaning that it is not possible for more than one agent at a time to access a lock, is one of the required lock properties. The other one is **liveness** which requires that any agent wanting the lock eventually gets access to it.

In our implementation of audio-based mutual exclusion, if the communication was always reliable and all the messages within the audio range were transmitted successfully, the lock requirements would always be satisfied over the local range of audio. The reason is that with a reliable communication the audio-based method is similar to the Ricart-Agrawala algorithm. Knowing that messages will reliably be transmitted between agents, both “safety” and “liveness” are guaranteed by the use of logical time-stamps on the request messages which provides a total ordering of the requests. This total order also makes these methods “deadlock free” and “fair”. Table 6.1 compares audio-based method with Ricart-Agrawala algorithm.

But the Nava communication module is an unreliable communication channel. The communication module is unreliable because message transmissions may get corrupted and also messages may get lost. As long as there is no collision, we assume that all of the corrupted messages will be detected for having a wrong checksum and thus they will be marked as noise/carrier existence. A noise which can be a corrupted message from another robot delays getting the lock and thus satisfies the safeness requirement.

To avoid collisions, Nava first listens for other robots before sending a message. But there is still a chance of collision if more than one robot starts sending messages at the same time. In our implementation discussed in Chapter 7, we improved this by resending request messages several times. By sending multiple request messages before getting the lock the chance of all the transmissions colliding will be decreased exponentially. All the repeated messages are marked with the same logical time as when the first request message is sent. The robot increments its own logical clock every time it sends a message. But this still does not guarantee safety.

One of the advantages of using audio communication in the local mutual exclusion method is the scalability provided due to physical limitations. The maximum number of robots communicating with each other is not related to the size of the whole world nor the total number of robots. Because of the locality of audio, the maximum number of robots that are with the communication range of each other is dependent on the physical size of these robots and the maximum number of them that can fit in the audio hearing range. By controlling system parameters, this physically imposed limit can also be changed. Changing the sound volume, changes the range over which mutual exclusion is performed and thus limits the maximum number robots taking part in a mutual exclusion communication. Refer to Section 5.2.4 for a better description of communication module parameters.

The probability of two robots' transmissions causing a collision can be calculated from the system properties. The programmer also has control over the some of the system parameters. We show that it is possible to set these parameters so that the chance of the collisions disrupting the mutual exclusion safeness to be below a desired probability.

The collision probability itself is dependent on the transmission probability function (a uniform random between one to ten seconds in the Nava communication module), the sampling rate ( $\frac{1}{16}$  of a bit in Nava), message transmission time (367ms in Nava) and the time for a message to arrive from one robot to another robot (transmission to reception delay). All of the previous numbers can be approximately measured and thus an upper bound on

the collision probability of robots transmitting at the same time can be calculated.

We assume that the probability of message collision for two robots is  $p_2$  and  $p_r$  for  $r$  robots. Increasing the number of robots increase the chance of collisions so if  $r > 2$ ,  $p_r > p_2$ . The collision avoidance mechanism in Nava tries to make  $p$  as small as it can.

The user controllable parameters in the system are the sound level and the number of requests before getting the lock. By setting the sound level to a specific value the maximum number of robots in the audio hearing range can be set to a maximum of  $r$  robots. By sending  $n$  requests before getting the lock, the probability of all request messages colliding will drop to  $(p_r)^n$ . Obviously  $p_r < 1$  as it is a probability.  $(p_r)^n$  is the chance of mutual exclusion failing even after  $n$  requests.

By increasing  $n$  the number of requests before getting the lock, we can set the  $(p_r)^n$  probability of mutual exclusion to be below any desired  $\rho$  value, as  $n = \log_{p_r}(\rho)$ . This shows how although the audio-based method cannot completely guarantee the safeness in the case of an unreliable network, the probability of it failing can be limited to any desired value by setting the system parameters.

Note that using Nava we cannot logically guarantee mutual exclusion. Instead we provide an approximation of mutual exclusion with some probability. To fully fix the collision issue it is possible to implement a collision detection system similar to CSMA/CD-W [56] protocol or a reliable transmission protocol using “acknowledge messages”.

### **Excessive Noise Effects**

To achieve local mutual exclusion, the proposed algorithm restarts the timer whenever it senses carrier existence that can be either noise or another robot’s transmission. The noisier the environment it will take longer for the robots to negotiate and thus delay getting the lock. In the worst case it is also possible that the noise completely blocks the communication. This problem can be worked around by changing the frequencies used or the bit transfer rate of the communication module. The underlying modulation code can also be enhanced using more error-resistant techniques.

## Chapter 7

# Local Mutual Exclusion Demonstration

In this chapter we discuss the implementation details of the mutual exclusion algorithm. Later in Section 7.2 we demonstrate the use of the local mutual exclusion method in a charging application.

### 7.1 Implementation

The local mutual exclusion is implemented as a Player client. This allows the controller to be tested both in simulation connected to the Stage simulator and also on real robot hardware.

The Nava network layer can transmit four-byte message packets. Each message contains one byte for robot id and one byte for message type, that is either a lock **request** message or a **lock** held announcement. Two more bytes are used for storing the Lamport time. You can see an illustration of the message packet in Figure 7.1.

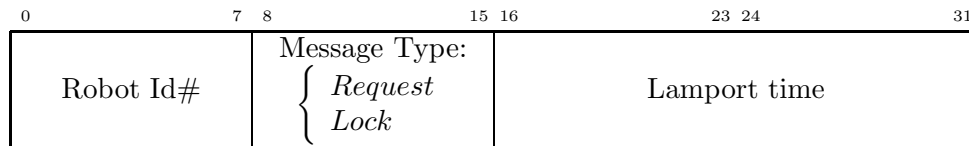


Figure 7.1: The format of four-byte audio message of mutual exclusion experiment.

The logical clock is initialized to zero. It is incremented on every message that is sent or every noise that is received. For every data message received, the logical clock is set to one plus the maximum of the current local clock and the message time-stamp. When resending the same request, the time field of the message is the time of the first request message. But the internal logical clock is incremented because sending is also considered an event.

Each agent records its state of being **Released** when it does not need the lock, **Wanted** when it requests the lock, or **Held** when it already has the lock.

In the “Released” state, the robot does not need the lock. In this state, the controller does not transmit any audio messages. It will only update its logical lock based on the received messages.

When the lock is held, the controller frequently announces that with a message of the type “Lock”. This will prevent any another robot in that region from obtaining the lock.

The robot enters the “Wanted” state when the application requests it. This happens when the application needs access to a shared resource. When the robot enters the “Wanted” state, it stores the Lamport clock as the request time in  $T_i$  and starts a count-down timer that is initialized to fixed  $t_{grab}$  seconds. This timer runs on real-time clock not the logical clock.

The robot sends multiple messages of the type “request” every  $t_{request}$  seconds before the count-down timer reaches zero. During this period any received sound, no matter if it is a message packet or just noise, will force the count-down timer to restart from the  $t_{grab}$  again. When the timer reaches zero, it shows that there was not any sound while timer was running and so it is safe for the robot to go into the lock “Held” state.

While in the “Wanted” state if the robot hears a message indicating that there is another robot requesting the lock, it compares the pair of request time and robot number  $(T_i, i)$  with the received message’s  $(T_j, j)$ . If the total order of the pairs shows that the other robot is prior to the current robot, the lock “request” message will be deferred to a later time. It will be deferred until at least  $t_{grab}$  seconds later. Deferring lets the other robot get the lock. The current robot also sets the lock timer to  $t_{grab}$  seconds after when the first request message will be sent. This is for the robot itself to avoid getting the lock without sending any requests while in the silent mode.

A pseudo-code implementation of the local mutual exclusion algorithm is described in Program 7.2. The constants, global variable and functions needed by that pseudo-code are defined in Program 7.1. The states and transitions of this implementation are illustrated in

the finite state machine of Figure 7.2.

Normally the lock announce time ( $t_{lock}$ ) is set to be shorter than the time between requests ( $t_{request}$ ). The reason is that for a robot that is in the lock “Held” state it is more important to deliver its lock held message to other robots rather than receiving their messages. But a robot in the lock “Wanted” state should be try to deliver its request and receive other robots requests to make a fair decision. The  $t_{grab}$  lock should also be long enough for receiving multiple lock requests and lock held messages.

## 7.2 Application: Charging

In our Autonomy Lab, one of our goals is to build self-maintaining robots. A robot that can recharge itself when needed, can maintain its autonomy for a long period [60]. Power for a self-maintaining robot is a valuable resource.

As a sample application for the local mutual exclusion algorithm which is also in the direction of the research in our lab, we demonstrate a multi-robot exploration, conflict resolution and charging application. In this task, we have a large environment with chargers spread randomly around the world. Chargers are marked so that they are recognizable for robots but they are all similar to each other and robots cannot distinguish between different chargers. This is similar to having color markers of a specific color on each charger.

The robots randomly explore the world while doing their normal duty until they need a recharge. Then they will start searching to find a charger. Because there are more robots than the number of chargers in the world, in some cases multiple robots try to recharge themselves at the same time. Instead of physically fighting to access the charger, here the robots use audio based communication and a local mutual exclusion algorithm to decide which robot can use the charger first.

When a robot finds the charger it stops at a safe distance and starts sending lock request messages. After acquiring the lock the robot moves forward to the charger and starts charging. When done, the robot releases the lock and moves away, letting the other robots access the charger.

The system architecture implementing the charging application both in simulation and in real-world is illustrated in Figure 7.3. As we said before the same controller program runs in both simulation and in real-world.

```

global {
   $i \leftarrow$  (unique random) // robot id
   $logical\_clock \leftarrow 0$  // logical clock
   $state \leftarrow$  RELEASED // current robot state
   $T_i$  // request's logical time-stamp
   $transmit\_timer$  // count-down timer for transmission
   $lock\_grab\_timer$  // count-down timer for getting the lock
   $grab\_time \leftarrow 40s$  // constant: time to get the lock ( $t_{grab}$ )
   $request\_time \leftarrow 15s$  // constant: time to send request message ( $t_{request}$ )
   $announce\_time \leftarrow 10s$  // constant: time to send lock message ( $t_{lock}$ )
}

// Application Interface
procedure REQUESTLOCK()
{
   $state \leftarrow$  WANTED
   $T_i \leftarrow logical\_clock$ 
   $transmit\_timer \leftarrow request\_time$ 
   $lock\_grab\_timer \leftarrow grab\_time$ 
  wait until  $state =$  HELD
}

procedure RELEASELOCK()
   $state \leftarrow$  RELEASED

// Internal functions definitions
procedure UPDATELOGICALCLOCK( $Type, T_i$ )
{
  if ( not  $Type =$  Noise) and ( $T_i > logical\_clock$ )
  {
    then  $logical\_clock \leftarrow T_i$ 
  }
   $logical\_clock \leftarrow logical\_clock + 1$ 
}

procedure POSTPONEGRAB()
   $lock\_grab\_timer \leftarrow grab\_time$ 

procedure POSTPONEREQUEST()
{
   $transmit\_timer \leftarrow grab\_time + request\_time$ 
   $lock\_grab\_timer \leftarrow transmit\_timer + grab\_time$ 
}

procedure SCHEDULEREQUEST()
if  $transmit\_timer > request\_time$ 
then  $transmit\_timer \leftarrow request\_time$ 

procedure UPDATETIMER( $t$ )
   $t \leftarrow t - One\_Second$ 

```

Program 7.1: The variables and functions needed for the implementation of the local mutual algorithm shown in Program 7.2.

```

// Main Program
main
while true
    {
    if NAVA_MESSAGE RECEIVED()
        {
         $(Type, T_j, j) \leftarrow$  NAVA_RECEIVEMESSAGE()
        UPDATELOGICALCLOCK(Type, Tj)
        if state = WANTED
            {
            then {
                POSTPONEGRAB()
                if Type = Lock
                    {
                    // Another robot has the lock
                    // Continue requesting
                    SCHEDULEREQUEST()
                    }
                if Type = Request and  $(T_i, i) > (T_j, j)$ 
                    then POSTPONEREQUEST()
                if Type = Noise
                    then // Do nothing.
            }
        }
    }

    do {
        if state = WANTED
            {
            then {
                UPDATETIMER(lock_grab_timer)
                if lock_grab_timer = 0
                    {
                    then {
                        state  $\leftarrow$  HELD
                        transmit_timer  $\leftarrow$  announce_time
                    }
                }
            }

            // transmit messages
            if state = WANTED
                {
                then {
                    UPDATETIMER(transmit_timer)
                    NAVA_TRANSMITMESSAGE(i, Request, Ti)
                    transmit_timer  $\leftarrow$  0
                }
            }
            if state = HELD
                {
                then {
                    UPDATETIMER(transmit_timer)
                    NAVA_TRANSMITMESSAGE(i, Lock, logical_clock)
                    transmit_timer  $\leftarrow$  0
                }
            }
        }
    }
}

```

Program 7.2: The pseudo-code implementation of the local mutual exclusion algorithm. The global variables and the functions need by this algorithm are defined in Program 7.1

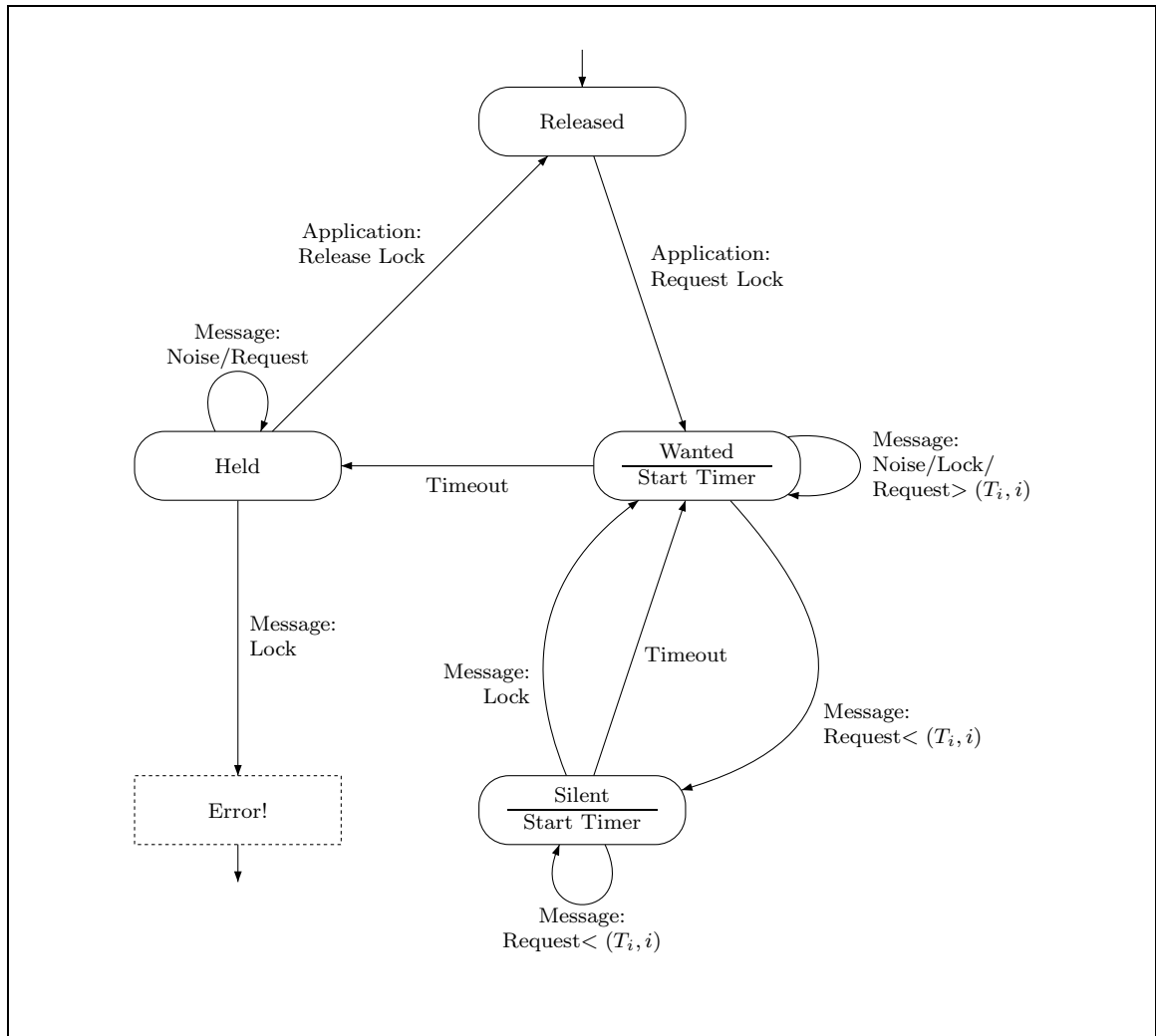


Figure 7.2: The state machine describing the possible states of the local mutual exclusion method. The text under the line inside the states is the action that is performed when entering a state. There are two separate timers to Wanted and Silent states.

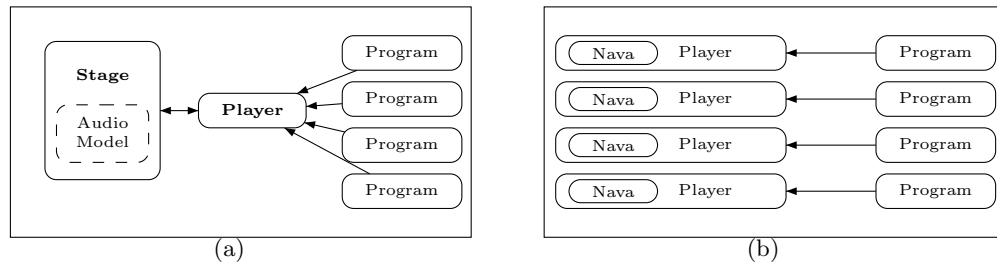


Figure 7.3: System architecture implementing the charging application. The robot controller programs are similar software running on each robot running both the navigation behaviors plus the mutual exclusion algorithm (a) in simulation (b) in real-world.

### 7.2.1 Simulation

Using Stage and the audio model discussed in Chapter 2, we demonstrated the charging task in simulation. Running the charging application in simulation lets us validate the implementation and also to test different scenarios such as large number of robots. Figure 7.4 shows screen-shots of the charging application running in simulation.

In this simulation, each robot controller waits for 40 seconds of silence before getting the lock. The time between lock announcements is a Gaussian random of mean of 10 seconds and standard deviation of 1.0. The time between lock requests was a Gaussian random of mean 15 seconds and standard deviation of 2.0.

Figure 7.5, shows the messages logged from two simulation trials. These simulation runs are very similar to the ideal case where there is no noise in the system. In Figure 7.5(a), only a single robot wants to access the charger. It enters the “Wanted” state at  $t = 3.3$  seconds and starts sending the requests nearly every 15 seconds. The first one is sent at  $t = 14.0$ . Finally the robot grabs that lock at  $t = 43.3$ , forty seconds after it entered the “Wanted” state. After that it sends the lock announcements every 10 seconds until  $t = 99.8$  when it releases the lock.

In Figure 7.5(b), three robots communicate for exclusive access to a single charger. In that trial, two robots: robot #11 and robot #30 enter the “Wanted” state soon after the experiment started. Both robots’ logical request times are set to zero because they have not received any messages before. Robot #30 receives the request message  $(0, R_{11})$  and compares it to its own request time:  $(0, R_{30})$ . According to the total order rules

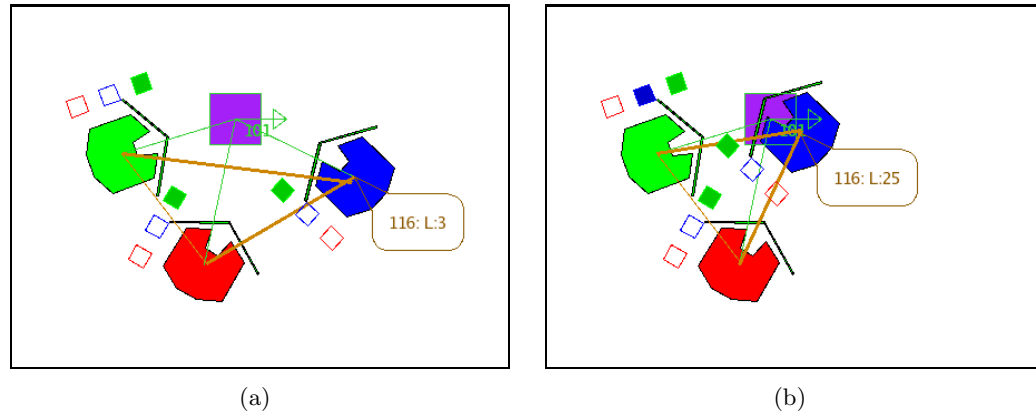


Figure 7.4: Charging application using mutual exclusion in simulation. (a) Two robots communicating to access the charger. (b) A robot is charging while two other robots are waiting for the access.

Robot Id	Request tuple	Time granted
Robot #11	$(0, R_{11})$	44.6s
Robot #30	$(0, R_{30})$	105.5s
Robot #212	$(2, R_{212})$	171.6s

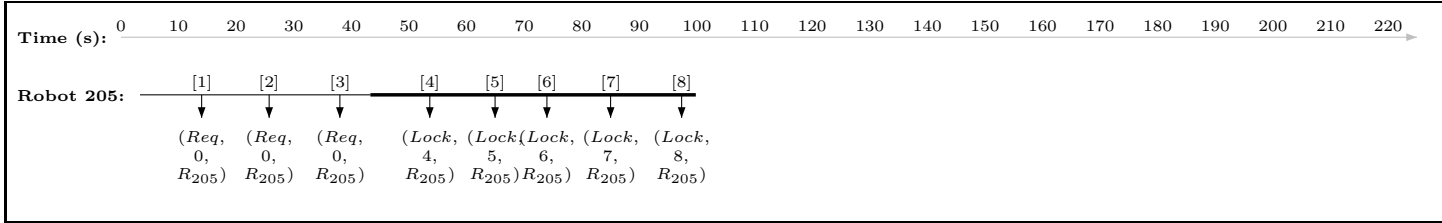
Table 7.1: The request pair and the time lock is granted for the three robots of the simulation trial depicted in Figure 7.5(b).

(Section 6.2.4):  $(0, 11) < (0, 30)$ , thus robot #30 will stay silent for a maximum of forty seconds. This lets the robot #11 to finally get the lock at  $t = 44.6$ .

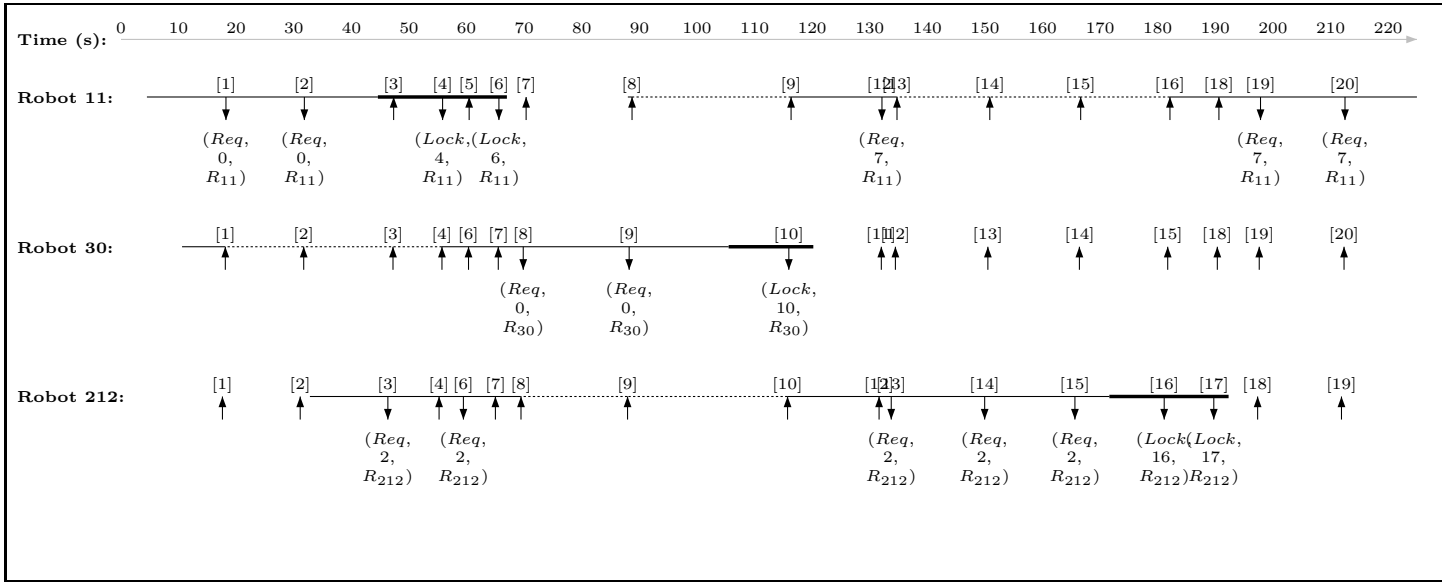
The third robot, robot #212 enters “Wanted” state at  $t = 32.8$  when it already received two messages and so its request Lamport time is 2. It only gets the lock after both robot #11 and robot #30 get and then release the lock:

$$(0, 11) < (0, 30) < (2, 212)$$

The robots are granted lock in the order of their requests. The “request time-robot id” pair and the time lock was granted are shown for each robot in Table 7.1.



(a)



(b)

Figure 7.5: The transmitted messages for mutual exclusion algorithm logged from a simulation run similar to Figure 7.4.  $\downarrow$  specifies a sent message in  $(Type, T_i, i)$  format.  $\uparrow$  is a receive event. The numbers in brackets are logical clocks. Thin lines are when the robot is requesting the lock and thick is when it holds the lock. Dashes lines are when the robot is silent. (a) Only one robot is requesting the lock. (b) Three robots negotiate to get the lock.



Figure 7.6: iRobot Create Programmable Robot.

## 7.2.2 Real-World Demonstration

### System

For real-world demonstration we used Create robots manufactured by iRobot. Figure 7.6 shows a Create robot. Create is a programmable robot that is based on the Roomba robotic vacuum cleaner but without the vacuum hardware. The original Roomba vacuum cleaner has sold over 2 millions units since its introduction in 2002. Tribelhorn in [48] showed how Roomba can be used as a low-cost resource for robotics research and education.

The Create robot is equipped with multiple sensors [20]. On the front it has a bumper with a two bit state showing the right and/or the left side of the robot bumping into an obstacle. Cliff sensors on the bottom of the robot prevent it from falling and a single short-range IR sensor on the front-right let the robot perform right wall following algorithm. Each robot has an omni-directional IR receiver. Robot accessories such as remote controller, Virtual Wall and Home Base can communicate with robot with this IR sensor.

Using a rechargeable battery in the Create robot, the robot can monitor battery voltage, current, temperature and charging state. The robot can recharge itself by docking into the Home Base. The Home Base sends one omni-directional and two directional IR beams. The directional beams are used to guide the robot for docking. See Figure 7.7.

We equipped each of our robots with a Gumstix<sup>1</sup> single board computer. This  $8cm \times 2cm$  gum stick sized computer has an ARM based processor and runs Linux. We used a Connex

---

<sup>1</sup><http://www.gumstix.com/>

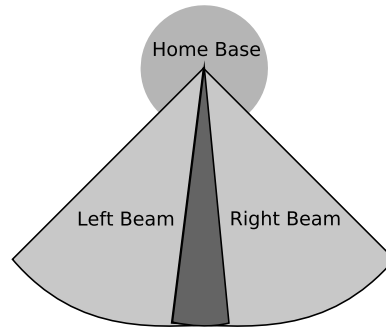


Figure 7.7: Home Base IR beams

400xm motherboard running at 400 MHz and has 16MB of flash and 32MB of RAM. Each of the boards is also attached to a Wifistix for wireless communication and a Roboaudio-TH for stereo sound input/output and also an Atmel ATmega128 micro-controller. Figure 7.8 is a picture of the baseboard prototype.

The robot runs Player and the controller code onboard. Player already has a driver for the Create robot and the Nava module is used as a plug-in driver to Player for transmitting audio messages.

### Demonstration

We run the charging trial with three robots. Figure 7.9 shows three robots communicating before one of them gets the charger. The MARK and SPACE frequencies are set to  $2300Hz$  and  $3700Hz$  respectively and the data-rate is  $300bps$ .

The parameters set to be similar to the simulation. Each robot waits for 40 seconds of silence before getting the lock. The time between lock announcements is a Gaussian random of mean of 10 seconds and standard deviation of 1.0. The time between lock requests was a Gaussian random of mean 15 seconds and standard deviation of 2.0.

In the single robot-single charger configuration and in a quiet environment the robot will get access to charger in 40 seconds most of the time. With two to three robots often the first robot can acquire the lock in a 1-2 minutes time. The greater the number of robots, the harder it becomes to communicate correctly and convince all the other robots to stay silent.

Figure 7.10 illustrates the messages passed between three robots logged from a real run. The first trial with one robot shown in Figure 7.10(a) is very similar to the simulation

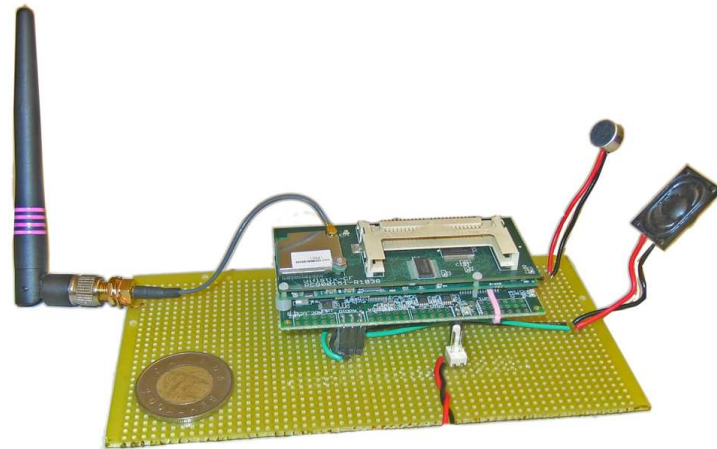


Figure 7.8: Gumstix, Wifistix, Roboaudio-TH, Microphone and Speaker.

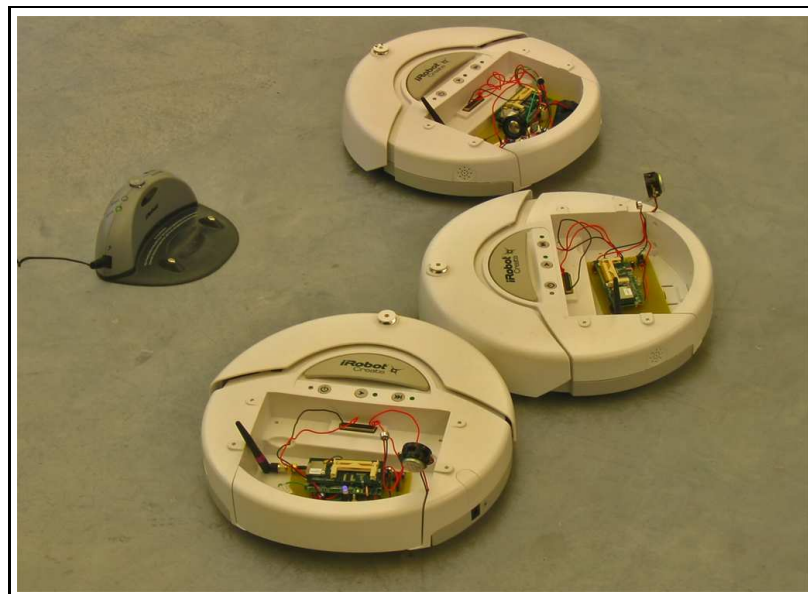
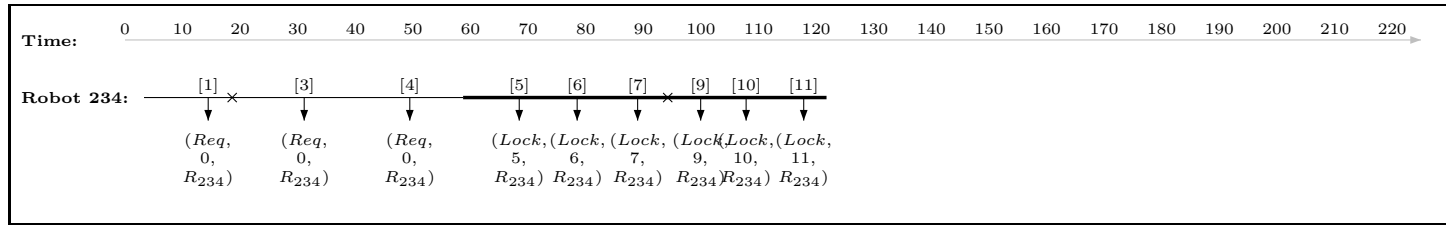
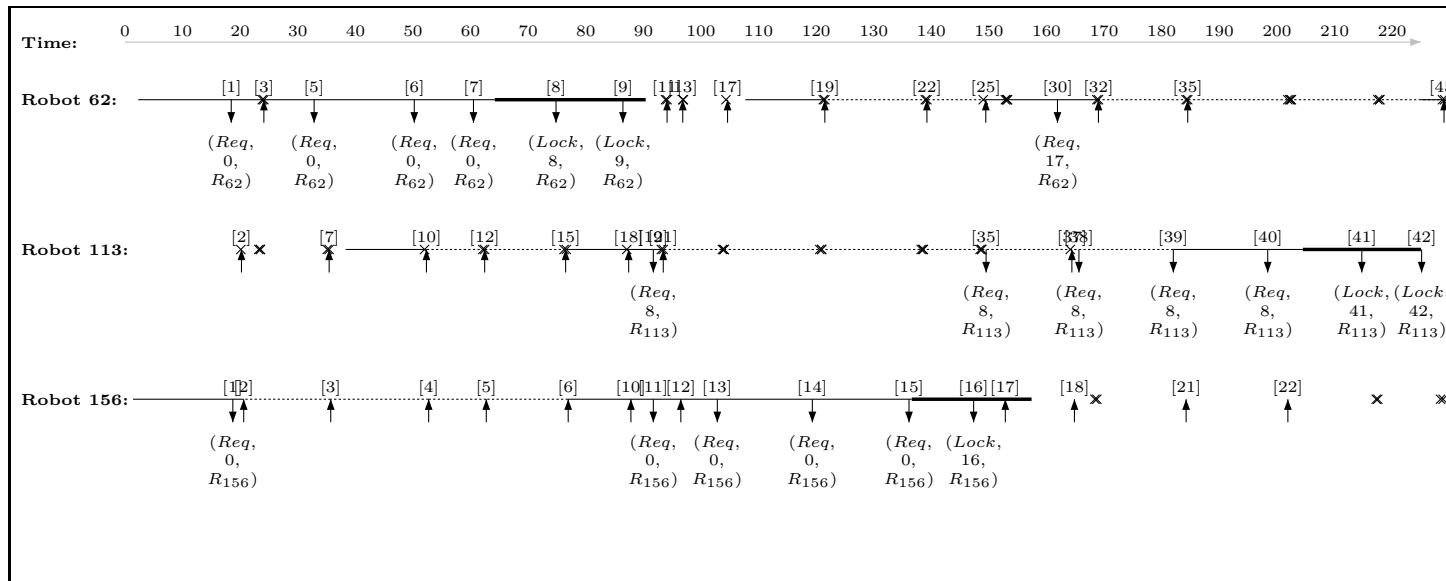


Figure 7.9: iRobot Create robots communicate to decide which one gets access to the charger.



(a)



(b)

Figure 7.10: The transmitted messages for mutual exclusion algorithm logged from the real experiment shown in Figure 7.9.  $\downarrow$  specifies a sent message in  $(Type, T_i, i)$  format.  $\uparrow$  is a receive event.  $\times$  is receiving noise or a corrupted message. The numbers in brackets are logical clocks. Thin lines are when the robot is requesting the lock and thick is when it holds the lock. Dashed lines are when the robot is silent. (a) Only one robot is requesting the lock. (b) Three robots negotiate to get the lock.

Robot Id	Request tuple	Time granted
Robot #62	$(0, R_{62})$	64.2s
Robot #156	$(0, R_{156})$	136.6s
Robot #113	$(8, R_{113})$	204.5s

Table 7.2: The request pair and the time lock is granted for the three robots of the real-world trial depicted in Figure 7.10(b).

depicted in Figure 7.5(a), except that although the robot entered the “Wanted” state at  $t = 3.3$ , it received a noise at  $t = 18.6$ . This noise caused the robot to defer grabbing the lock until forty seconds later at  $t = 58.7$ .

In the multi-robot experiment shown in Figure 7.10(b), three robots are negotiating for access to the charger. Compared to the similar trial in simulation (Figure 7.5(b)), during the real-world run some of the messages do not get transferred correctly and are marked as noise. Also, the delay between sending a message and receiving it on other robots is now noticeable. But still the robots can achieve mutual exclusion. Both Robot #62 and Robot #156 enter the “Wanted” state nearly at the same time when their logical clock is at 0. Robot #113 enters the “Wanted” state later, when its logical clock is at 8. Eventually the robots all get the lock in the same order as their requests.

For comparison, the request time-robot id pair and the time lock was granted are shown for each robot in Table 7.2.

### Future Enhancements

The main problem with the current system implementation is that it is prone to communication errors. There is a high penalty associated with the noise and corrupted messages. A very noisy environment can completely prevent robots from obtaining the lock. This is very dangerous in the charging application. As a solution, in the case of an emergency need for power, the robots can switch back to physical fighting for accessing the charger.

The noise comes from different sources. Ambient noise and humans talking nearby can block the audio communication. When the robots are in motion, they are noisy themselves. When going to a new environment, we try to tune the frequencies and the microphone/speaker configurations to receive less noise. The data transfer bit-rate is another factor that can be adjusted. Spread spectrum techniques can also be used to avoid jamming

by noise.

The message corruption is mainly due to the lack of an error correction method in our system. A better implementation of the communication module can use error recovery techniques and better encoding. This might help to decode more messages successfully rather than marking slightly corrupted packets as noise. Different methods as simple as Manchester coding [43] and Hamming error correction codes to more complex ones like Viterbi demodulation algorithm [55] or Reed-Solomon error correction [40] are some of the possible options.

## Chapter 8

# Conclusions and Future Work

### 8.1 Summary

In this thesis we showed how audio can be exploited for our benefit in multi-robot systems. We studied some of the characteristics of sound such as its locality, intensity gradient and directionality. We developed a model that can simulate a simple version of these properties. To provide physical audio communication between robots, a CSMA audio message transmission module is also developed. Studying two distinct generic applications for audio communication proved this type of messaging to be of great use.

The Sounds Good experiment showed how by adding audio information to the decision making process the performance increased significantly. This performance gain was still achievable for a robot team equipped with simple bi-directional microphones. The audio-based method to achieve local mutual exclusion brought ideas from distributed systems and added the spatial locality of sound messages to them. It could also handle the unreliability of the medium to a certain extent. A charging application demonstrated the local mutual exclusion's use toward building self-maintaining mobile robots.

Other than the results of the research, during this work we released all of the code and tools. These tools are open-source, reusable and modular code that we hope, will be used in different applications and possibly in other research areas.

Our contributions during this research include the following items:

- Implementing a simple and practical audio signal model for simulation of audio-based communication in multi-robot systems. This model is integrated into the Stage robotic

simulator.

- Using the simulations done in the Sounds Good experiment to suggest that audio communication among robot teams increase the team performance even while using simple directional audio sensors.
- Developing the Nava network module which implements a Carrier Sense-Multiple Access protocol with Collision Avoidance (CSMA-CA) for audio communication among robots in real-world. Nava is a plug-in to the Player robotic server.
- Proposing a distributed algorithm for achieving mutual exclusion locally using audio signaling. We demonstrated the use of this method in a sample charging application.

## 8.2 Future Work

This work is just a starting point in the research about audio communication in multi-robot systems. There are lots of implementation details and design aspects. Not all the parameters in the system were tuned to achieve the best results. In the pervious chapters, we described some of the changes that can be made to enhance the system in its respective context. All those along with the following new ideas can be considered in future research.

### 8.2.1 Hybrid Communication

In this thesis, we focused on audio communication being the only transmission medium. It is possible to use a combination of different communication methods to take advantage of other types of media as well. In a sensor network application, Girod in [15] used wireless communication for time synchronization and audio for relative localization of sensor nodes. Wireless networks can provide high data rates and reliability and audio links can provide locality and gradient. The difference of transmission speed of sound versus light or radio-waves can be used to determine the distance from the transmitter.

### 8.2.2 Bio-inspired Communication

The implementation of the Nava audio communication layer uses the traditional modulation methods of digital data communication. It is feasible to use other types of natural sounds such as human voice or animal songs to transfer data. Lopes in [30] studied different types

of modulation for data transmission using audio. His work is a survey of different methods including ASK, FSK, synthesized music instruments, and animal sounds.

Studying animal behaviors to understand the way audio is used among them can be quite beneficial in both understanding the animals and also to discovery of new ideas for use in technology. Wiley in [57] suggested that the animals' vocalizations are evolved to match the physical constraints of acoustic communication in the atmosphere. These adaptations are to reduce the effects of frequency-dependent attenuation, refraction, amplitude fluctuations and reverberation. The same concept can be used in robot audio communication to make the real-world transmission of sound more similar to our audio model. The simple audio model we developed could only simulate the direct-sound propagation.

### 8.2.3 Modern Network Protocols on Audio

In computer networks and packet switching networks, data packets are routed between nodes. A node can relay another node's packet on the way to the destination. In some network protocols a reliable connection is accomplished using error control methods. For example the *Automatic Repeat-Request (ARQ)* makes use of acknowledgements and timeouts to achieve reliable transmission. Congestion-control and flow-control are other examples of modern control methods for data transmission. The same concepts can be developed in an audio based multi-robot network.

Another possible application is for the robots to transfer data from one robot to another by storing them and then physically transporting to another location. Pentland in [37] demonstrates the same idea that is used to provide internet connectivity to rural areas using buses physically transporting the data.

### 8.2.4 Sound Signature

Unwanted noise caused by air-conditioners and various machines in an environment including the robot itself, also the animal and human sounds, can disrupt audio communication between robots. But on the other hand, these sounds can be analyzed to characterize and possibly recognize places [7]. For example the sound signature of a forest is totally different from the sound signature of a class room or a kitchen. The location of the stationary noise sources is also another property of the environment that is sensible by mobile robots [31].

Reverberation of sound in a closed space can also be an identifier of that place. The

*Room Impulse Response* is a histogram showing the power of all the echoes of an impulse sound signal. It is a distinct mark for identifying the location. A robot can emulate an impulse by generating a loud click. Recording the echoes heard afterward estimate the Room Impulse Response.

The location information could be valuable data for robots and other technology products such as location aware sensors. The robots can use all the information and fuse them with the previous knowledge for novel applications.

### **8.3 Final Word**

We demonstrated possibly the first real-world multi-robot system that uses audio communication. We studied audio communication in detail and presented two generic applications that use acoustic transmissions. There is still more remaining to be done in future.

# Bibliography

- [1] J. Arias. *RTTY: an FSK decoder program for Linux, Version 2.1*, February 2003. [Online]. Available: <http://www.ele.uva.es/~jesus/rtty/>.
- [2] P. Batavia and I. Nourbakhsh. Path planning for the eye personal robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'00)*, volume 1, pages 15–20, Takamatsu, Japan, October 2000.
- [3] M. Bertram, E. Deines, J. Mohring, J. Jegorovs, and H. Hagen. Phonon tracing for auralization and visualization of sound. In *Proceedings of 16th IEEE Visualization Conference (VIS 2005)*, pages 151–158, Minneapolis, MN, USA, October 2005. IEEE Computer Society.
- [4] C. Breazeal and L. Aryananda. Recognition of affective communicative intent in robot-directed speech. *Autonomous Robots*, 12(1):83–104, 2002.
- [5] J. C. Breidenthal, C. D. Edwards, E. Greenberg, G. J. Kazz, and G. K. Noreen. End-to-end information system concept for the mars telecommunications orbiter. In *Aerospace Conference, 2006 IEEE*, March 2006.
- [6] C. Carollo. Sound propagation in 3d environments. In *Game Developers Conference (GDC'02)*, San Jose, CA, USA, March 2002. [Online]. Available: <http://www.gamasutra.com/features/gdcarchive/2002/>.
- [7] S. Chu, S. Narayanan, C. C. J. Kuo, and M. J. Matarić. Where am i? scene recognition for mobile robots using audio features. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, pages 885–888, Toronto, Ontario, Canada, July 2006.
- [8] P. R. Cook. *Real Sound Synthesis for Interactive Applications*. A. K. Peters, Ltd., 2002.
- [9] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed systems (4th ed.): concepts and design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.
- [10] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin ; New York, 1997.

- [11] E. W. Dijkstra. A note on two problems in connexion with graphs. In *Numerische Mathematik*, volume 1, pages 269–271. Mathematisch Centrum, Amsterdam, The Netherlands, 1959.
- [12] R. W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.
- [13] T. Funkhouser, N. Tsingos, and J. M. Jot. Sounds good to me: Computational sound for graphics, virtual reality, and interactive systems. SIGGRAPH 2002 Course Notes #45, July 2002.
- [14] B. Gerkey, R. T. Vaughan, and A. Howard. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics (ICAR'03)*, pages 317–323, Coimbra, Portugal, June 2003.
- [15] L. Girod. *A self-calibrating system of distributed acoustic arrays*. PhD thesis, University of California at Los Angeles, Los Angeles, CA, USA, 2005. Adviser-Deborah L. Estrin.
- [16] S. Haim. *Dictionary English-Persian, Persian-English*. Languages of the World Publications, 2000.
- [17] E. Hecht. *Optics (4th Edition)*. Addison Wesley, August 2001.
- [18] O.E. Holland, C. Melhuish, and S. Hoddell. Chorusing and controlled clustering for minimal mobile agents. In *Proceedings of the Fourth European Conference on Artificial Life*, pages 539–548, Cambridge, MA, USA, 1997. MIT Press.
- [19] J. Huang, T. Supaongprapa, I. Terakura, N. Ohnishi, and N. Sugie. Mobile robot and sound localization. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'97)*, volume 2, pages 683–689, Grenoble, France, September 1997.
- [20] iRobot Corporation. *iRobot® Create: Owners guide*, 2006. [Online]. Available: <http://www.irobot.com/sp.cfm?pageid=294>.
- [21] N. Jakobi. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive Behavior*, 6(2):325–368, 1997.
- [22] V. M. Janik. Whistle matching in wild bottlenose dolphins (*tursiops truncatus*). *Science*, 289(5483):1355–1357, 2000.
- [23] H. W. Jensen. Global illumination using photon maps. In *Proceedings of the Eurographics workshop on rendering techniques*, pages 21–30, London, UK, 1996. Springer-Verlag.
- [24] B. Kapralos, M. Jenkin, and E. Milios. Sonel mapping: acoustic modeling utilizing an acoustic version of photon mapping. In *Proceedings of the 3rd IEEE International Workshop on Haptic, Audio and Visual Environments and Their Applications (HAVE 2004)*, pages 1–6, Ottawa, Ontario, Canada, October 2004.

- [25] B. Kapralos, M. Jenkin, and E. Milios. Acoustical diffraction modeling utilizing the Huygens-Fresnel principle. In *Proceedings of the IEEE International Workshop on Haptic, Audio and Visual Environments and their Applications (HAVE 2005)*, pages 39–44, Ottawa, Ontario, Canada, October 2005.
- [26] P. Karimian, R. T. Vaughan, and S. Brown. Sounds good: Simulation and evaluation of audio communication for multi-robot exploration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'06)*, pages 2711–2716, Beijing, China, October 2006.
- [27] M. Konishi. Sound localization in owls. In E. G. Adelman and B. H. Smith, editors, *Encyclopedia of Neuroscience*, pages 1906–1908. Elsevier, 1999.
- [28] L. Lamport. A new solution of Dijkstra's concurrent programming problem. *Communications of the ACM*, 17(8):453–455, 1974.
- [29] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.
- [30] C. V. Lopes and P. M. Q. Aguiar. Acoustic modems for ubiquitous computing. *IEEE Pervasive Computing*, 2(3):62–71, July 2003.
- [31] E. Martinson and A. Schultz. Robotic discovery of the auditory scene. *IEEE International Conference on Robotics and Automation (ICRA'07)*, pages 435–440, April 2007.
- [32] P. McDowell, B. Bourgeois, P. J. McDowell, S. S. Iyengar, and J. Chen. Relative positioning for team robot navigation. *Autonomous Robots*, 22(2):133–148, 2007.
- [33] P. K. McGregor. Signalling in territorial systems: A context for individual identification, ranging and eavesdropping. *Philosophical Transactions: Biological Sciences, The Evolution and Design of Animal Signalling Systems*, 340(1292):237–244, May 1993.
- [34] C. Melhuish, O. Holland, and S. Hoddell. Convoying: using choring to form travelling groups of minimal agents. *Robotics and Autonomous Systems*, 28:207–216(10), August 1999.
- [35] S. Moorehead, R. Simmons, and W. L. Whittaker. Autonomous exploration using multiple sources of information. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'01)*, volume 3, pages 3098–3103, Seoul, Korea, May 2001.
- [36] E. Østergaard, M. J. Matarić, and G. S. Sukhatme. Distributed multi-robot task allocation for emergency handling. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'01)*, volume 2, pages 821–826, Maui, HI, USA, 2001.

- [37] A. Pentland, R. Fletcher, and A. Hasson. Daknet: rethinking connectivity in developing nations. *Computer*, 37(1):78–83, January 2004.
- [38] G. Ricart and A. K. Agrawala. An optimal algorithm for mutual exclusion in computer networks. *Communications of the ACM*, 24(1):9–17, 1981.
- [39] A. Rowe, C. Rosenberg, and I. Nourbakhsh. A low cost embedded color vision system. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and System (IROS'02)*, volume 1, pages 208–213, Lausanne, Switzerland, September 2002.
- [40] D. Salomon. *Coding for Data and Computer Communications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
- [41] H.-U. Schnitzler, C.F. Moss, and A. Denzinger. From spatial orientation to food acquisition in echolocating bats. *Trends in Ecology and Evolution*, 18:386–394(9), August 2003.
- [42] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. John Wiley & Sons, Inc., New York, NY, USA, 2001.
- [43] W. Stallings. *Data and Computer Communications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.
- [44] M. Stojanovic. Recent advances in high-speed underwater acoustic communications. *IEEE Journal of Oceanic Engineering*, 21(2):125–136, April 1996.
- [45] P. Stone and M. Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, 1999.
- [46] A. Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, Upper Saddle River, NJ, USA, 2002.
- [47] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. MINERVA: a second-generation museum tour-guide robot. In *Proceedings of IEEE International Conference on Robotics and Automation, (ICRA '99)*, volume 3, pages 1999–2005, Detroit, MI, USA, 1999.
- [48] B. Tribelhorn and Z. Dodds. Evaluating the roomba: A low-cost, ubiquitous platform for robotics research and education. *IEEE International Conference on Robotics and Automation ICRA '2007*, pages 1393–1399, April 2007.
- [49] N. Tsingos, T. Funkhouser, A. Ngan, and I. Carlbom. Modeling acoustics in virtual environments using the uniform theory of diffraction. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 545–552, Los Angeles, CA, USA, 2001. ACM Press.

- [50] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, October 1950.
- [51] J. Valin, F. Michaud, J. Rouat, and D. Létourneau. Robust sound source localization using a microphone array on a mobile robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'03)*, volume 2, pages 1228–1233, Las Vegas, Nevada, USA, October 2003.
- [52] R. T. Vaughan and B. Gerkey. On device abstractions for portable, reusable robot code. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robot Systems (IROS'03)*, volume 3, pages 2421–2427, Las Vegas, Nevada, USA, October 2003.
- [53] R. T. Vaughan, K. Støy, G. S. Sukhatme, and M. J. Matarić. Blazing a trail: Insect-inspired resource transportation by a robot team. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS'02)*, pages 111–120, Knoxville, Tennessee, USA, October 2000.
- [54] R. T. Vaughan, K. Støy, G. S. Sukhatme, and M. J. Matarić. Go ahead, make my day: Robot conflict resolution by aggressive competition. In *Proceedings of the 6th International Conference on Simulation of Adaptive Behaviour (SAB)*, pages 491–500, Paris, France, August 2000.
- [55] A. J. Viterbi and J. K. Omura. *Principles of Digital Communication and Coding*. McGraw-Hill, Inc., New York, NY, USA, 1979.
- [56] J. Wang, S. Premvuti, and A. Tabbara. A wireless media access protocol (CSMA/CD-W) for mobile robot based distributed robotic systems. In *Proceedings of the IEEE International Conference on Robotics and Automation, (ICRA 1995)*, volume 3, pages 2561–2566, Nagoya, Japan, May 1995.
- [57] R. H. Wiley and D. G. Richards. Physical constraints on acoustic communication in the atmosphere: Implications for the evolution of animal vocalizations. *Behavioral Ecology and Sociobiology*, 3(1):69–94, 1978.
- [58] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA '97)*, pages 146–151, Monterey, CA, USA, July 1997.
- [59] B. Yamauchi. Frontier-based exploration using multiple robots. In *Proceedings of the Second International Conference on Autonomous Agents (Agents'98)*, pages 47–53, Minneapolis, Minnesota, United States, 1998. ACM Press.
- [60] P. Zebrowski and R. T. Vaughan. Recharging robot teams: A tanker approach. In *Proceedings of the International Conference on Advanced Robotics (ICAR'05)*, pages 803–810, Seattle, Washington, July 2005.