Abstract

BRALY, JOHN CHRISTOPHER. The Development of a Low-Cost and Robust Autonomous Robot Colony Using LEGO® Mindstorms™. (Under the direction of Dr. Edward Grant)

The late twentieth century marked the birth of urban search and rescue robots. The act of rescuing victims from collapsed or damaged buildings is extremely dangerous for the humans involved. After the attacks on the World Trade Center, researchers recognized the need for small robots with limited capabilities to be used in conjunction with more advanced robots for urban search and rescue. This research has developed a low-cost, autonomous robot colony with limited sensor capabilities using the LEGO® Mindstorms™ development platform. The study of this colony will provide insight into the group behavior of a marsupial robot colony used for urban search and rescue.

A microphone sensor was developed to facilitate communication among the robot agents that comprise the colony. The incoming analog signal was amplified using a standard non-inverting operational amplifier configuration. This amplified signal was input into a tone detection circuit. This circuit was designed to provide a digital output to the LEGO® robot if a single tone of a specific frequency was detected. Other frequency tones have no effect on the circuit. Using this sensor, the robots could be controlled with different frequency tones.

The task undertaken by the robots was a shepherding mission. The goal of the sheepdog robot was to herd the sheep robot into a pen located at a fixed location. A helper dog robot was added to assist the sheepdog when needed. The interaction, as well as communication, between the sheepdog and helper dog was studied.

The Development of a Low-Cost and Robust Autonomous Robot Colony Using LEGO® Mindstorms™

by

JOHN CHRISTOPHER BRALY

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

ELECTRICAL AND COMPUTER ENGINEERING

Raleigh

2003

APPROVED BY:

[illegible]
Chair of Advisory Committee

[illegible] [illegible]

[illegible]

Biography

J. CHRIS BRALY was born July 16, 1978 in Falls Church, VA. He received his Bachelor of Science in Electrical Engineering with a minor in Biomedical Engineering from The University of Virginia on May 21, 2000. After spending a year as an Associate Engineer in the Submarine Engineering Department at Newport News Shipbuilding Company, he enrolled at North Carolina State University. He is currently pursuing his Master of Science in Electrical Engineering, conducting research in autonomous robotics and smart-sensor development at the North Carolina State University Center for Robotics and Intelligent Machines (CRIM).

Acknowledgements

I would first like to thank my advisor, Dr. Eddie Grant, for allowing me to pursue this crazy idea of building something useful out of LEGOs®. Without his guidance and patience, this project could have never been completed. He allowed me to join the CRIM and pursue my interest in robotics without ever having worked with me. For that, I thank him dearly. I would also like to acknowledge my advisory committee, Dr. John F. Muth, Dr. Mark White, and Dr. H. Troy Nagle for their support throughout this process.

I also thank all of the members of the Center for Robotics and Intelligent Machines for their friendship and assistance. I consider myself lucky for being able to have worked with so many intelligent people. I would especially like to thank Kyle Luthy, Tim Slusser, and Leonardo Mattos for their unwavering support and the aid they have provided to me.

Finally, I would like to thank Holly for her loving support and endless patience. She has supported me selflessly throughout this entire process. Without her, I'm not sure I would have ever finished.

Table of Contents

Table of Contents (continued)

List of Figures

List of Figures (continued)

List of Tables

Chapter 1 – Introduction

The study of cooperative animal societies helps to provide insight into group dynamics and behavior. By studying the foraging behaviors of fish, the mob behavior of whip-tail wallabies, or the organization of primate colonies, researchers gain an in-depth knowledge of animal behavior. Ants are one of the most studied biological systems. Their social organization, methods of communication, and decision-making are some of the most sophisticated of any biological society [3]. Studying different biological societies serves as a model for mobile robot societies. In fact, cooperative multi-agent robot society research was motivated by biological systems [31]. Many researchers have attempted to create robot societies that mimic animal societies.

Section 1.1 – Project Motivation

Mobile robotics research changed directions in the mid-1990s after two disasters stunned the world. The Kobe earthquake in January of 1995 and the Oklahoma City bombing in April of 1995 resulted in massive devastation and a tremendous loss of life. Robotics researchers recognized that rescuers needed help when searching for victims trapped in the rubble of collapsed buildings. The act of searching for victims is not only difficult, but also extremely dangerous. Rescue workers face the possibility of gas leaks, explosions, and further collapse. The use of mobile robots for urban search and rescue missions help reduce the risks faced by workers at the outset of the operation.

One of the major problems faced by the robots at a search and rescue site is the overall size of the search robot. Most robots that have the ability to perform complex tasks are rather large due to the extensive onboard sensor suite, the complex communication system, and numerous batteries. Unfortunately, their size prevents them from searching much of the search area. For this reason, many are investigating the possibility of using small "micro-rovers" in tandem with the larger robots for search and rescue missions. The larger robots will remain relatively unchanged except for having the smaller robots gather the sensory information. The larger robot also transports the micro-rovers to the rescue site. This cooperative group of mobile robots has been termed "marsupial robots" because it is similar to the way a kangaroo mother carries her young.

This research focuses on developing a marsupial-like robot colony using a low-cost, robust robot platform. Instead of concentrating on creating the larger robot, several micro-rovers are built to investigate ways of communicating with the team of robots. This thesis presents background information regarding the LEGO® Mindstorms™ platform used to develop the mobile robots. A description of the experiments performed on the colony and the results are then presented. Finally, potential future research areas are suggested.

Section 1.2 – Project Goals

The objectives of this thesis are to

1. Provide a complete description of the LEGO® Mindstorms™ platform used to develop the robot colony.
2. Investigate various communication schemes that can be used to provide instruction to the robot colony.
3. Describe the design and construction of the robot colony, including additional hardware and software used.
4. Establish a communication link between colony members and examine the interaction among team members during experimentation.

Section 1.3 – Thesis Outline

In Chapter 2, an extensive literature review is presented. Cooperative, multi-agent robot societies are discussed, as well as biologically inspired research, LEGO® Mindstorms™ research, and robots used for urban search and rescue. A complete overview of the LEGO® Mindstorms™ robot development platform is presented in Chapter 3. In Chapter 4, different methods of communicating with the members of the robot colony are discussed. Chapter 5 contains a description of the robot colony developed. In Chapter 6, the experiments performed with the robot colony and the results of these experiments are then presented. Chapter 7 contains concluding remarks and suggestions for future research. Finally, an extensive list of references is included in Chapter 8, and detailed descriptions of the hardware and software used are provided in the Appendix.

Chapter 2 - Literature Review

In 1948 and 1949, while investigating his theories about the nervous system, W. Grey Walter built two autonomous robots to help him understand the operation of animal brains. Each *Machina speculatrix* robot had a vacuum tube that simulated two interconnected neurons. These amplifier circuits connected two sensors, a photocell and a touch sensor, to the two motors [29] [50]. When the touch sensor was closed, one of the amplifier circuits oscillated, and the robot changed direction [29] [50] [51]. The two "tortoises" exhibited some intriguing behaviors, including the tendency "to explore the environment rather than to wait passively for something to happen" [50, pg. 126]. Other behaviors included the attraction to moderate light, the repulsion when exposed to bright light, mutual recognition (the robots would "flock" together when there was no outside stimulus present), prioritizing tasks (i.e. avoiding an obstacle instead of moving toward a light), and the inclination to seek environments with favorable conditions (i.e. moderate light) [3] [50].

Walter then began to study whether or not his robots could learn. He attached a circuit, the Conditioned Reflex Analogue (CORA), to his original robots creating *Machina docilis*. CORA provided a link between either of the robot's sensors (light or contact) with a sound stimulus. Using this circuit, Walter trained his robots to come by first blowing a whistle and then showing the robot a light. He then trained it move away from a loud sound by blowing the whistle and then triggering the contact reflex [29] [50] [51]. These experiments with *Machina speculatrix* and *Machina docilis* provided the foundation for the study of robot-robot and human-robot interaction.

Prior to the 1980s, robotics researchers focused their efforts primarily on issues dealing with single-robot systems. However, researchers began to shift their interests to multiple mobile robot systems in the late 1980s [31]. In 1988, Fukuda and Nakagawa [17] proposed a robotic system that would be able to autonomously reconfigure its shape and software given a specific task. This system is comprised of individual autonomous “cells” that have a single function and a small amount of intelligence, much like the individual cells of the body. In the body, each cell works alone but can cooperate with other cells in the group to perform a specific task. Gerardo Beni [9] discusses the idea of creating an autonomous robotic system in which the individual robots work together to accomplish an explicit task. These simplistic robots do not have a central controller nor do they share memory, but when working together, they can accomplish complex tasks.

Throughout the 1990s, several research themes emerged for multi-agent systems. Some of the most common themes include foraging for items, performing tasks (assembly or disassembly), maintaining formations while moving throughout an area, surveillance, transporting objects, path-planning, collision avoidance, and robot-soccer [3] [31]. Rybski and others [46] have developed a multi-agent system that can be used for reconnaissance and surveillance. The system is comprised of a few large robots and several small “scouts” that serve as the eyes and ears of the operation. The large robots function as supervisors, collect data from the scouts, and coordinate the behaviors of the scouts. Researchers in Brazil [42] have developed two robots that perform a box-carrying task without having to explicitly communicate (no data flow between robots) with each other. The determination of the robot’s role in the task is made by relying solely on local sensor data. The research has shown that this method of communication is just as effective as explicit communication. Other researchers have investigated dynamic role assignment of multi-agent teams for various tasks. Emery et al. [15] present several techniques for collaboration and coordination of a team of robots that play soccer. This approach allows for the reduction of interference among team members, as well as determining their role (i.e. offense or defense) based on their location on the field. This eliminates the need for specialized players and allows all teammates to help each other if needed. Chaimowicz and others [12] also use dynamic role assignment for a group of robots that cooperatively search and retrieve objects scattered throughout the environment. This allows for adaptations to be made if an unexpected change in the environment occurs and results in improved overall efficiency. Lynne Parker [41] has investigated how cooperative team performance is affected by robot team member awareness when performing a puck-moving mission. Her research indicates that the awareness on team performance is a function of several factors, including team size, how well the effects of actions are sensed, the amount of work available for each team member, and the cost of executing redundant tasks. She has also developed ALLIANCE [40], a software architecture that "facilitates fault tolerant, reliable, and adaptive cooperation among small- to medium-sized teams of heterogeneous mobile robots, performing (in dynamic environments) missions composed of independent tasks that can have ordering dependencies" [40, pg 221]. The feasibility of this control architecture was demonstrated by performing a simulated hazardous waste cleanup with a team of mobile robots in the laboratory. The research indicates that ALLIANCE improves the team robustness by continually monitoring the sensors of a single robot and then adapting the robot's response based on environmental changes that have occurred and the actions of its teammates.

Many researchers [3] [4] [13] [21] [31] [34] [41] acknowledge the benefits of using multiple-robot systems instead of single-robot systems. Such systems are more cost effective since many smaller robots can be built for the same cost as one large robot. Multi-agent systems are also more redundant, and therefore more fault-tolerant, than a single-robot system. If one robot fails, the others can still complete the task without any major problems. Also, with a team of robots, the sensor information of one robot can be shared with the others, allowing for more informed decisions to be made by individual team members. Finally, there is an improved system performance when several robots are used, allowing for divide and conquer techniques to be employed. With multi-agent systems, several "tasks can be completed considerably more efficiently overall for a wide range of tasks and environments using groups of robots working together" [3, pg 359].

There are, however, several problems that can arise when using multiple robots [3] [34]. Interference can occur with too many robots and not enough work. Problems can also occur when numerous robots are being used in confined areas. Team members can unintentionally interfere with each other in this situation. It is also difficult for a robot to know when it or another team member is being unproductive. Communication between team members is also a problem area with multi-agent systems. Specialty hardware, extra processing, and more power are required for successful communication. Also, noisy channels and signal strength can affect system performance. Noisy channels can degrade the signal to the point where the message is misunderstood or not even received at all. Signal strength is especially important when sending distress signals. Too weak of a signal might result in it not being heard, while too large of a signal might cause the entire colony to provide assistance. For successful operation, a robot must know what its team members are doing. Without communication between robot team members, as well as an appropriate sensor suite, robots may compete against each other and reduce overall system performance. Nevertheless, these problems can be overcome with careful planning by the system designer.

The multi-agent robot systems research that emerged at the end of the 1980s and continued through the mid-1990s was inspired by biological systems and “the collective intelligence demonstrated by social insects” [31, pg 7]. An ant colony provides a good example. The colony is comprised of hundreds of ants, each with a specific duty. Yet a group of these ants can work together to move a dead earthworm from one place to another [31]. Russell et al. [45] created a robotic system that replicates the way ants mark the trail between their nest and a food source (both ingoing and outgoing). Like ants, the robots in this system perform navigation tasks by leaving and detecting trails of volatile chemicals. The researchers suggest that this type of system could be used by a group of cleaning robots to mark an area of the floor that has already been cleaned. At the University of Strathclyde in Glasgow, Scotland, Lambert [28] and Russell designed and built autonomous robots to imitate the behavior of a sheepdog and a sheep. The primary goal of their project was to have the sheepdog herd the sheep into a pen located at an arbitrary location while maintaining the natural behaviors of both the sheep and sheepdog.

The dog was designed to round up any sheep in the “field” and corral them into a pen without any external assistance from a “shepherd” or another dog. It was also designed to identify the pen and sheep and to calculate the distance and direction to each object. The sheep was designed to identify the dog and other sheep (if any) and also establish the distance and direction to each object. In order to accomplish the goal of rounding up the sheep while maintaining natural behavior, the dog was allowed to travel at a faster speed than the sheep. Both robots were designed to detect and avoid collisions with objects in the field. In addition, both the sheep and sheepdog were to be identical, with the exception of the software used to control the two robots. Finally, they were to operate over as large of an area as possible.

The behavioral model of the sheep consisted of three parts: graze, flock, and flee. First, the sheep would move within the field in a random way, thus appearing to graze. If the sheep detected the dog at a distance, it flocks towards other sheep if they are not close to the dog. If the dog moved too close to the sheep, it would panic and move away at full speed. The behavioral model of the dog is made up of four parts: finding the pen, locating the sheep, herding the sheep towards the pen, and funneling it into the pen. First, the dog would search for the pen. Once the pen was located, the dog would search for the sheep. When it was located, the dog moves towards the sheep. It would move around the sheep to a position where the sheep is between itself and the pen. This caused the sheep to move away from the dog and towards the pen. Once the dog had moved the sheep near the pen, it would the force the sheep into the pen.

Combining the biologically inspired multi-robot systems research of the mid-1990s and the need to help rescuers during urban search and rescue missions resulted in the development of marsupial robots. "Urban search and rescue (USAR) focuses on locating and extracting people trapped in collapsed or damaged structures. Rescuers are under extreme time pressure; after 48 hours, victim mortality drastically increases owing to exposure and lack of food, water, and medical treatment" [35, pg 14]. Marsupial robots are simply a group of robots in which there is a large "mother" robot that carries at least one smaller "daughter" robot, similar to the way a kangaroo mother carries her young. The mother robot provides for the daughter robot in many ways. It protects the daughter robot by transporting it from one location to another. It also provides battery power to the daughter robot, by either a tether or a recharging station. It can also help the smaller microrover during its mission by acting as a leader or manager. The mother can collect the sensor data from the daughter robot and use it to make decisions that pertain to the task at hand. It can also serve as a communication relay station between the microrover and the human operator. Finally, the mother robot can rescue the daughter robot if it gets into trouble [34] [35] [37].

Traditionally, only highly trained individuals and dogs were used in USAR missions. However, the idea of using robots to help rescue survivors came about in 1995. On January 17, 1995, an earthquake centered near the area of Kobe and Osaka, Japan registered 7.2 on the Richter scale. This earthquake killed 5,100 people, injured 26,800 others, and caused approximately US$100 billion damage. Soon after the earthquake, the Tokyo Fire Department's Fire Science Laboratories began to develop a line of robots that could be used in urban search and rescue applications. On April 19, 1995, 168 people were killed and more than 500 people were injured in the bombing of the Alfred P. Murrah Federal Building in Oklahoma City, OK. John Blitch, a Masters of Science student of Robin Murphy at the Colorado School of Mines, served as a rescue worker. When he returned, he and Murphy, an expert researcher in the field of USAR, decided to focus their research efforts on developing robots that could be used to help urban search and rescue missions [14]. Others [26] also began researching the use of robots for urban search and rescue in the late 1990s. A distributed team of mobile robots without a central supervisor was developed to search for an object and then move the object to a determined area. Using the implemented algorithm, each robot searches for the object concurrently, signals to other team members once the object has been found, and then cooperatively moves the object once it has been found.

One of the major problems with USAR research is the lack of realistic field studies. Researchers at the University of South Florida have teamed together with the local fire department [10] [11] [14] [36] to test their robots in a staged search and rescue environment. However, these staged environments can only identify potential problems and validate hypothesized situations that may be encountered during an urban search and rescue mission. Unstaged USAR environments are needed in order to determine if the current research is moving in the right direction. After the attacks on the World Trade Center in New York City on September 11, 2001, robots were used in an actual search and rescue mission for the first time. Soon after the attacks, Blitch organized a group of robot researchers and manufacturers to help with the search and rescue mission. The group was at the site for about four weeks, and found ten victims (more than 2% of all victims discovered). This experience helped to validate that the USAR research is moving in the right direction [10] [14].

There are several benefits of using robots, especially marsupial robot teams, for urban search and rescue efforts. Most importantly, rescue situations are extremely dangerous for those involved. There is a high potential for gas leaks, explosions, and further collapse. Robots can send environmental conditions of the search area, such as the presence of harmful gasses and seismic data, to rescue workers. Using robots can reduce the risks faced by humans at the outset of the rescue mission. Also, rescue workers must be very deliberate while searching for victims in order to remain safe. In some situations, workers might be required to leave an unstable site until it becomes safe to return. Robots can perform searches at a faster pace without feeling the fatigue and stress a human would feel [35] [36].

Marsupial robots are particularly useful in search and rescue situations. A marsupial robot team is specifically designed for very small robots to be carried to a site by a larger robot. The size of the daughter robot allows it to move through small voids that larger robots and humans cannot fit into. Also, the daughter robot does not have to worry about sending sensory information back to the rescue workers. This is performed by the mother, resulting in the battery power of the daughter being used primarily for the search. Murphy's experiments show that a marsupial team can perform better than individual robots can when it comes to reaching destinations and arriving at these destinations quicker over longer distances. Murphy is also investigating using robots as part of the marsupial team that can change their shape depending on the environment, as well as the human-marsupial team roles during a USAR mission [35] [37].

Lambert and Russell's experiment was replicated in 2002 by graduate students at North Carolina State University using LEGO® Mindstorms™ [8]. Using primarily LEGO® Mindstorms™ parts, a sheep and sheepdog were built. The sheepdog was able to herd the sheep into the pen without the aid of any additional processing power or human intervention. This project showed that the LEGO® Mindstorms™ were a feasible platform to develop autonomous mobile robots. Many others have used LEGO® Mindstorms™ effectively in their research. Michael Gasperi has recreated Grey Walter's tortoises using LEGO® Mindstorms™ [19]. Kumar [27] uses LEGO® Mindstorms™ as an integral part of his undergraduate artificial intelligence (AI) course. The robot development kit gives students the opportunity to focus their attention on solving AI problems, and spending time building their robot. The kit is also relatively inexpensive, averaging about $200, and can be found at local toy stores or online. This allows for most of the students to purchase their own set.

Researchers in Japan [52] have performed experiments that evolve the morphology and controller of a LEGO® robot. Iversen et al. [25] present findings on the automatic verification of real-time control programs running on LEGO® Mindstorms™ system. They demonstrate their verification techniques by building a robot that can sort LEGO® bricks by color. These researchers show that LEGO® Mindstorms™ can be used as an effective platform to develop fairly complex, but inexpensive, robots.

Chapter 3 – The LEGO® Mindstorms™ Platform

Previous research was conducted to determine the feasibility of using the LEGO® Mindstorms™ Robotics Invention System 2.0 as a platform for autonomous mobile robot development [8]**.** During this project, two mobile robots were created to mimic the behavior of a sheep and a sheepdog. The results of this research indicated that the LEGO® Mindstorms™ could serve as a capable platform to use for the design and construction of mobile robots without having to make any modifications to the original system. The capabilities and limitations of the LEGO® Mindstorms™ platform will be discussed in the following section.

Section 3.1 – RCX and Programming

At the heart of the LEGO® Mindstorms™ Robotics Invention System is the robotic command explorer, or RCX (Figure 3.1). Each RCX is powered by a total of six AA batteries. The major components of the RCX are the Hitachi H8 microcontroller with 32 KB of RAM, three output ports used to power the motors, three input sensor ports, and a liquid-crystal-display (LCD) screen. It also contains an infrared transmitter and receiver, which can be used to communicate with the base station plugged into a computer or another RCX block [44] [49]. A Stanford University graduate student, Kekoa Proudfoot, was the first person to disassemble the RCX and fully document the internal components [49]. A picture of Kekoa's disassembled RCX is shown in Figure 3.2.

Figure 3.1 LEGO® Mindstorms™ RCX

The lack of sensor input and motor output ports are the most serious limitations of the RCX. Only three sensors can be used with the RCX at one time. In order to build a functional autonomous robot, more than three sensors are often needed. However, the lack of motor ports does not hinder development as much. For most applications, the three motor output ports will suffice. For most applications, more than three motors will result in too much weight, and the robot will not be able to move very well. To allow for more than three sensors to be connected to the RCX, several sensors can be connected to the same port by multiplexing them together. By changing the sensor type from light to touch and then back

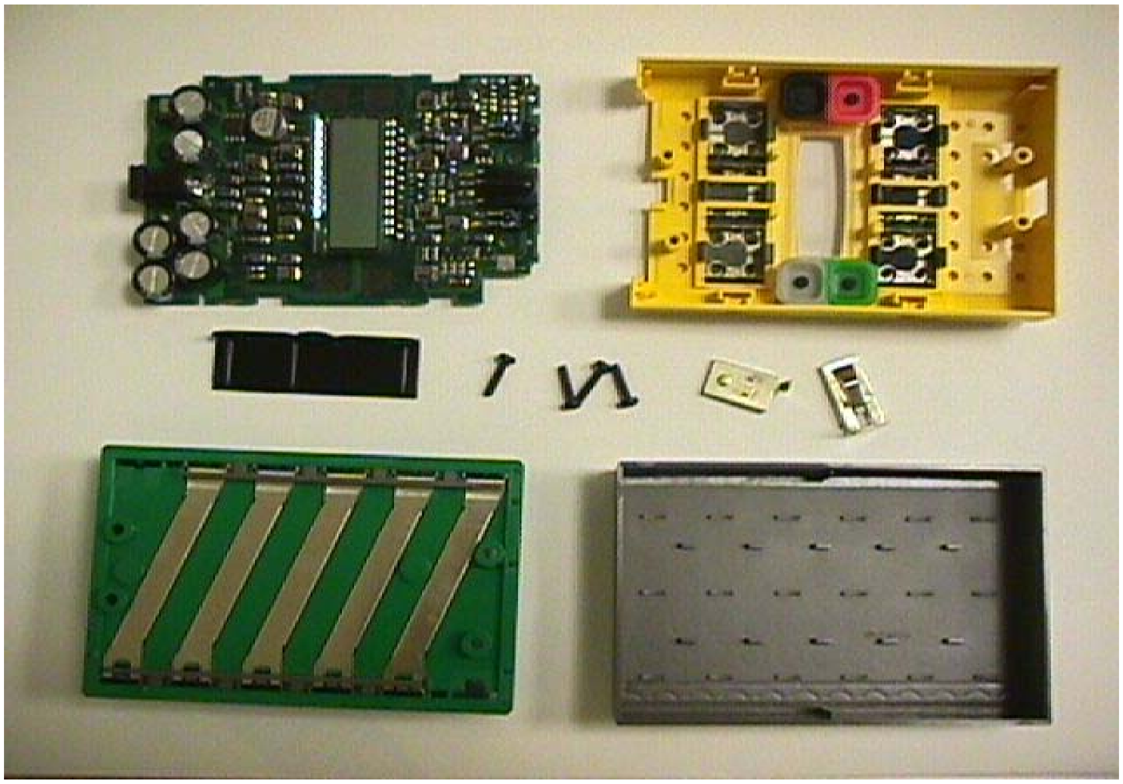


Figure 3.2 Disassembled RCX [44]

to light, the RCX can cycle through the sensors. This action is similar to cycling though outputs of a standard multiplexer using a clock. As the RCX changes from one sensor to another sensor, it does not only just switch the signal to the sensor, but it also switches the power to the sensor off. This results in a conservation of battery power, since only one of the multiplexed sensors will be on at a time [1]. Other similar expansion techniques for more input and output ports have been found in online literature [20].

Another drawback is the limited amount of onboard memory (32 KB), of which about half is taken up by the firmware. The firmware is used to interpret the downloaded programs and execute the processor machine code [49]. A total of five programs can be loaded onto the RCX at one time. However, these programs can be no larger than 6 KB and must be limited to the use of up to 32 variables [27]. This limited amount of memory for a program is not a severe drawback since typical RCX programs tend to be only hundreds of bytes long [5]. This shortcoming does not prevent the development of autonomous mobile robots.

The visual programming environment for the LEGO® Mindstorms™ Robotics Invention System 2.0 poses a slight problem when developing robots using this platform. In order to write a program, blocks are arranged using the supplied PC software [49]. This method of programming may be perfect for the audience the product was designed for (children ages 12 and up). It is, however, difficult to implement advanced algorithms using this technique. The supplied programming environment does not support recursion and nested control constructs [27]. Fortunately, alternative languages and cross compliers have been developed for more advanced programming. These options include Not Quite C (NQC) [5] [6], pbForth (a variant of Forth) [23], leJOS (JAVA based) [47], Visual Basic [22], Visual C++ [7], and Ada [16].

For this project, all code was written in the NQC programming language (Beta Release – 2.5 a5) due to the familiarity of the C language by the researcher. NQC, developed by Dave Baum, allows the user to write programs using an ordinary text editor in a known dialect. The compiler then converts the NQC code into a language that can be downloaded and understood by the RCX firmware. Using this technique to write programs, rather than the LEGO® visual programming technique, gives the user more control over the RCX hardware [49].

NQC supports traditional C function definitions and commands (such as *for* and *while* loops) with additional commands designed specifically for the LEGO® Mindstorms™ kit. These functions include commands to define sensor ports, to define motor directions and power levels, to play various sounds, and to send and receive simple messages and serial data using infrared [6]. The major drawback to using infrared communication between two different RCX blocks is that the message cannot be received from all directions. In order for an RCX block to receive a message, its receiver must be pointing directly at the transmitter of the other RCX block that is sending the message. There is also extremely large power consumption when sending a message, resulting in a battery lifespan much shorter than normal.

One aspect of NQC that is different from traditional C is the task code block. Since the RCX supports multi-tasking, “an NQC task directly corresponds to an RCX task” [6, pg. 50]. The RCX can support up to 10 tasks running simultaneously, including *main*, which is a task in NQC rather than a function as it is in the C programming language [6]. Tasks have a distinct advantage over traditional functions because, once started, they run continuously until a stop command is issued. Functions only execute once when called, making tasks more convenient for procedures that must run constantly.

Section 3.2 – Motors and Gearing

Next to the RCX, the motors, gears, and the specialty LEGO® bricks are the most important components of the LEGO® Mindstorms™ kit. Without them, a robot could not be built. The Robotics Invention System contains numerous gears of varying sizes that allow for many different ways to "gear up" (increase rotational speed) or "gear down" (decrease rotational speed) the motor. It also contains several unique LEGO® blocks that can be used with the motors and gears to create an assortment of moving parts.

The two motors (one is shown in Figure 3.3) included effectively provide the necessary power needed to develop a functional LEGO® robot. Unfortunately, the LEGO® Mindstorms™ kit only comes with two motors, which makes it difficult to create more complex robots. Individual motors, however, can be purchased if they are required [43]. The motors can be driven forward or backward in the *on* mode, not move at all in the *off* mode, or be allowed to "coast" in the *floating* mode. The *floating* mode is synonymous to neutral for an automobile. Each motor can operate at one of eight power levels. The RCX uses pulse-width modulation to create the intermediate power levels using a digital signal. Varying the widths of the pulses creates the intermediate power levels. The RCX sends pulses every 8ms with the width of the lowest power level being 1ms and the width of the highest power level being 8ms. The motor is constructed such that an internal flywheel is used to keep the motor spinning until the next pulse is supplied [5].

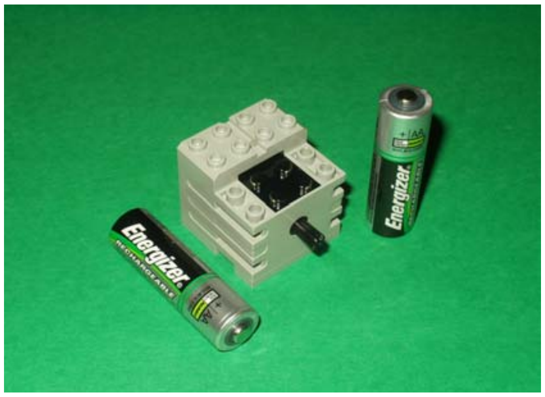


Figure 3.3 LEGO® Mindstorms™ motor

In 2002, LEGO® upgraded the motors supplied with the Robotics Invention System. Due to the fact that some of the kits were purchased prior to the change, the new (LEGO® part number 43362) and old (LEGO® part number 71427) motors will be discussed. The two motors are almost identical, but there are slight differences. Most noticeable is the weight of the two motors. The pre-2002 motor weighs about 42 grams as compared to about 28 grams of the new motor. This decrease in weight will help when larger, more complex robots are being developed. Another less noticeable change is the addition of a PTC resistor that protects the motor from over-current by increasing the resistance if the temperature rises, thus limiting current. Based on Phillipe Hurbain's experiments, the old motor performs slightly better, but the weight of the new motor can justify its use [24].

Section 3.3 – LEGO® Sensors

The following section describes the sensors included in the LEGO® Mindstorms™ Robotics Inventions System and the Ultimate Accessory Set. The usefulness of the sensors will be discussed along with ways to improve the effectiveness of the sensors. In addition to the extra sensors, the Ultimate Accessory Set also contains a remote control, a LEGO® lamp (Figure 3.4), and extra LEGO® building materials. The LEGO® lamp does not produce much light and is not very useful in a lit room. It might, however, be useful if the robot is operating in a dark environment. The remote control can be used to manually drive the motors, send a message to the RCX, or select which of the five programs to run. It is particularly useful since the robot can be started from a distance and prevents the user from getting in the way. The infrared receiver on the RCX must be pointed at the remote control in order for the remote to work.

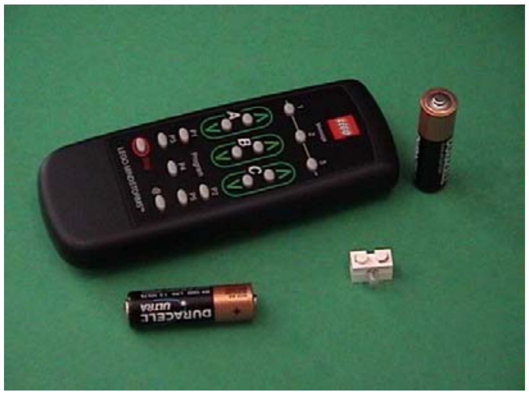


Figure 3.4 LEGO® remote and lamp from Ultimate Accessory Set

Section 3.3.1 – Light Sensor

The two major components of the LEGO® light sensor (Figure 3.5) are a red light emitting diode (LED) and a phototransistor. The phototransistor responds to the amount of incoming light, whether it is the amount of light being produced by a source or the amount of reflected light produced by the red LED [5]. Because of these two components, it is one of the most versatile LEGO® sensors. For example, it can be used on a robot to measure the amount of ambient light in a room and perform various tasks accordingly. It can also be used to determine the distance from a particular light source (by measuring the amount of incoming light) or the distance to an object (by measuring the amount of reflected red light). However, measuring the distance to an object using the light sensor is not very effective unless the object is very close to the sensor. Finally, it can be used as a sensor that detects different colors by measuring the amount of reflected red light produced by the LED, since different colors reflect a different amount of red light.

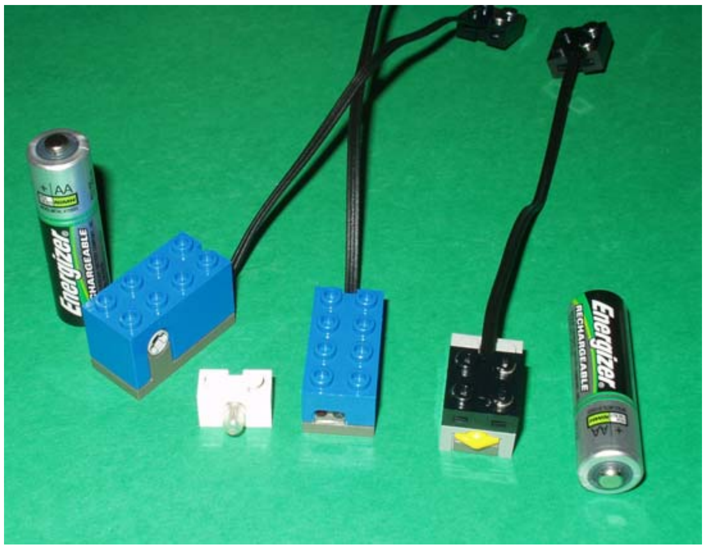


Figure 3.5 LEGO® Mindstorms™ sensors from left to right: rotation sensor, lamp, light sensor, and touch sensor

If the red LED is not needed for the desired application, it can be "eliminated" to increase the sensitivity and performance of the light sensor. The light sensor can be disassembled and the red LED can be physically removed, resulting in improved performance [2] [20]. This method is not recommended since permanent damage to the structure of the sensor will occur, and possible damage to the sensor circuitry could result, rendering it useless. Another method that does not require any disassembly of the sensor has been performed [24]. Inserting a tiny piece of black plastic between the red LED and the phototransistor effectively blocks the red light and prevents it from being detected by the phototransistor. A 2x1 LEGO® Technic block can then be placed in front of the light sensor to secure the plastic. A small piece of electrical tape might be required to attach the plastic. Pictures illustrating this procedure are shown below in Figure 3.6.

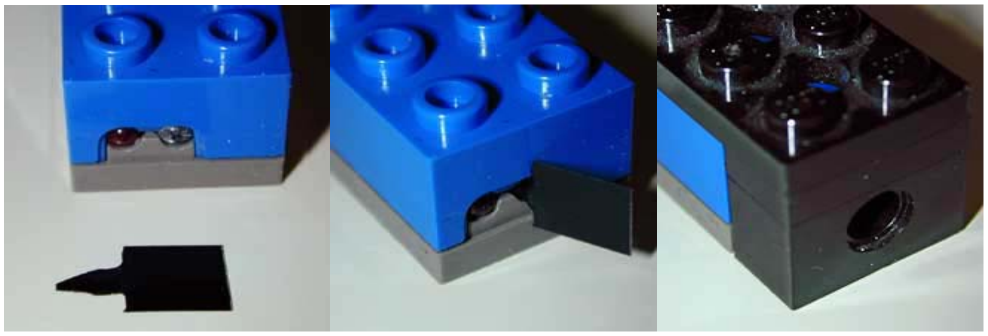


Figure 3.6 Improving light sensor sensitivity. Use a small piece of black plastic to block the red LED emitter from the light detector (far left picture). Insert the plastic in between the red LED on the left and the light detector on the right (center picture). Put a 2x1 LEGO® Technic brick in front of the light sensor to secure the "filter" and allow the light detector to sense light through the hole (far right picture) [24].

With this configuration, the light sensor is not as susceptible to the effects of ambient light. It also makes the sensor extremely directional by restraining the detection of incoming light to the light that enters through the hole in the Technic brick. Tests have shown that when using this design, infrared light can be detected at distances up to twelve feet with the room lights on. This is quite impressive when compared to the results achieved when using standard cadmium sulfide (CdS) photoresistors.

One important factor to remember when using the light sensor is that the robot will need to be calibrated each time it is used. Each environment has a different amount of ambient light. Even if a Technic brick is placed in front of the sensor as shown above, calibration might be required in order to work properly. Also, the light sensor requires power to operate. This means that as the batteries become weaker, the power supplied to the sensor decreases. The decrease in power will also result in the need for recalibration [5].

Section 3.3.2 – Rotation Sensor

The LEGO® rotation sensor (Figure 3.5) is not included in the Robotics Invention System. It comes in the Ultimate Accessory kit, or it can be purchased separately from LEGO®. This sensor measures the amount of rotation relative to a base position. The base position is defined, by default, as the position of the sensor when the program is started. However, a special command can be issued during operation to change the base position from the default position to the current position of the sensor [5].

The base position of the sensor is given the value of zero. After one complete revolution, the value of the sensor is 16, which corresponds to 22.5° per count. To increase the accuracy of the measurement, gear reduction can be used [5]. The sensor also counts down. For example, if the sensor completes one complete revolution by rotating in one direction (i.e. clockwise), the value will be 16. If it then completes one revolution in the opposite direction (anti-clockwise), then the value of the sensor will be zero. If it then makes another revolution in the anti-clockwise direction, the value of the sensor will be -16. The rotation sensor ranges from -32767 to +32767 [19].

The major drawback of the rotation sensor is that it performs very poor when the speed of rotation is slow. Experiments have shown that the sensor skips values when it is rotating slowly, resulting in an error of 22.5° per skipped value. In some applications, this might not be a problem. Yet for most applications, this large error would result in failure. Philippe Hurbain has disassembled his rotation sensor, and after performing an in depth analysis to determine exactly how it works, has determined that if a capacitor is added to the circuit, the sensor no longer skips values at low speeds [24]. An easier way to remedy this problem is to put the sensor on the shaft of the motor before any gear reduction is added. By doing this, the motor speed can be set high enough so the sensor does not skip any values. Gear reduction then can be added to increase the accuracy of the sensor. This method does not risk damage to the sensor by attempting to take it apart.

Section 3.3.3 – Touch Sensor

The LEGO® touch sensor (Figure 3.5) is the simplest of all the LEGO® sensors. When the button on the front of the sensor is not pressed, the circuit is open and the output value is LOW. The output value changes to HIGH when the button is pressed and the circuit is closed. These sensors can be connected to the same input port in a variety of ways to create either an AND configuration or an OR configuration. If connected in the AND configuration, two LEGO® lamps can be used to determine which touch sensor has been pressed [20]. When the sensors are connected in the OR configuration, the RCX cannot distinguish which sensor is being pressed [5]. The touch sensors can be used to construct "bumpers" that will result in the robot performing a certain task, such as reversing its direction, if it runs into an obstacle.

Section 3.3.4 – Other LEGO® Sensors

Besides the sensors that are included with the Robotics Invention System and the Ultimate Accessory kit, there are numerous sensors available that interface with the RCX. Several companies sell sensors that can be used with the RCX. PITSCO (the LEGO® educational division) sells all the sensors previously discussed, as well as a temperature sensor [43]. DCP Microdevelopments also makes a variety of sensors that will interface with the RCX, including a humidity sensor, a pH sensor, a pressure sensor, and a sound level sensor [30].

There are also many LEGO® enthusiasts that have built "homebrew" sensors using LEGO® blocks, standard electronics, and a bit of ingenuity. Michael Gasperi has an extensive list of sensors built by hobbyists, including an angle sensor, a rotation sensor, a motor speed/torque sensor, a differential light sensor, a pyroelectric motion sensor, and an ultrasonic range sensor [20]. Philippe Hurbain has developed several sensors including a color sensor and a wire guidance sensor that allows a robot to follow a wire using an AC current flow to generate a magnetic field [24].

Chapter 4 – Robot Communication

One if the most challenging aspects when developing a cooperative robot team is establishing a communication link between team members. The major problem is that a communication system is not reliable at all times. For example, noisy channels can cause a robot to miss a message or interpret it incorrectly. Also, in some environments, electronic countermeasures may be used to block communications between team members. Malfunctioning receivers can also cause a robot to miss a message. Other important factors regarding a communication system are range, signal strength, and type of communication (direct communication or communicating through observation) [3]. In this chapter, the two primary techniques of establishing a communication link between the LEGO® robots that were investigated are discussed.

Section 4.1 – Infrared Light

Two methods of infrared communication between robots were explored. The first method consisted of using the infrared transmitter and receiver on the RCX. The second method was to pulse infrared light at different frequencies, each corresponding to different messages.

Section 4.1.1 – RCX Infrared Transmitter/Receiver

The first communication scheme that was investigated was using the RCX infrared transmitter and receiver to send messages to the robots. The RCX has the ability to receive simple 8-bit messages ranging from 0 to 255. It continually monitors for incoming messages, remembering the most recently received message. If no message has been received, the default value stored in memory is zero [5] [6].

The basic idea behind this method of communication is each numbered message corresponds to a different instruction. With 255 possible messages (since the default value for no received message is zero), many different instructions can be given to the robots. Another benefit to using this communication scheme is that no additional hardware or software is required. The RCX has the ability to send and receive infrared messages by using predefined NQC functions.

Unfortunately, there are major drawbacks to using this communication scheme. First of all, in order to receive a message, the robot must point the receiver directly at the transmitter. The receiver is not omnidirectional. If the transmitter is at a fixed orientation, this is not that big of a problem. The robot could easily orient itself so it could receive a message. It would either have to know when it is being given an instruction, or check at regular intervals to see if a message is being sent. If the transmitter were allowed to move, it would be extremely difficult to align both it and receiver in order to detect a message. Another problem is the amount of power required to send an infrared message. If an RCX is used to transmit the message, the batteries are drained very rapidly, even when using the low-power transmission setting. On the other hand, if the infrared tower used to download programs onto the RCX is employed to send messages, the power consumption is not an issue. Using the infrared tower, however, does not allow for communication among team members, which is desired. Due to this, as well as the high power expenditure that occurs and the fact that messages cannot be received at any orientation, the built-in RCX infrared transmitter was not used to communicate between robots.

Section 4.1.2 – Pulsing Infrared Light

By transmitting infrared light at different frequencies, different messages can be sent by changing the frequency of the transmitted signal. When the robot detects a signal of a specific frequency, the instruction associated with that frequency could then be executed. Detection of the infrared signal can be accomplished by building an array with several detectors designed to sense infrared light of a specific wavelength. Each array element would detect a different wavelength, and thus correspond to a different instruction. A block diagram of this detection scheme is shown in Figure 4.1. In order to tell which array element was sensing the signal, the output of each element would be sent to a multiplexer before being interfaced with the RCX. If a multiplexer is used, a clock is required to cycle through each input. Either a 555-timer circuit or RCX sensor input will provide an adequate clock for the multiplexer. In order to use the RCX sensor input as a clock, the sensor type must be changed from *touch* to *light* and back to *touch* over a pre-determined duration. The only problem with using this configuration is that two sensor inputs would be used, leaving only one for the remaining sensors. For this reason, the more desirable configuration to use is the 555-timer circuit.

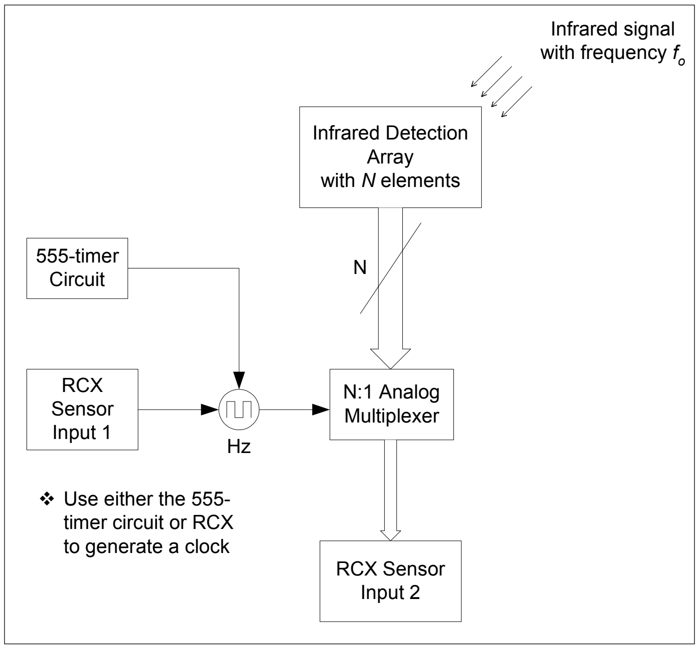

Infrared signal with frequency f_o
Infrared Detection Array with N elements
N
555-timer Circuit
RCX Sensor Input 1
Hz
N:1 Analog Multiplexer
❖ Use either the 555-timer circuit or RCX to generate a clock
RCX Sensor Input 2

Figure 4.1 Block diagram of infrared detector [8]

The easiest way to transmit an infrared signal at a specific frequency is by using the 555-timer, as shown below in Figure 4.2. The frequency of the output signal can be adjusted by changing R_1, R_2, or C_1. The frequency is related to these circuit elements by $f = \frac{1.44}{(R_1 + 2R_2)C_1}$ [8]. The obvious problem is that in order to change the emitted frequency, the circuit elements have to be changed. To make it simple to change the frequency, R_2 and C_1 should remain constant while R_1 should be adjustable. This circuit configuration still requires an operator to change the emitted frequency. Therefore, it cannot be used for robot-to-robot communication. As with the RCX infrared transmitter, this infrared transmitter must be aligned with the detector in order to receive the signal. For this reason, as well as the inability to send signals at different frequencies automatically, this method is not used.

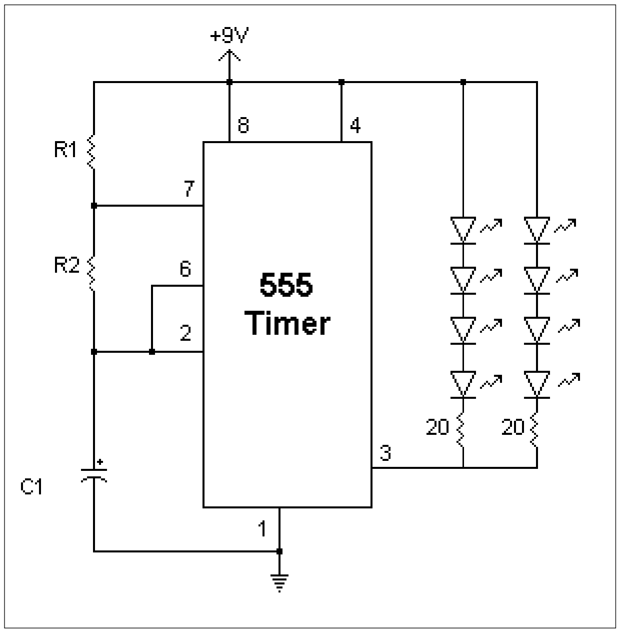


Figure 4.2 Block diagram of infrared emitter [8]

Section 4.2 – Audible Sound

The second technique for establishing a communication link between robots is by using audible sound. The major advantage to using sound over infrared light is that the sound can be detected from any direction, especially when using an omnidirectional microphone. In addition, if there is an object blocking the infrared signal, the receiver will not detect the signal. However, this is not the case with an audible sound signal. The microphone can be orientated away from the sound source and still detect the sound. Another benefit of using audible sound for communication between the robots is the fact that the RCX has sound generating capabilities. The RCX can play tones of a specific frequency and duration using NQC commands [6]. This prevents the need for additional hardware required to generate tones. It also allows for the robots to communicate with each other instead of just having a human operator issuing instructions.

There are a few drawbacks to using sound. Weak signal strength or noise could cause problems when receiving the sound signal. However, with proper amplification and filtering, these problems can be reduced significantly. Also, in hostile environments, audible sound used to communicate between robots will not prevent the robots from being detected by the opposition. Therefore, it is not a viable option for robot-to-robot communication.

The idea of using sound as of method of communicating between robots stemmed from the research conducted in the Center for Robotics and Intelligent Machines (CRIM) during the summer of 2002 regarding the development of a flexible acoustic array [32]. This large-scale acoustic array was designed to use triangulation and beam-forming methods in order to determine the location of and track a sound source. From this research, the concept of creating a scaled-down version of this acoustic array that could be used on an autonomous mobile robot was developed. The smaller version of the large-scale acoustic array could also be used for locating the position of a sound source and for tracking sounds, as well as for communicating with other robots. For the purpose of this research, a "one-element array" was developed in order to investigate the communication among robots.

Chapter 5 – LEGO® Robot Colony

The robot colony developed for this project stems from previous research that resulted in the creation of LEGO® robots that mimic the behavior a sheep and sheepdog. For that project, the goal of the sheepdog was to locate the sheep and herd it into a pen. Every movement made by the dog depended on its orientation relative to the sheep and the pen. The overall "thought process" of the dog was to move around the sheep such that the sheep was between itself and the pen. Once this occurred, the dog would then attempt to force the sheep into the pen. The pen is located in the corner of an 8' x 8' "field" enclosed by black walls. The new robot colony consists of two additional agents, as well as the sheep and sheepdog. While the overall task of the robots that comprise the new colony remains the same (for the sheepdog to force the sheep into the pen), the major objectives of this project are quite different.

Section 5.1 – Robot Colony Agents

This robot colony consists of four agents, each with a different task. Two agents, the sheep and sheepdog, have already been introduced. The last robot that comprises this multi-agent robot colony is essentially a "helper dog" robot. As indicated by its name, the purpose of this robot is to help the sheepdog when needed. The final agent of the colony is a human operator, or "shepherd." The human oversees the mission and provides guidance and instruction when needed. The role of these agents, as well as their design, will now be discussed in detail. Pictures of each robot are included in the Appendix.

Section 5.1.1 – Robot Design and Sensors

This section provides an in-depth description of several design issues that were present during the development of this robot colony. For the most part, the basic design of all three robots is the same. There are, however, some differences among the three. The basic design of the robots will be presented first, followed by specific details of each robot.

Section 5.1.1.1 – Basic Robot Design

For each robot, two motors located at the rear are used to drive and steer the robot. The drive motors are shown in Figure 5.1. For this application, the gear ratio is 1:1. No special gearing configuration was required for this design. By rotating the motors the same direction, the sheep will move either forward or backward. If the motors are rotated in opposite directions, the robot will effectively turn left or right. This method of "turning" the robot actually causes it to spin. In fact, it is based on the way a tank turns. The original design included tank treads, but the weight of the robot increased with the addition of extra components and resulted in difficulties with turning. For this reason, the tank treads were removed. Since there is no caster on the robot, the two front wheels have no tires. By not putting tires on the wheels, they will slide along the floor when the robot spins, resulting in an effective method of turning the robot.

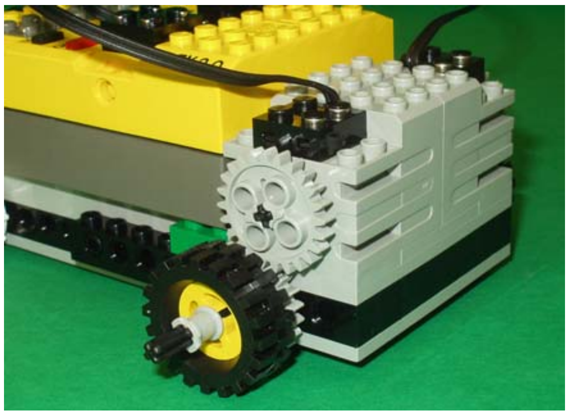


Figure 5.1 Left and right rear drive motors

A third motor is used to rotate a LEGO® light sensor on the sheep and sheepdog. The light sensor is attached to a shaft that is "connected" to the motor shaft using a worm gear. This allows the motor to rotate at high speeds, while rotating the light sensor at a much slower speed. A picture of the scanner is shown in Figure 5.2. The LEGO® rotation sensor is mounted on the shaft of the motor. It monitors the position of the light sensor. Approximately 4.25 rotations by the motor correspond to rotating the light sensor by 180º. By having the rotation sensor mounted on the motor shaft before any gear reduction, better resolution and less error are achieved.

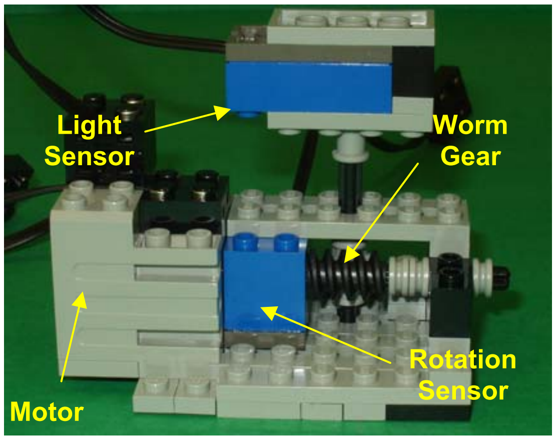


Figure 5.2 Light sensor scanning device

One of the problems that occurred frequently while running experiments were the robots kept getting stuck along the walls and with each other. The walls are comprised of several 29.5" x 1.5" x 9" (L x W x H) boards attached together. In some places, the wall sections do not come together flush. A front bumper with wheels was added to ensure that the robots would not get stuck at these places if they ended up along the wall. A picture of the front bumper is shown in Figure 5.3. Left-side and right-side bumpers (Figure 5.4) were added to prevent the robots from getting caught on each other. These bumpers run along the entire

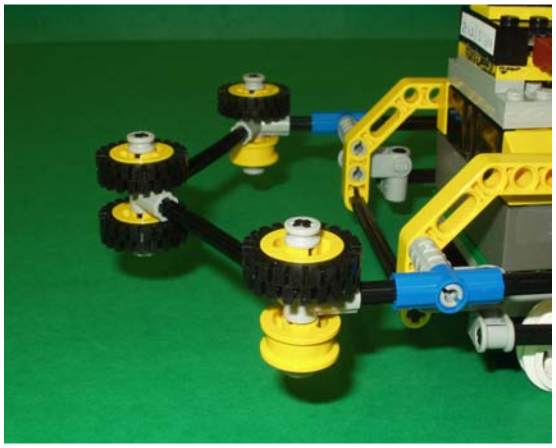


Figure 5.3 Front bumper

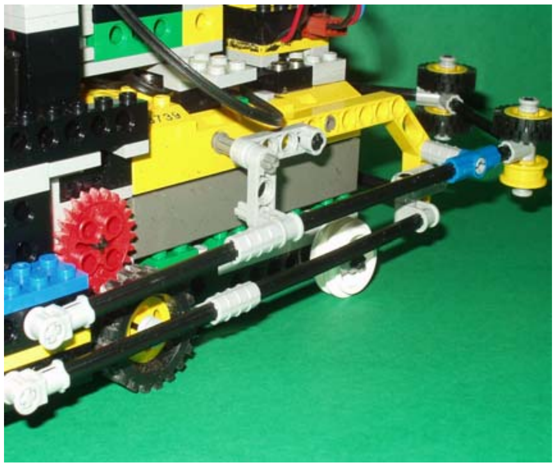


Figure 5.4 Side bumper

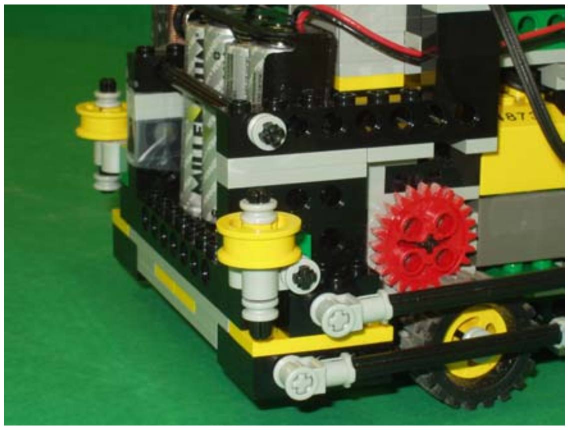


Figure 5.5 Back corner wheels to aid when turning near a wall

length of the robot and protect the wheels of the robots. The addition of bumpers around the robots significantly reduced the number of times the robots became stuck. These bumpers serve as a barrier around the robot and are not part of any sensor. Occasionally, the back of the robots would become stuck on the wall if they attempted to turn while in contact with them. In order to remedy this problem, wheels were added to the back corners of the robots. The addition of these wheels allows the robots to slide along the walls if they run into them while turning. Figure 5.5 shows these wheels on the robot.

Section 5.1.1.2 – Sheep

There are three sensors used on the sheep. The LEGO® light sensor is used to detect both the wall and the sheepdog. It is set to output the raw (0 to 1023) values when measuring light. The sensor value approaches zero as the amount of light detected increases. Since the walls are black, the light sensor will read a high number (greater than 900) when it is close to a wall. When the dog is near, the light sensor will detect the light mounted on the dog and read a lower number (less than 720).

The location of the light sensor is critical for detecting the sheepdog. To reduce the effects of ambient light on the sensor, a 2x1 LEGO® Technic block was placed in front of the sensor. This makes the light sensor extremely directional by only allowing light to enter through the hole in the Technic brick. Since this is the case, the light on the sheepdog must be at the same height as the light sensor on the sheep. Otherwise, the sheep would not be able to “see” the dog. It is not vital that the sheep recognize the dog from across the field. The sheep is only concerned when the dog is in close proximity. For this reason, a Maglite flashlight bulb is used to identify the sheepdog. Figure 5.6 shows the Maglite flashlight bulb used to identify the dog.

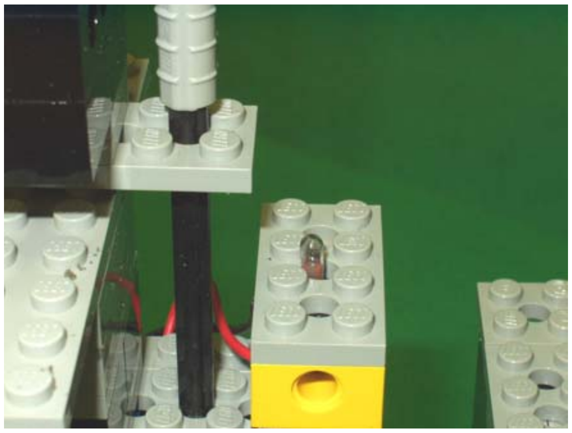


Figure 5.6 Light bulb used to identify the dog

The second sensor used is the LEGO® rotation sensor. This sensor is used to determine the direction the light sensor is pointing relative to the front of the sheep. The information provided by the sensor is used to determine which direction to move away from either the wall or dog. It is also used to determine when the sensor has rotated approximately 180º. Initially, the light sensor is pointing toward the front of the sheep, and the rotation sensor reads zero. While the light sensor is scanning, if the rotation sensor reads a value that corresponds to ±180°, it will cause the light sensor to stop and rotate in the opposite direction. This prevents the wire connecting the sensor to the RCX from becoming wrapped around the rotating shaft. The rotation sensor also helps determine on which side of the sheep the dog or wall has been detected. For example, if the sensor reads a negative value (other than the value corresponding to -180°), the light sensor is pointing toward the left side of the sheep. Conversely, if the sensor reads a positive value (other than the value equivalent to +180°), the light sensor is pointing toward the right side of the sheep. By using the value of the rotation sensor, the sheep know which way to turn so it can move away from any trouble.

Section 5.1.1.3 – Sheepdog

The sheepdog uses four sensors to complete the task of herding the sheep into the pen. Like the sheep, it has a scanning device with a light sensor mounted on a rotating shaft. A rotation sensor is used to monitor the position of the light sensor. However, this light sensor is used to find the pen, and not used for detecting walls or the sheep. The location of the light sensor is critical in order for the dog to locate the pen. The light sensor must be at the same height of the infrared array mounted on the pen or the dog will not be able to herd the sheep into the pen. An array of infrared LEDs is used to identify the pen because the dog needs to be able to find the pen from any position in the field. The dog is also equipped with same bumpers that the sheep has. The front bumper has three wheels that prevent the robot from becoming stuck along the wall, and the side bumpers run along the length of the robot. There are also back corner wheels that facilitate turning if it is along the wall.

The second light sensor on the dog is used to detect the sheep and walls. Unlike the light sensor used to detect the pen, this sensor does not rotate. Since all three motor outputs are being used, this sensor must remain fixed. In order to search for the sheep using this sensor, the dog spins in a circle until the light sensor detects the infrared LED array (shown in Figure 5.7) mounted on the sheep. This array of ten infrared LEDs is positioned on the sheep such that they are at the same height as a light sensor on the sheepdog. Infrared LEDs, rather than light bulbs, are used because the dog needs to be able to identify the sheep from across the field. This

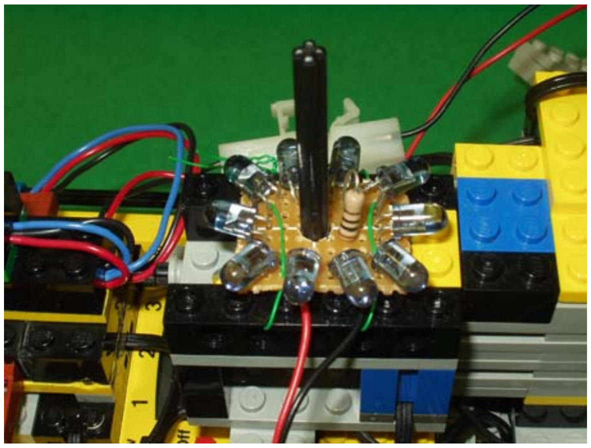


Figure 5.7 Infrared LED array on the sheep

technique of detecting the sheep has proven to be effective since tracking the sheep is not required. Once the sheep has been found, the dog immediately moves to a location based on its distance from the sheep and orientation relative to the pen. For this reason, the pen, rather than the sheep, must be tracked at all times. The final sensor used by the dog is a tone detection sensor. This sensor is used to listen for a specific frequency tone. When the dog hears this tone, it will stop its current task and perform different, predefined task. A detailed discussion regarding the design of this sensor is presented later in this chapter.

With only three sensor input ports and four sensors being used, there is a slight problem. Several hobbyists [20] have devised numerous ways of overcoming the problem of a lack of sensor inputs. One solution is to use a multiplexer so multiple sensors can be connected to the same port. A three-channel active multiplexer from Mindsensors Robotics [1] was used to allow the dog to have four sensors. The term "active" implies that power is required to operate each sensor (i.e. rotation, light, etc.). A picture of the Mindsensors Robotics multiplexer is shown in Figure 5.8. This simple multiplexer consists of one output port and three input ports. The output port connects the multiplexer to an RCX sensor input port. The three input ports are used to connect the sensors. A circuit diagram of the multiplexer is shown in the Appendix. For this application, only two sensors (both light sensors) are connected to the multiplexer. In order to cycle through each output of a multiplexer, a clock is required. Switching the sensor type of the input port from *light* to *touch* and back to *light* will create a clock. Because the light sensor

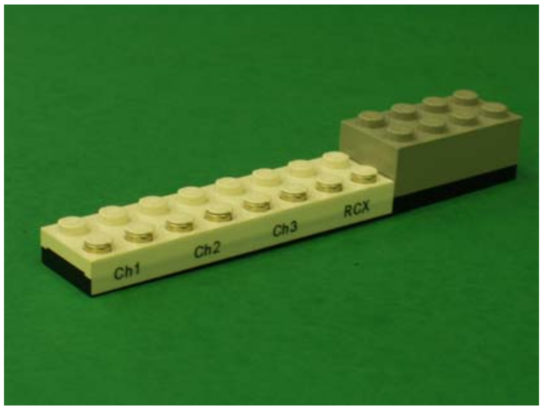


Figure 5.8 Three-channel active multiplexer

requires power to operate and the touch sensor does not, changing sensor types effectively creates a clock that can be used to cycle from one input to another. An interesting feature of this multiplexer is that it is not necessary to sequentially cycle through inputs. For example, Channel 3 can be selected after Channel 1 has been selected without having to read Channel 2. One benefit of using this multiplexer is that only one channel receives power at a time. For instance, if it is desired to switch from Channel 2 to Channel 3, power will only be applied to Channel 3 once the switch has occurred. This results in the power requirement of only one sensor when using the multiplexer regardless of the amount of sensors connected to the multiplexer. The one drawback is that only one of the possible three sensors can be used at a time. If a sensor must remain on for the entire mission, it cannot be connected to the RCX via the multiplexer. The multiplexer allows for a larger sensing suite onboard the robot, however, only a total of three sensors can be used at once.

Section 5.1.1.4 – Helper Dog

The purpose of this robot is to help the sheepdog corral the sheep into the pen when called upon. The helper dog uses three sensors (two light sensors and one microphone sensor) when providing help to the sheepdog. Like the sheepdog, this robot has two light sensors that are used to detect the sheep, wall, and pen. One sensor is at the same height as the infrared LED array on the sheep. This sensor is also used to detect a wall. The second light sensor is at the same height as the infrared LED array on the pen. The only difference with these sensors when compared to those on the sheepdog is that they are both fixed and oriented pointing forward. Since all of the sensor inputs are being used, there is not a port available to use an angle sensor for monitoring the rotation of a light sensor. Instead of being able to scan and look for the pen or sheep, the helper dog must turn to look for both. This causes a few problems in regards to decision making, but they can be overcome because of the role the helper dog plays during this mission. The microphone sensor is used to listen for a signal instructing the dog to help the sheepdog with corralling the sheep. This sensor will be discussed in detail later in this chapter.

Like the sheep and sheepdog, there are also bumpers present on the helper dog. The front bumpers with wheels help keep the dog from getting stuck on the wall. The side bumpers prevent the wheels of the robots from becoming caught on each other. There are also wheels mounted on the back corners of the robot to facilitate turning when near a wall.

The helper dog uses a Maglite flashlight bulb in the same manner as the sheepdog. The light bulb is placed at the same height as the light sensor on the sheep. When the sheep comes close to the helper dog, it will run away. The sheep has the identical reaction to the helper dog as it does with the sheepdog. Again, it is not important that the sheep see the helper dog from anywhere in the field.

Section 5.1.1.5 – Human Agent

The final agent that comprises this robot colony is the human operator. For this application, the human serves as the shepherd of the flock. He allows the dog to herd the sheep into the pen independently, but also provides assistance when necessary. In other words, the human operator is the manager of this mission. If he deems it necessary that the dog requires help, he will instruct the helper dog to help herd the sheep into the pen by playing a tone. The human operator plays the role of the mother robot in a marsupial robot colony by overseeing the mission and providing guidance or help when needed.

Section 5.1.2 – Control Software

In this section, the software used to control the robots will be discussed. Each robot was programmed using the Not Quite C (NQC) programming language developed by Dave Baum. This language was chosen because of the researcher's familiarity with the C programming language. The NQC code for each robot is provided in the Appendix.

Section 5.1.2.1 – Sheep NQC Control Software

The behavioral model of the sheep consists of four major components: look for the dog or wall, randomly move around in the field, run away from the dog, and move away from the wall. A diagram of the behavioral model of the sheep is shown in Figure 5.9. Initially, the sheep is wandering, or "grazing," in the field. The sheep moves around the area randomly, thus simulating the grazing pattern of sheep. While grazing, the sheep is also scanning and looking for the dog or wall. The scanner is rotated 180° clockwise, and then it is rotated 360° in the counterclockwise direction. Once the light sensor has finished this rotation, it reverses its direction of rotation and completes another 360° rotation. This process is repeated unless the dog or a wall is detected. Both actions (scanning and

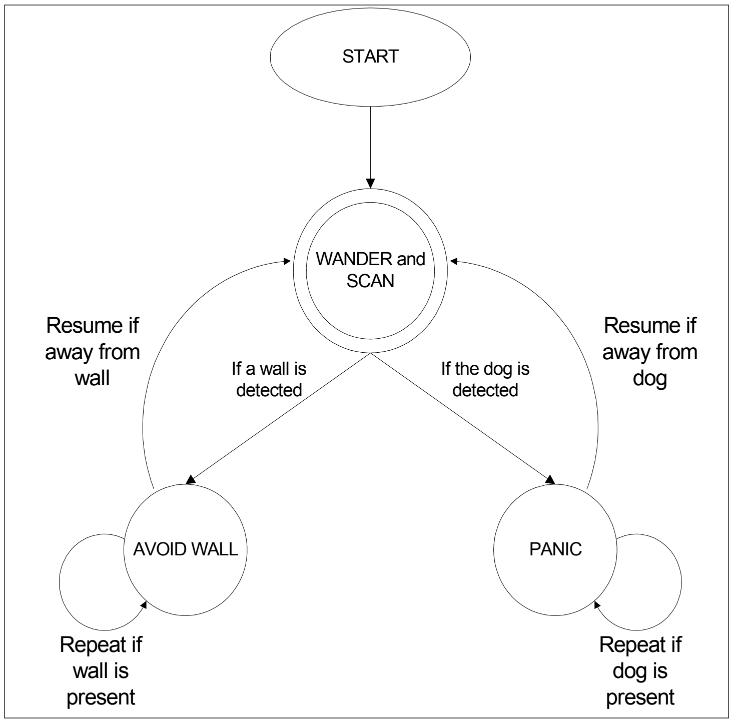


Figure 5.9 Behavioral model of sheep

wandering) are defined as tasks and they occur simultaneously. Once started, they will continue unless the *stop* command is issued.

When the sheep detects either a wall or the dog, it will stop grazing and perform the appropriate action depending on what has been noticed. If the sheep encounters a wall, it will calmly move away from it and then resume with wandering and scanning. However, if the sheep detects the dog, it will panic and run away. Both of these actions are accomplished in similar manners. First, the wander task is stopped, and then the corresponding function that will move the sheep away from the wall or dog is called. The direction the sheep will move depends on where the wall or dog is detected. For example, when the wall or dog is detected on the left side of the sheep, it will spin clockwise (to the right) and then move away from the wall or dog. The major difference with these two actions is the speed at which the sheep moves away. It treats the wall as a fence and calmly moves away from it. The sheep, however, fears the dog and runs away when it is detected. Both maneuvers are repeated until the sheep is at a safe distance from either one. The sheep then returns to grazing and scanning.

Section 5.1.2.2 – Sheepdog NQC Control Software

The behavioral model of the sheep dog is comprised of two parts: avoid walls and corral the sheep into the pen. A diagram of its behavior model is provided in Figure 5.10. Initially, both tasks occur simultaneously. If a wall is detected, the sheepdog will stop the corral task, move away from the wall, and then resume corralling the sheep. If the dog remains near a wall, it will continue to move away from the wall before restarting the process of herding the sheep into the pen. If a wall is not detected, the sheepdog will continually repeat the corral task until the sheep has been funneled into the pen. If, at any time, the sheepdog becomes stuck, a procedure is attempted to free the sheepdog. If the angle sensor does not move over a specified amount of time, the sheepdog will assume it has become stuck. It will then move backward and attempt to free itself.

The corral task is made up of five functions. A block diagram of the corral task is shown in Figure 5.11. The first action undertaken by the sheepdog is finding the sheep. The dog uses the fixed light sensor to "look" for the sheep. It spins around until the infrared light emitted from the array mounted on the sheep is detected. Once the sheep has been detected, the sheepdog stops spinning. The dog then

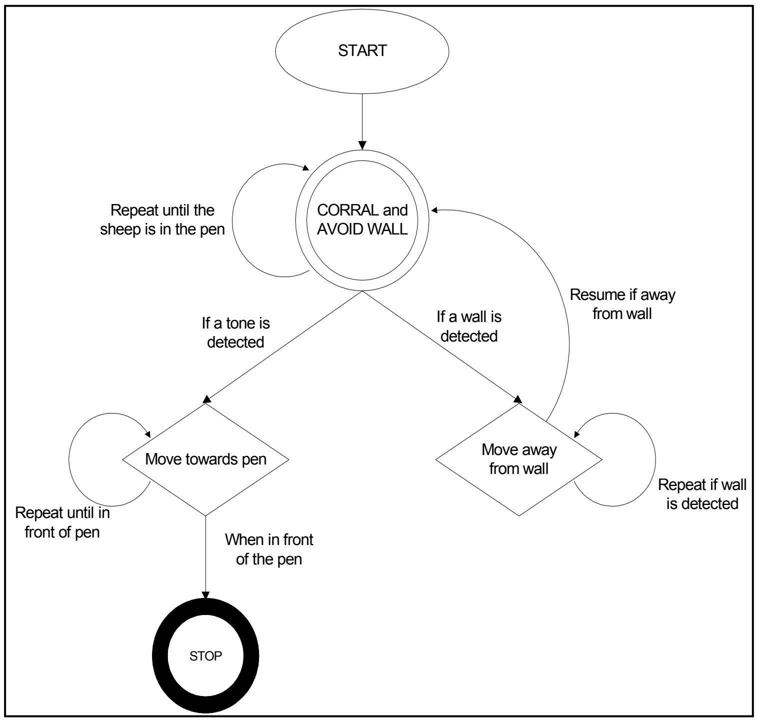


Figure 5.10 Behavioral model of sheepdog

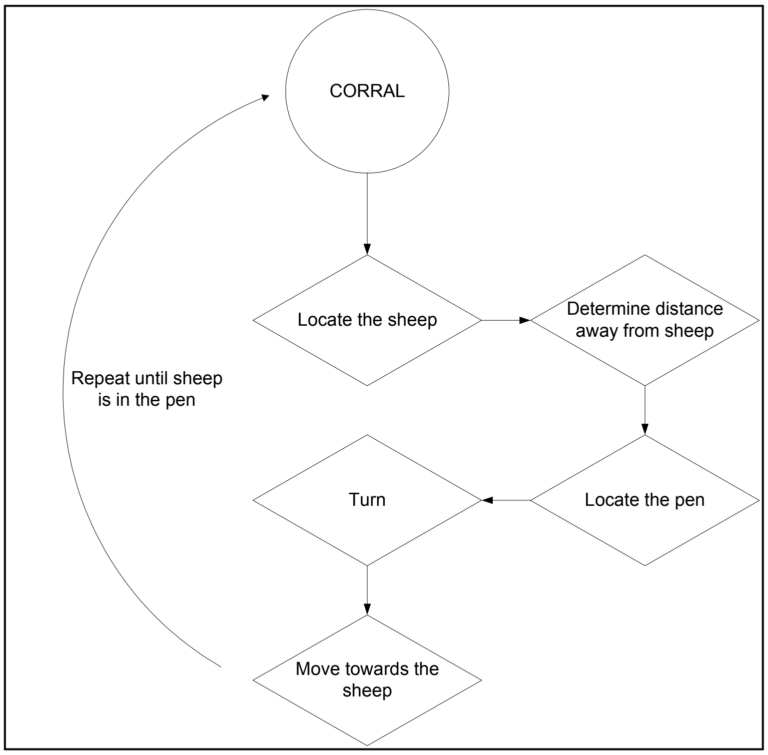

CORRAL
Locate the sheep
Determine distance away from sheep
Repeat until sheep is in the pen
Turn
Locate the pen
Move towards the sheep

Figure 5.11 Block diagram of corral task

determines its distance from the sheep based on the strength of the light signal detected. This distance will determine how far the sheepdog should travel before repeating the corral task. Using the light sensor mounted on the rotating shaft, the dog then searches for the pen. Once the pen has been found, the position of the light sensor is recorded and used to help determine how the sheepdog should approach the sheep. Based on the orientation of the light sensor, the dog will spin either clockwise or counterclockwise. This is performed with the intention of having the dog move to one side of the sheep rather than directly at it. Eventually the dog will move into a position such that the sheep is between itself and the pen. When this is the case, the dog will not need to turn before traveling toward the sheep, and it can move directly towards the sheep. The final action that is performed in the corral task is moving toward the sheep. Based on the distance to the sheep that was determined earlier, the dog will move forward for a corresponding amount of time. The task is then repeated until the sheep has been corralled into the pen.

If the microphone sensor detects a tone, all actions are stopped in order to execute a "higher" priority action. This action is analogous to one member of a marsupial robot colony sending a distress signal to other team members. When another team member receives a call for help, the robot will go to the aid of the teammate regardless of what was being performed prior to getting the call. However, this signal does not necessarily have to represent a call for help. If a member of a search and rescue team finds a survivor, a signal can be sent to the other team members and system operators to alert them of the discovery. Other team members can then respond to the call and give additional aid to the victim. For this application, the sheepdog is instructed to return to the pen when the microphone sensor detects a tone.

Section 5.1.2.3 – Helper Dog NQC Control Software

The behavioral model of the helper dog is not complex. It has three parts: to wander around the field, to avoid walls, and to help the sheepdog corral the sheep. A block diagram of the behavioral model of the helper dog is shown in Figure 5.12. Initially, the helper dog wanders around the field. While it is moving in the field, if a wall is detected, it will stop, move away from the wall, and then resume its movements. During the time the helper dog is moving in the field, it is not attempting to detect the sheep, sheepdog, or pen. Since its light bulb will be on, the helper dog should not have a problem with getting stuck on the sheep. If it gets too close to the sheep, the sheep will detect the light and move away from the helper dog. There could be some problems with the sheepdog, since the helper dog cannot see it. However, since the sheepdog is tracking the sheep and the sheep will avoid the helper dog, the sheepdog should also avoid the helper dog.

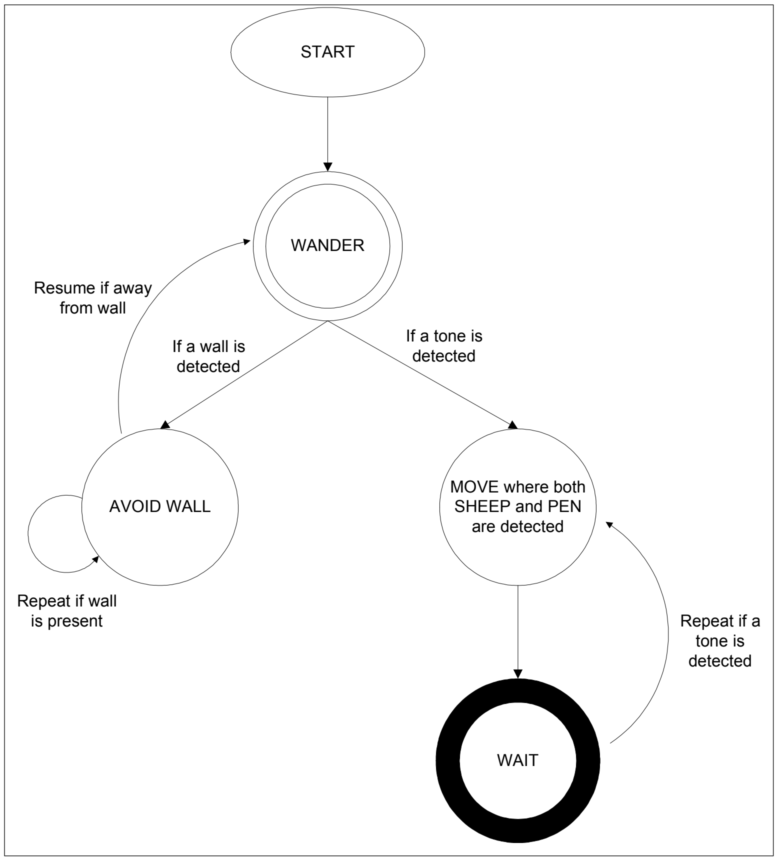

START
WANDER
Resume if away from wall
If a wall is detected
If a tone is detected
AVOID WALL
MOVE where both SHEEP and PEN are detected
Repeat if wall is present
Repeat if a tone is detected
WAIT

Figure 5.12 Behavioral model of the helper dog

If the microphone sensor on the helper dog detects a tone, it will stop "wandering" and begin to help corral the sheep. The helper dog will position itself such that the sheep is between itself and the pen. If both light sensors detect the sheep and the pen, the helper dog will know the sheep is between the pen and itself. Once the helper dog has reached this position, it will stop and wait for another tone. When it detects another tone, it will again find the sheep and move into a position where the sheep and pen are in front of itself. The purpose of moving to this position is to reduce the area the sheep will be able to move.

Section 5.2 – Microphone Sensor

In order to make communication between robots possible, a sensor was designed to recognize a certain frequency tone. The microphone sensor developed for robot-to-robot communication will be discussed in this section. The circuit design, sensor construction, and sensor interfacing will be presented in detail. The datasheets of the major components of the circuit are included in the Appendix.

Section 5.2.1 – Circuit Design

The circuit for the microphone sensor can be divided into two distinct parts: input sound amplification and frequency detection. The microphone circuit used to detect and amplify the sound was designed during the CRIM development of a large-scale acoustic array. This circuit consists of a microphone and an operational amplifier configured in a non-inverting manner. A circuit diagram is provided in Figure 5.13. Notice that the microphone is a two-pin microphone. One pin is connected to ground, and the other is for power/output. In order to block the DC voltage used to power the microphone, a capacitor was added to the input of the op-amp. In addition, since there is not a negative supply voltage available when using four AA batteries, the input has to be biased in order for the entire input signal to be amplified. The microphone used for sound detection is the Panasonic omnidirectional electret condenser microphone cartridge [39]. This microphone

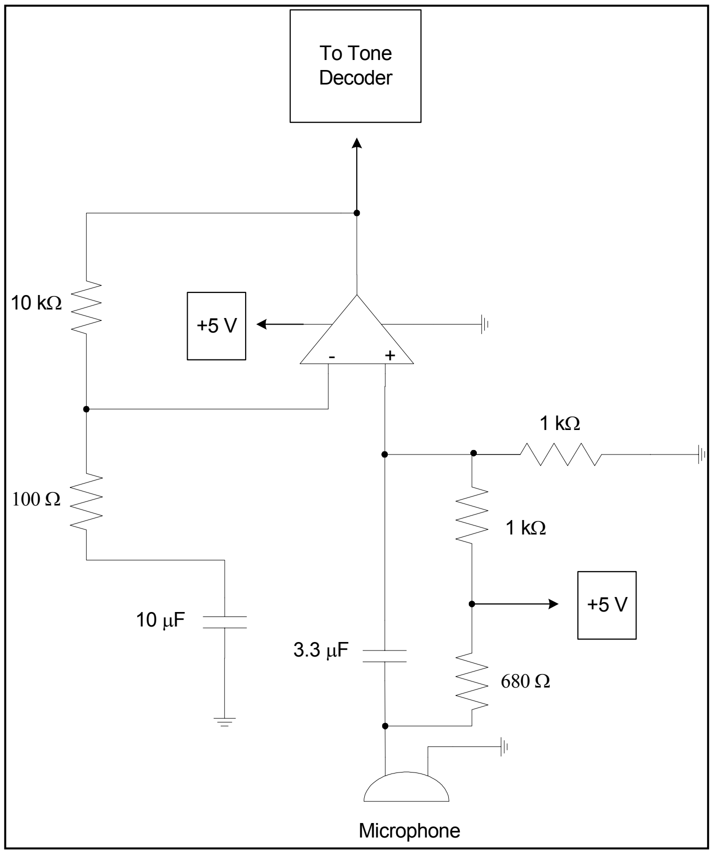


Figure 5.13 Circuit diagram of the microphone amplifier

was chosen primarily because it can detect sound from any direction, but also for the wide frequency response (20 Hz to 16 kHz) and the wide range of operating voltages (maximum of 10 volts). The op-amp used to amplify the detected sound is the Texas Instruments μA741 general-purpose operational amplifier [48]. Note that the gain of the amplifier circuit is 100. In order to boost the signal strength for the tone detection circuit, the gain needs to be as large as possible without saturating the signal.

The tone detection circuit used in this sensor was developed as a simple way to convert an analog input to a digital output in order to interface the sensor with the RCX. The basic idea behind the tone detection circuit is to send a digital output to the RCX when a specific frequency sound is detected. The integrated circuit used to perform the frequency detection is the New Japan Radio FSK Demodulator/Tone Decoder chip [38]. The internal components of this chip are shown in Figure 5.14. Using this component, the output is either a logic *high* or logic *low*. Using Figures 5.14 and the tone detection circuit diagram

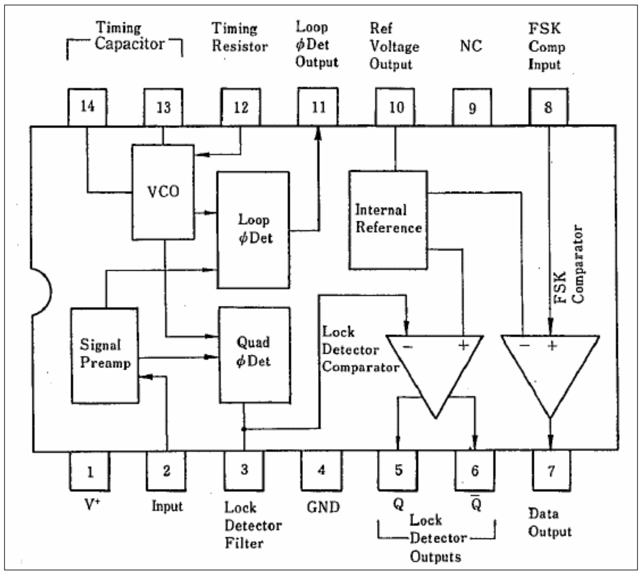


Figure 5.14 Internal components of tone detection chip [38]

(Figure 5.15) as reference, the roles of several circuit components can be explained. The components that are used to set the center frequency are resistors R, R_x (used to fine tune the center frequency), and capacitor C_0. Resistor R_1 sets the detection bandwidth, capacitor C_1 sets the lowpass-loop filter time constant and the loop damping factor, R_Q is a pull-up resistor for the logic *high* output, and resistor R_D and capacitor C_D are used to prevent chatter at the logic output. The following steps are taken to determine the values of certain components of the tone detection circuit [38].

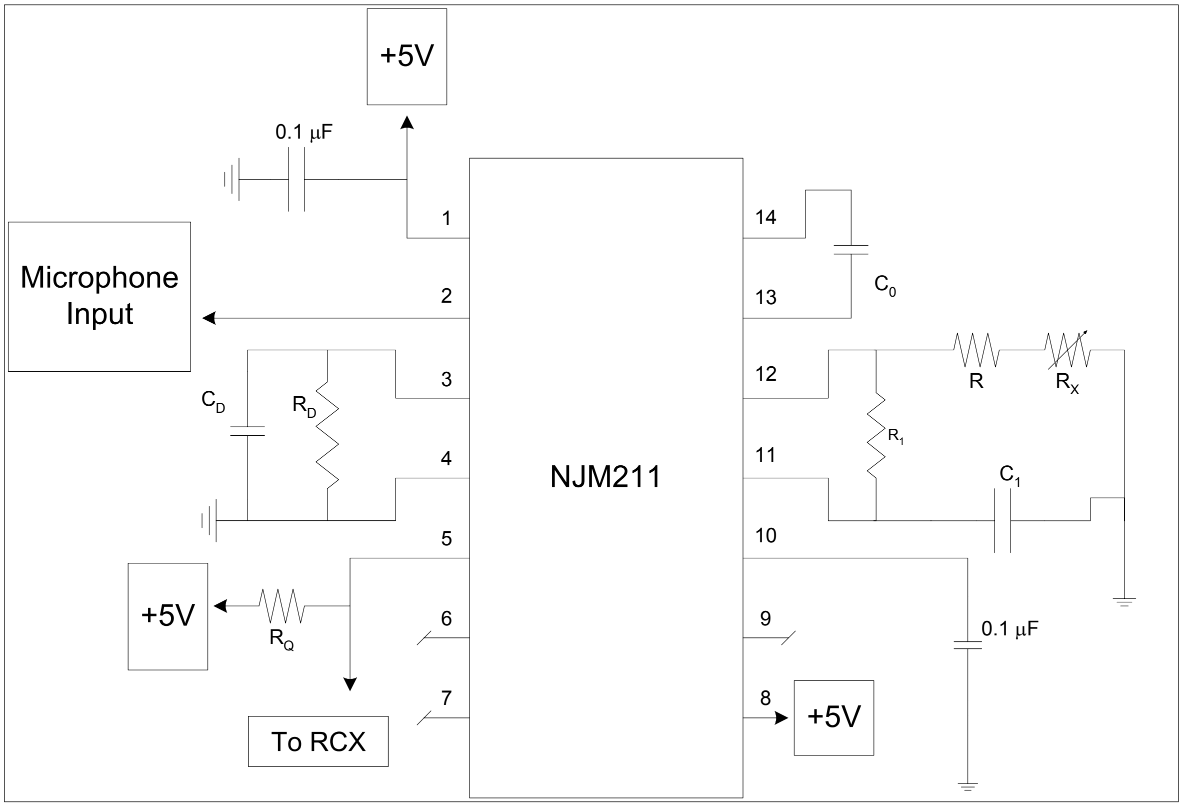


Figure 5.15 Tone detection circuit

Step 1: Pick the center frequency, f_o, in Hz and the bandwidth, $\pm \Delta f$, in Hz. Set R + R_x to be any value between the range of 15kΩ to 100kΩ. Calculate C_o to set f_o using

$$C_o = \frac{1}{(R + R_x) f_o} \qquad \textbf{Equation 5.1}$$

Step 2: Calculate R_1 to set $\pm \Delta f$ using

$$R_1 = \frac{(R + R_x) \times f_o}{\Delta f} \qquad \textbf{Equation 5.2}$$

Step 3: Calculate C_1 for a given loop damping factor ζ using

$$C_1 = \frac{C_o}{16\xi^2} \qquad \textbf{Equation 5.3}$$

For most tone-detection applications, ζ = ½ is optimal. By increasing C_1, the out-of-band signal rejection is improved, but the phase-lock loop capture time is increased.

Step 4: Setting R_D = 470 kΩ, calculate C_D (in μF) to avoid chatter at the logic output using

$$C_D \geq \frac{16}{\text{(capture range in Hz)}} \qquad \textbf{Equation 5.4}$$

Note that by increasing C_D, the logic output response time decreases.

Using the above equations with f_o = 1.0 kHz, Δf = 20 Hz, R = 18kΩ, R_x = 5 kΩ tunable resistor, and R + R_x set to 20kΩ, the following values were calculated for the respective circuit components: C_o = 0.05μF, R_1 = 1MΩ, C_1 = 0.0125μF (with ζ = ½), and $C_D \geq$ 0.42μF. Some of these calculated values were rounded off to the nearest standard value. For the actual circuit, C_o = 56000pF (0.056μF), C_1 = 10000pF (0.01μF), and C_D = 0.47μF. In addition, only the value of R was changed to obtain a different center frequency. This was done so several sensors could be built without having to use many different components.

The chip used in the tone detection circuit was chosen for several reasons. First, the component has a wide frequency range, from 0.01 Hz to 300 kHz. Not only can this element be used for detecting audible sounds, it can also be used to detect signals above the human audible range, which is attractive for covert applications. Another feature of this chip is the ability to have a “tunable” bandwidth for the incoming signal. By changing only one resistor, different bandwidths can be set depending on the application. It also has an attractive operating voltage range, from 4.5 volts to 20 volts. This wide operating voltage range accommodates the operating voltages of both the microphone and the operational amplifier. Finally, it has a wide input range, from 2mV_{rms} to 3V_{rms}. Due to this wide dynamic range, the detected signal does not require several stages of amplification in order to boost the signal to a required level. In fact, it may not require any amplification at all, resulting in a reduction of the overall sensor size. Amplification was used in this application to prevent the use of an unnecessarily loud tone for communicating with the robots.

Section 5.2.2 – Sensor Construction

To facilitate easy integration of the microphone sensor with the LEGO® robots, it was determined that the circuit needed to be housed inside LEGO® blocks. In order to make this possible, surface mount components must be used to keep the overall size of the circuit minimal. This required a printed circuit board to be designed and fabricated. The layouts of the fabricated two-sided boards are shown in Figure 5.16. The fabricated boards with all

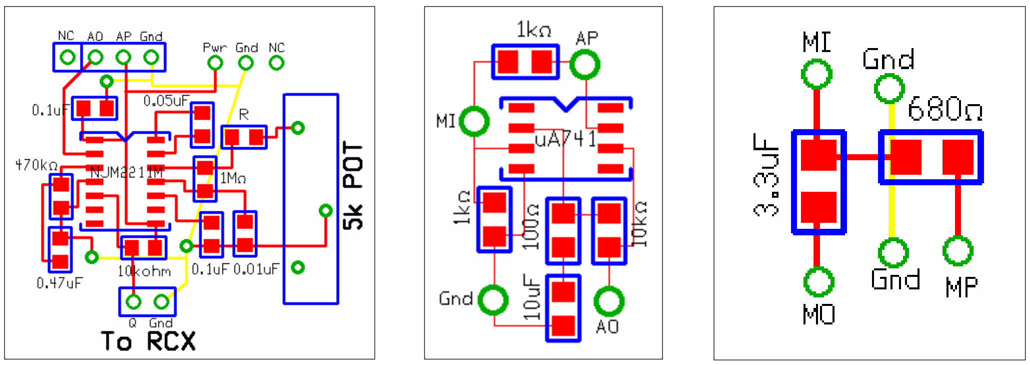


Figure 5.16 Circuit board layout for tone detection circuit (left), amplifier circuit (middle), and microphone circuit (right)

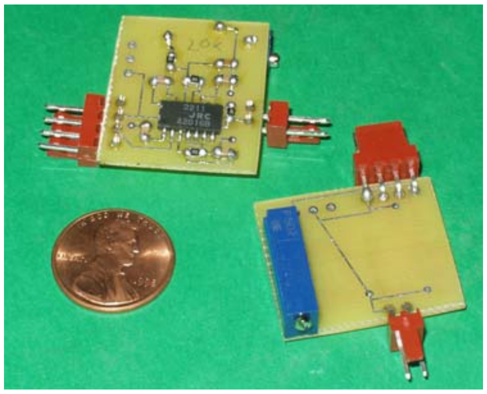
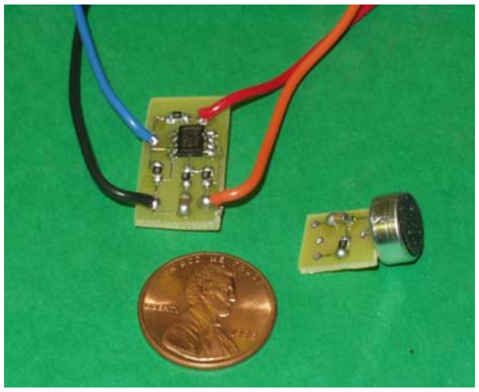



Figure 5.17 Top and bottom view of fabricated tone detection circuit board (left) and amplifier and microphone circuit boards (right)

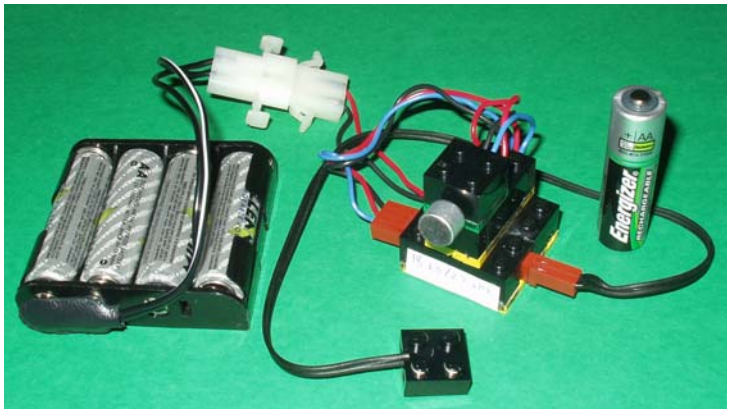


Figure 5.18 Completed tone detection sensor

components attached are shown in Figure 5.17. The inside of the LEGO® blocks were hollowed out to allow the circuit boards to fit inside the blocks. The microphone board (≈1cm x 1cm) was designed to fit inside of a 2x2 LEGO® brick. The amplifier board (≈1cm x 2cm) was designed to fit inside of a 2x4 LEGO® brick. The tone detection circuit (≈2.5cm x 3cm) was designed to fit inside of two 2x4 LEGO® bricks. To make the enclosure for the tone detection circuit, one side of each 2x4 brick was removed, and the two bricks were glued together to create a 4x4 LEGO® brick. The completed sensor is shown in Figure 5.18.

Section 5.2.3 – Interfacing the Sensor with the RCX

Since the output of the circuit is either logic *high* or logic *low*, interfacing the microphone sensor with the RCX was not difficult. First, the sensor input port that was used for the microphone was configured as sensor type *light*. This will result in a voltage at the input port. If the output of the tone detector circuit (pin 5) is connected to the RCX sensor input port, the voltage at the port will be forced to zero when the tone is detected. This will change the sensor raw value from 1023 to 0. When the value drops to zero, the robot will know a tone has been detected.

Several issues arose when this sensor was added to the LEGO® robots. Most interestingly, the center frequency was quite a bit higher than expected. This could have resulted because of the rounding that occurred when choosing components used in the tone detection circuit. There was also an error found in the circuit board layout after the boards were fabricated. However, the circuit did function properly at the higher center frequency, and the error was not corrected. Another problem that could be attributed to the error in the circuit layout is that constant recalibration must occur. The center frequency is greatly affected by the supply voltage level. As the batteries wear down, the center frequency slowly decreased in value. This might also occur since the operating voltage is close to the minimum value specified. A significant problem regarding the overall weight of the robot resulted when the tone detection sensor was added. The sensor itself is not heavy. However, the four AA batteries that are needed to power this circuit result in a significant amount of added weight. They were originally attached in the front of the robot, and this caused the rear wheels to spin without moving the robot. This problem was alleviated when the batteries were moved to the back of the robot. This added weight resulted normal movement by the robot.

Chapter 6 – Experiments and Results

Once the robot colony was constructed, several experiments were performed using some or all of the members. First, the original experiment was conducted with the upgraded robots. Next, communication between robots was investigated. Finally, the interaction between robot teammates was studied. Detailed accounts of the various experiments performed on the colony, as well as the results, are provided in this chapter.

Section 6.1 – Robot Calibration

Before any experiments could be conducted, a calibration of each robot occurred. The most important factor to consider was the light level in the room where the experiments were being conducted. Because the robots make decisions based on the amount of light detected by the light sensor, it is very important the light sensors on the robots be calibrated. First, the light sensor for the sheep, sheepdog, and helper dog used to detect the wall was calibrated. As the robot approaches the black walls, the output of the light sensor increases as the amount of light detected decreases. The identifying light source on each robot (infrared LED array or light bulb) must be turned on or the calibration will be incorrect. Some of the emitted light is reflected off the wall and detected by the light sensor. The reflected light effectively makes the walls appear lighter than they actually are. The output value of the light sensor was recorded at various positions along the wall, from about six to eight inches away. The normal measured value when not looking at a wall is in the range of 850 to 875. When near a wall, the minimum value for the sheep was 930, the minimum value for the sheepdog was 920, and the minimum value for the helper dog was 890. These values are used as the threshold value for the wall. The minimum value measured was used to ensure it would recognize the wall at all times. If the light sensor detected any value higher than this threshold, the robot would immediately take action in order to avoid the wall.

Next, the light sensor for the sheepdog and helper dog was calibrated to recognize the pen. Since it is required that both dogs be able to see the pen from anywhere in the field, the calibration must take place at the farthest point from the pen. Because the field is a square and the pen is in the corner, the farthest distance from the pen is along the diagonal. At this distance, the light sensor for both the sheepdog and helper dog measured 780. This value was used as the threshold for recognizing the pen. For any value less than the threshold, both dogs assume they are looking at the pen. The normal value measured by this sensor when not looking at the pen ranged from 800 to 850. Since there is no overlap, the robots should not think they are looking at the pen when, in reality, they are looking at something else.

Both the sheepdog and helper dog were then calibrated to detect the sheep. Again, it is required that the sheep can be detected from across the pen. Along the diagonal of the field, both robots measured 815. The same principle that was used for detecting the pen is used for sensing the sheep. For any value lower than 815, the sheepdog and helper dog would assume they had detected the sheep. The sensor used to detect the sheep is also used to detect the walls. This is possible since there is no overlap of the possible values of a wall or the sheep. The value for a wall is greater than 890 (for the helper dog) or 920 (for the sheepdog) and the value for a sheep is less than 815.

Finally, the sheep was calibrated to detect the sheepdog and helper dog. Since the sheep should run away from the dog when it is close, this calibration was performed from a distance of six inches. The maximum value measured from different locations around the two dogs was 730. For any value less than this, the sheep will run away from both dogs.

This sensor is also used to detect the walls. This is not a problem because the value for a wall is greater than 930 and the value for the sheepdog and helper dog is less than 730. There is no overlap of the possible values for the dog or a wall.

The other calibration that needed to be performed was for the sheepdog regarding the distance to move forward when tracking the sheep. Since the helper dog and sheep do not track anything, the distance they travel at any one time is not important. The sheep never has to worry about moving a precise distance to reach an object. It must only be concerned with running away from the dogs. The helper dog only needs to be in a position where the sheep is in between itself and the pen. It does not need to be a certain distance from the sheep. The sheepdog is the only robot that must move precise distances during operation. To minimize the time taken to herd the sheep into the pen, the sheepdog should move forward for a period of time that corresponds with the distance to the sheep. For example, if the light sensor on the sheepdog measures a value close to 815, the sheepdog is far away from the sheep. In order to reach the sheep, the sheepdog will need to move forward for a longer period of time than if it was closer to the sheep.

The first step in this calibration process was to measure how far the dog traveled for a given amount of time with the highest motor power. Figure 6.1 shows a plot of these measurements. As expected, the results are linear. The next step was to measure the value of the light sensor used to detect the sheep at different distances from the sheep. These measurements are shown in Figure 6.2. There were four different measurements taken at each distance corresponding to the front, back, left side, and right side of the sheep. The average of all four measurements was also calculated and graphed. The equation of the average trendline is shown below the legend. Notice these results are non-linear. Figure 6.3

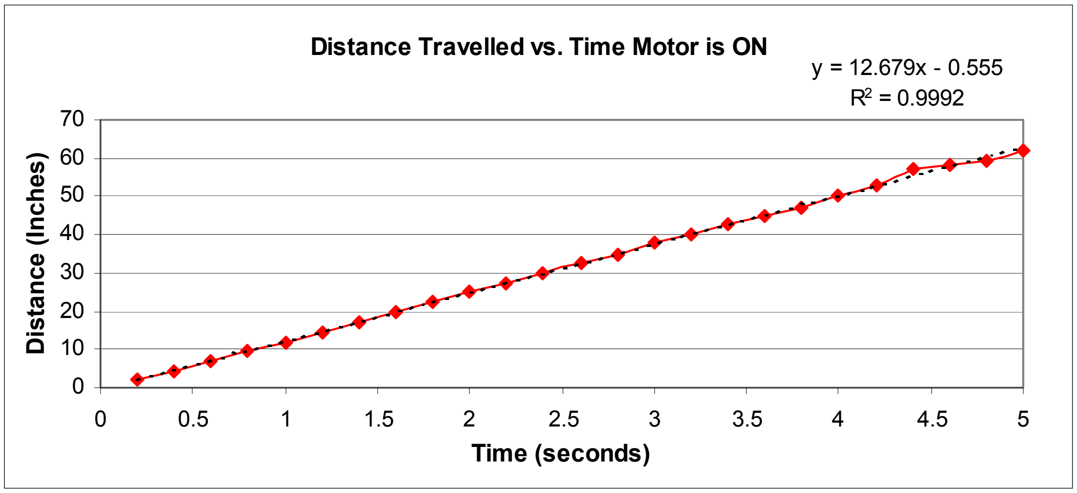


Figure 6.1 Distance the sheepdog traveled for a given amount of time

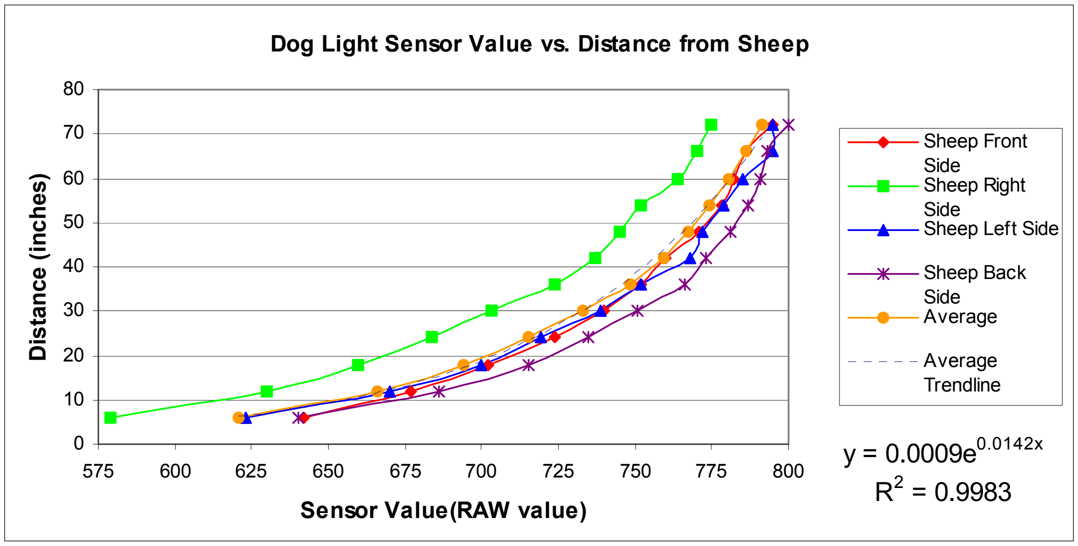


Figure 6.2 Sheepdog light sensor values at different distances from the sheep

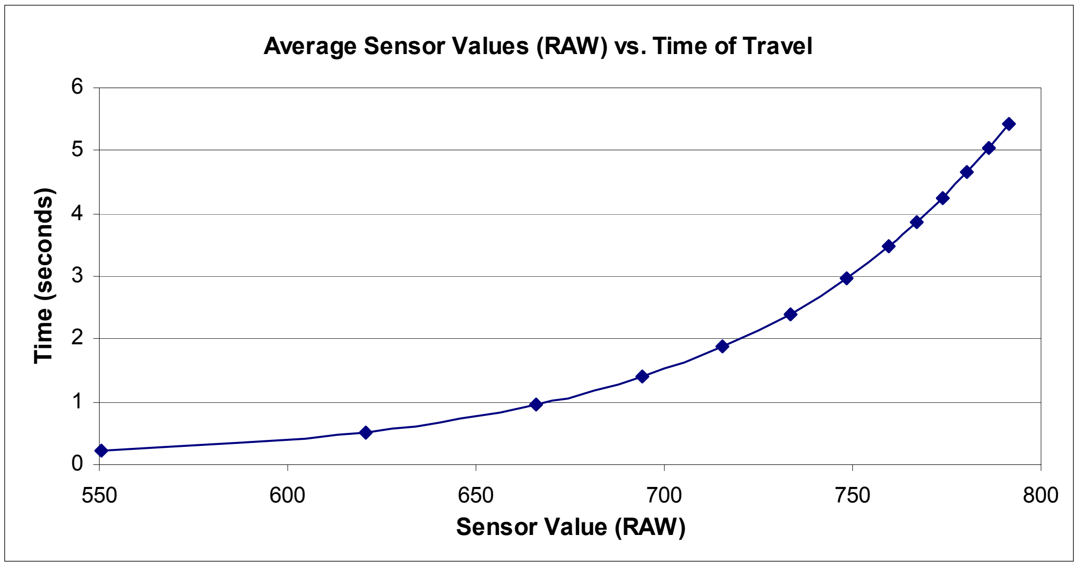


Figure 6.3 Amount of time to travel for a specific light sensor value

shows the time of travel for a specific light sensor value. This plot was generated using the average sensor values from Figure 6.2 and the equation of the generated trendline. This plot illustrates that if the light sensor read approximately 750, the dog must travel for almost three seconds to reach the sheep. Using this graph, a look-up table was created that matched a given time to travel with a range of sensor values.

Unfortunately, this calibration procedure did not work very well, especially for lower light sensor values. Since the dog finds the sheep by spinning, it will stop once the light sensor measures a value less than the maximum threshold for the sheep (815). However, the point where it detects the sheep might not where the beam of infrared light is at the maximum intensity. This causes the sheepdog to calculate an incorrect distance. This is not a problem when the sheepdog is far away from the sheep. For example, the light sensor could read 750 and the maximum intensity reading could be 740. This would not cause much of a difference in travel time, if any. However, if the sheepdog is close to the sheep and the light sensor reads 675 instead of 550, this results in a problem. The dog will travel a distance that corresponds to 675 instead of 550. More than likely, the sheepdog will go past the sheep, causing it to backtrack and resulting in additional time needed to the herd the sheep into the pen. Due to this problem, the travel times that correspond to a specific range of light sensor values were determined by trial and error.

Section 6.2 – Original Experiment

With the updated robots, the original experiment was recreated with the sheep and sheepdog. A number of experiments were performed with the sheep and sheepdog starting at different positions within the field. A grid was defined in order to accurately position the robots for each experiment. At the outset of each experiment, the robots were placed at the

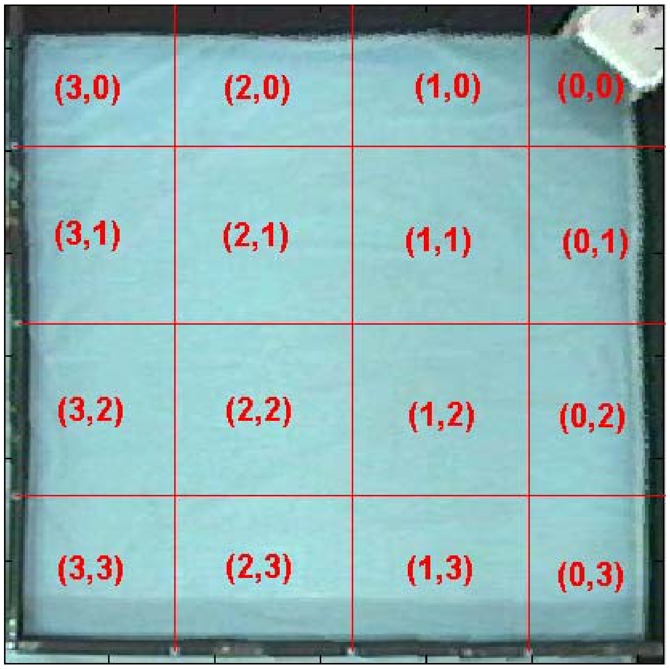


Figure 6.4 Field with starting position grid

center of one of the grid elements. Figure 6. 4 shows an overhead picture of the field with an overlay of the grid. Each block is approximately two feet by two feet, and the pen is defined to be at position (0,0). For each experiment, the initial position, start time, end time, and behavioral observations were recorded. The results are tabulated in Table 6.1. For the experiments, the

Table 6.1 Sheep and sheepdog experiment results

Sheepdog Starting Position	Sheep Starting Position	Total Time	Result
(0,0)	(3,3)	4 min 10 sec	Success
		5 min 10 sec	Fail
		5 min 56 sec	Fail
		7 min 8 sec	Fail
		3 min 59 sec	Fail
(0,0)	(2,2)	1 min 35 sec	Success
		1 min 39 sec	Success
		1 min 58 sec	Success
		4 min 2 sec	Success
		3 min 3 sec	Success
		5 min 13 sec	Fail
(2,2)	(1,1)	0 min 43 sec	Success
		1 min 25 sec	Success
		2 min 8 sec	Success
		6 min 34 sec	Fail
		2 min 50 sec	Fail
		4 min 19 sec	Fail
		1 min 48 sec	Fail
		6 min 4 sec	Fail
(0,3)	(2,1)	2 min 11 sec	Success
		4 min 4 sec	Success
		2 min 7 sec	Success
		2 min 39 sec	Success
		6 min 25 sec	Fail

overall success rate was just over 54%. This may seem a bit low, but sensor malfunction, incorrect robot calibration, or low battery levels caused most of the failures. The majority of these problems can easily be overcome. Figure 6.5 shows the path taken by the sheep and sheepdog during a successful trial. The total time taken to compete this experiment was 3 minutes and 31 seconds. An X marks the starting position of both robots. The sheepdog path is shown in red, and the sheep path is in green. From these results, several interesting behaviors are noticed. First, if the sheep is placed close to a wall (i.e. position (3,3)) at the outset of the experiment, the dog does not perform well. When the experiment begins, the sheep will immediately notice a wall and move away from it. This immediate movement seems to confuse the dog. It appears that the dog needs to formulate a plan and begin to implement it before the sheep moves. This is evident by the results achieved when the sheep was placed at position (2,2). The success rate was over 80% as opposed to only 20% when the sheep began at position (3,3). The dog was able to begin herding the sheep before the sheep made its first movement.

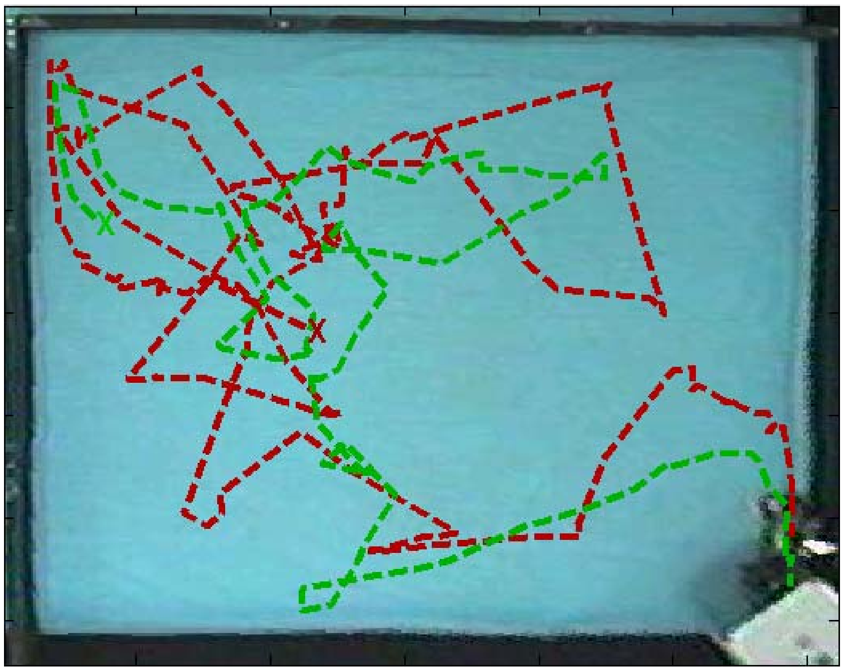


Figure 6.5 Paths taken by sheep and sheepdog

Another result was the least amount of time needed to herd the sheep into the pen occurred when the sheep began in between the sheepdog and the pen. This result was expected. However, the number of failures was not. As it turns out, four of the failures were due to a decrease in battery power provided to the infrared LED array on the sheep. As the voltage level drops, the strength of the infrared signal emitted from the array weakens. This results in the dog miscalculating the distance to the sheep or even not being able to find the sheep. When the 9V battery was replaced with a tether, there was only one failure and two successes.

An interesting statistic that resulted is the average time of success and failure. For a successful experiment, the average time to herd the sheep into the pen was 2 minutes and 26 seconds. The average time a failed experiment lasted was 5 minutes and 2 seconds. This leads to the belief that if the sheepdog is not able to herd the sheep into the pen relatively quickly, it will never be able to complete the mission. This could occur for a number of reasons. First of all, there is a heavy strain on the 9V battery used to power the infrared LEDs. The signal emitted from the LEDs is not pulsed, resulting in the battery being used constantly. After a few minutes, the battery will lose its strength, and the signal emitted from the array weakens. The longer an experiment lasts, the more distance miscalculations by the sheepdog will occur. However, allowing the battery to rest between experiments gives it the opportunity to regain some of its charge, and the experiments can continue. Also, as the light sensor rotates, some error occurs when measuring the amount of rotation. This error is due to the slippage of the scanning device. The longer the experiment takes, the larger the error will be. Eventually, it could reach a point where the error is so large that the sheepdog can no longer make accurate decisions based on the position of the light sensor. Finally, the robots became stuck during some trials. If, after a significant period of time, they were still stuck, the experiment was stopped.

Section 6.3 – Microphone Sensor Test

For this experiment, a microphone sensor was placed on the sheepdog. The purpose of this test was to determine if communication between robots was possible. Using the NQC command *PlayTone(frequency, duration)*, the RCX can generate a tone of a specific frequency in Hertz for a period of hundredths of seconds. The output of the microphone sensor on the sheepdog switches from logic *high* to logic *low* when a 2.8 kHz tone is detected. Initially, an RCX played a tone of 2.8 kHz and when the microphone sensor detected it, the dog was supposed to move forward. However, the tone played by the RCX was not loud enough for the microphone sensor to detect it. This indicates that robot-to-robot communication using sound is not feasible using LEGO® Mindstorms™.

Instead, a tone was generated using a function generator. The function generator was connected to a solid-state stereo amplifier, and the tone was output through a standard bookshelf stereo speaker. This simulated a robot issuing a command to another robot.

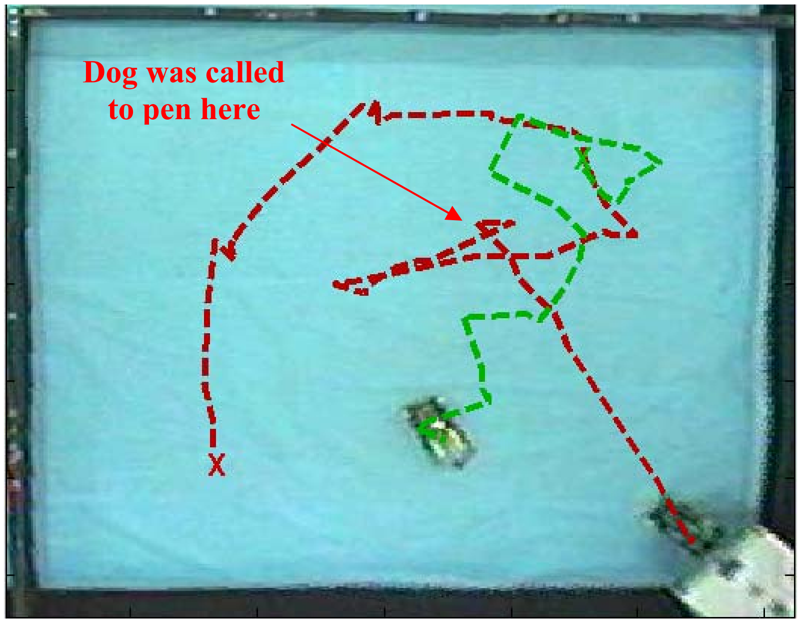


Figure 6.6 Results of microphone sensor test

During this experiment, the sheepdog would attempt to herd the sheep into the pen. If the microphone sensor detected a tone of 2.8 kHz, the sheepdog was to stop herding the sheep and return to the pen. Figure 6.6 shows the results of one trial. The path taken by the sheepdog is shown in red, and the path taken by the sheep is in green. The starting positions of each robot are marked with an X. For every trial, the sheepdog was successful in recognizing the tone and returning to the pen. However, there was one problem that occurred during this experiment. As the experiment progressed, the frequency that the microphone sensor would detect decreased. As the batteries used to power the circuit wore down, the frequency decreased. This resulted in constant recalibration of the robot, but not in system failure.

Section 6.4 – Assisting the Sheepdog

The final set of experiments was conducted with all three robots in the field. Again, the task of the sheepdog was to herd the sheep into the pen. However, this time, the helper dog was used to assist the sheepdog if needed. The helper dog would wander around the field until the microphone sensor detected a tone. Once the tone was detected, the helper dog would move to a position where the sheep was between itself and the pen. This action was intended to help the sheepdog funnel the sheep towards the pen. Unfortunately, the helper dog only hindered the mission. Instead of stopping when it had reached the desired position, it would continue to move in circles around the sheep. Since it did not have enough sensor ports to have a rotating light sensor, both light sensors were fixed. In this configuration, the only way the helper dog knew the sheep was between itself and the pen was to have one light sensor detect the pen and the other detect the sheep at the same time. Most of the time, the helper dog would be in the desired position, but it would only detect the sheep and not the pen.

The helper dog would also become caught on the sheepdog during the trials. There was no available sensor port on the helper dog to use for detecting the sheepdog. As it would move around the sheep, it would often run into the sheepdog and cause the test to fail.

Another problem that was encountered was that the sheepdog would sometimes get confused and think the helper dog was the sheep. The light sensor on the sheepdog used to detect the infrared LED array on the sheep would also detect the light on the helper dog. It would eventually reach a point where the light sensor could not "see" the light on the helper dog and would look for the sheep. Interestingly, in one of the trials, the sheepdog managed to herd the sheep into the pen despite the disruptions caused by the helper dog. Despite this perceived success, the helper dog did not provide any aid to the sheepdog.

In an attempt to prevent some of the problems encountered in the previous tests, the sheepdog was also equipped with a microphone sensor. If the sheepdog had problems, it was commanded to stop. Next, the helper dog was called on to provide support. Once the helper dog had positioned itself as desired, the sheepdog would be commanded to resume herding the sheep into the pen. The addition of the microphone sensor to the sheepdog did not really improve the performance. Both the sheepdog and helper dog continued to interfere with each other. However, the overall amount of times the two were stuck on each other was reduced. In addition, the sheepdog continued to confuse the helper dog with the sheep and attempted to herd it into the pen. Despite the poor performance of the robots, this experiment showed that communicating to the LEGO® robots with audible sound can be achieved with great success.

Chapter 7 – Conclusions and Future Research

Several conclusions about the feasibility of using LEGO® Mindstorms™ to develop autonomous robots have been developed from the research presented in previous chapters. The desirable aspects of this platform, as well as its deficiencies, are summarized below. Finally, recommendations on ways to improve the platform and future research using the LEGO® Mindstorms™ are offered.

Section 7.1 – Concluding Remarks

An extensive investigation was conducted to determine the possibility of using the LEGO® Mindstorms™ platform for mobile robot development. This research resulted in the emergence of several limitations. The biggest obstacle to overcome when using LEGO® Mindstorms™ is the limited number of sensor input ports. Only three sensors can be used on each robot at any one time. If more sensors are needed, a multiplexer or similar device must be added. However, the use of a multiplexer does not solve the problem completely. It just reduces it. Even with a multiplexer, only three sensors can be used at once. This research showed that effective communication between robots could not occur without more sensors. Those sensors needed to complete the task of herding the sheep into the pen used all of the input ports. Without having the ability for robot-to-robot communication, having the helper dog provide assistance to the sheepdog is not possible.

Another major problem experienced was the significant reduction in battery power that occurred during operation. The largest consumers of battery power are the motors. With three motors (on the sheep and sheepdog) running continuously during the mission, there is a large strain placed on the batteries. Additionally, all of the sensors used on each robot require power to operate. This, combined with the continuous use of the motors, resulted in large battery consumption causing a rapid discharge. As the battery power decreased, the outputs of the light sensors would change without the light conditions changing. Eventually, the sensors would output incorrect readings and the task would fail.

Even with these major drawbacks, LEGO® Mindstorms™ prove to serve as an adequate platform for developing inexpensive mobile robots. One of the major research areas in the field of robotics deals with designing inexpensive mobile robots with limited capabilities that, when used in conjunction with other robots of the same type, can function the same as a single advanced robot. The results of the experiments conducted with the sheep and sheepdog prove that LEGO® Mindstorms™ robots can perform fairly complex tasks with limited sensor capabilities. By having the ability to use advanced programming languages, complex tasks can be performed using the LEGO® robots. Also, the LEGO® Mindstorms™ platform allows for moderately advanced sensors, such as the microphone sensor, to be integrated into the system if needed. The research showed that detecting sound with the microphone sensor could control the robots. While the sheepdog was performing the complex task of herding the sheep into the pen, it could be instructed to stop and execute another task. It could then be commanded to resume the task of herding the sheep. Being able to use these types of sensors with the LEGO® robots also results in the possibility of completing complex tasks.

Section 7.2 – Future Research

In the immediate future, two items can be changed to improve overall system performance. First, the tone decoder circuit should be fixed. To operate at the desired center frequency, the layout of the circuit board should be changed to the layout shown in Figure

5.16. This change was not made since the sensor functioned as desired, although at a higher frequency. Fixing the circuit could potentially eliminate the need for constant recalibration during operation. Also, the identifying infrared LED array should be pulsed using a standard 555-timer circuit. This would help conserve the 9V battery used to provide power to the circuit.

In order to create a cooperative team using the LEGO® Mindstorms™, a communication link between the robots must be established. At the present time, this is not possible. First, a tone must be generated by one robot that can be heard by the other. Since the RCX cannot produce a loud enough tone, additional hardware will have to be developed. Next, the limited sensory capabilities must be overcome. If the same task of herding the sheep into a pen is desired, more sensors will need to be added in order for communication between robots to occur. However, three sensor inputs are probably sufficient for robot-to-robot communication, assuming the robots can generate a tone and the task required to be completed is much simpler. One method that should be investigated is using multiple RCX units for a single robot. This would not only increase the sensor capabilities, it would also double the output ports. These could be used to power any custom hardware used. Finally, instead of a simulated mother robot, an actual robot should be used. Incorporating the LEGO® robots into the CRIM's EvBot robot colony [18] [33] would allow for research in the area of marsupial robot colonies to be conducted.

Chapter 8 – References

[1] "Active Sensor Multiplexer." Mindsensors Robotics. Retrieved 4 February 2003. <http://www.mindsensors.com/active_mux.htm>

[2] Angeli, Frank. Light Sensor disassembly report. Updated 16 September 1988. Retrieved 1 February 2003. < http://www.crynwr.com/LEGO-robotics/light-sensor.html>

[3] Arkin, Ronald C. Behavior-Based Robotics. Cambridge: The MIT Press, 1998.

[4] Asama, Hajime, Akihiro Matsumato, and Yoshiki Ishida. "Design of an Autonomous and Distributed Robot System: ACTRESS." In Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems, 4-6 September 1989, pages 283-290.

[5] Baum, Dave. Dave Baum's Definitive Guide to LEGO Mindstorms. New York: Springer-Verlag New York, Inc., 2000.

[6] Baum, Dave. NQC Programmer's Guide, Version 2.5 a4. Released 27 January 2003. Retrieved 31 January 2003. <http://www.baumfamily.org/nqc/beta/NQC_Guide.pdf>

[7] Berger, Daniel. Advanced LEGO Mindstorms Programming in Visual C++. Max Plank Institute for Biological Cybernetics. Updated 20 November 2002. Retrieved 23 January 2002. <http://www.kyb.tuebingen.mpg.de/bu/people/berger/Mindstorms.html>

[8] Braly, J. Chris, Kyle Luthy, Jason Stevens, Carey Merrit, Mohammed Faza, and Kyle Pruitt. "Development of a Low-Cost, Robust Platform for Building Mobile Robots." Unpublished CRIM project report. North Carolina State University, 2002.

[9] Beni, Gerardo. "The Concept of Cellular Robotic System." In Proceedings of the IEEE International Symposium on Intelligent Control, Arlington, VA, 24-26 August 1998, pages 57-62.

[10] Casper, Jennifer L. "Human-Robot Interactions During the Robot-Assisted Urban Search and Rescue Response at the World Trade Center." Masters Thesis, Department of Computer Science and Engineering, University of South Florida, April 2002.

[11] Casper, Jennifer L. and Robin R. Murphy. "Workflow Study on Human-Robot Interaction in USAR." In Proceedings of IEEE International Conference on Robotics and Automation, Washington, DC, May 2002, pages 1997-2003.

[12] Chaimowicz, Luiz, Mário F. M. Campos, and Vijay Kumar. "Dynamic Role Assignment for Cooperative Robots." In Proceedings of the IEEE International Conference on Robotics and Automation, Washington, DC, May 2002, pages 293-298.

[13] Dario, P., F. Ribechini, V. Genovese, and G. Sandini. "Instinctive Behaviors and Personalities in Societies of Cellular Robots." In Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento, CA, 9-11 April 1991, pages 1927-1932.

[14] Davids, Angela. "Urban Search and Rescue Robots: From Tragedy to Technology." IEEE Intelligent Systems, Vol. 17, Iss. 2, March/April 2002, pages 81-83.

[15] Emery, Rosemary, Kevin Sikorski, and Tucker Balch. "Protocols for Collaboration, Coordination and Dynamic Role Assignment in a Robot Team." In Proceedings of the IEEE International Conference of Robotics and Automation, Washington, DC, May 2002, pages 3008-3015.

[16] Fagin, Barry. "An Ada Interface to LEGO Mindstorms". *Ada Letters Vol. 21 No. 2* (September 2000). Retrieved 23 January 2003. <http://www.faginfamily.net/barry/Papers/AdaLetters.htm>

[17] Fukuda, Toshio and Seiya Nakagawa. "Dynamically Reconfigurable Robotic System." In Proceedings of the IEEE International Conference of Robotics and Automation, Philadelphia, PA, 24-29 April 1988, pages 1581-1586.

[18] Galeotti, John. "The EvBot: A Small Autonomous Mobile Robot for the Study of Evolutionary Algorithms in Distributed Robotics." Master's Thesis. North Carolina State University, 2002.

[19] Gasperi, Michael. "Machina Speculatrix." Retrieved 12 February 2003. <http://www.plazaearth.com/usr/gasperi/walter.htm>

[20] Gasperi, Michael. Mindstorms RCX Sensor Input Page. Retrieved 29 January 2002. <http://www.plazaearth.com/usr/gasperi/LEGO.htm>

[21] Halme, Aarne, Peter Jakubik, Torsten Schönberg, and Mika Vianio. "The Concept of Robot Society and its Utilization." In Proceedings of the IEEE/Tsukuba International Workshop on Advanced Robotics, 8-9 November 1993, pages 29-35.

[22] Hanley, David and Sean Hearne. LEGO Robotics Course. Retrieved 23 January 2003. <http://emhain.wit.ie/~p98ac25/>

[23] Hempel, Ralph. pbForth Home Page. Retrieved 23 January 2003. <http://www.hempeldesigngroup.com/LEGO/pbForth/homePage.html>

[24] Hurbain, Philippe. LEGO Mindstorms compatible devices. Retrieved 1 February 2003. <http://www.philohome.com/sensors.htm>

[25] Iversen, Tortsen K., Kåre J. Kristofferson, Kim G. Larsen, and Morten Laursen. "Model-Checking Real-Time Control Programs: Verifying LEGO Mindstorms Systems Using UPPAAL." In Proceedings of Euromicro Conference on Real-Time Systems, Stockholm, Sweden, 19-21 June 2000, pages 147-155.

[26] Jennings, James S., Greg Whelan, and William F. Evans. "Cooperative Search and Rescue with a Team of Mobile Robots." In Proceedings of the International Conference on Advanced Robotics, Monterey, CA, 7-9 July 1997, pages 193-200.

[27] Kumar, Amruth N. "Using Robots in an Undergraduate Artificial Intelligence Course: An Experience Report." In Proceedings of the ASEE/IEEE Frontiers in Education Conference, Reno, NV, 10-13 October 2001, pages TD4-10-14.

[28] Lambert, Robert L. "A Talented Robot." Unpublished project report. University of Strathclyde, 1990.

[29] LeBouthillier, Arthur E. "W. Grey Walter and his Turtle Robots." The Robot Builder Vol. 11 Num. 5 (May 1999): Retrieved 7 February 2003. <http://www.csulb.edu/~wmartinz/rssc/newsletters/may99.pdf>

[30] "LogIT Sensors." DCP Microdevelopments. Retrieved 3 February 2003. <http://www.dcpmicro.com/LEGO/index.htm>

[31] Liu, Jiming and Jianbing Wu. Multi-agent robotic systems. Boca Raton: CRC Press, 2001.

[32] Luthy, K. A., J. C. Braly, L. S. Mattos, E. Grant, J. F. Muth, A. Seyam, A. Dahwan, and T. Ghosh. "Initial Development of a Portable Acoustic Array on a Large-Scale E-Textile Substrate." In Proceedings of the Materials Research Society Fall Symposium, Vol. 736, Boston, MA, 2002.

[33] Mattos, Leonardo. "The EvBot II: An Enhanced Evolutionary Robotics Platform Equipped with Integrated Sensing for Control." Master's Thesis (not published).

[34] Murphy, Robin R. Introduction to AI Robotics. Cambridge: The MIT Press, 2000.

[35] Murphy, Robin R. "Marsupial and shape-shifting robots for urban search and rescue." IEEE Intelligent Systems, Vol. 15, Iss. 2, March/April 2000, pages 14-19.

[36] Murphy, Robin R., Jennifer L. Casper, Jeff Hyams, Mark Micire, and Brian Minten. "Mobility and Sensing Demands in USAR." In Proceedings of the International Conference of the IEEE Industrial Electronics Society, Nagoya, Japan, 22-28 October 2000, pages 138-142.

[37] Murphy, Robin R., Michelle Ausmus, Magda Bugajska, Tanya Ellis, Tonia Johnson, Nia Kelley, Jodi Kiefer, and Lisa Pollock. "Marsupial-like Mobile Robot Societies." In Proceedings of ACM International Conference on Autonomous Agents, Seattle, WA, 1999, pages 364-365.

[38] New Japan Radio Co., Ltd. "FSK Demodulator/Tone Decoder (NJM2211M)." Datasheet. Retrieved 26 February 2003. < http://www.njr.co.jp/pdf/be/be06008.pdf>

[39] Panasonic USA. "Omnidirectional Electret Condenser Microphone Cartridge (WM-52BM)". Datasheet. Retrieved 26 February 2003. <http://rocky.digikey.com/WebLib/ Panasonic/Web%20data/WM-52B,54B%20Series.pdf>

[40] Parker, Lynne E. "ALLIANCE: An Architecture for Fault Tolerant Multirobot Cooperation." In IEEE Transactions on Robotics and Automation, Vol. 14 Iss. 2, April 1998, pages 220-240.

[41] Parker, Lynne E. "The Effect of Action Recognition and Robot Awareness in Cooperative Robotic Teams." In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Pittsburgh, PA, 5-9 August 1995, pages 212-219.

[42] Pereira, Guilherme A. S., Bruno S. Pimentel, Luiz Chaimowicz, and Mário F. M. Campos. "Coordination of Multiple Mobile Robots in an Object Carrying Task Using Implicit Communication." In Proceedings of the IEEE International Conference on Robotics and Automation, Washington, DC, May 2002, pages 281-286.

[43] PITSCO Online Store. Retrieved 3 February 2003. <http://www.pldstore.com>

[44] Proudfoot, Kekoa. RCX Internals. Retrieved 29 January 2002. <http://graphics.stanford.edu/~kekoa/rcx/>

[45] Russell, Andrew, David Theil, and Alan Mackay-Sim. "Sensing Odour Trails for Mobile Robot Navigation." In Proceedings of the IEEE International Conference on Robotics and Automation, San Diego, CA, 8-13 May 1994, pages 2672-2677.

[46] Rybski, Paul E., Nikolaos P. Papanikolopoulos, Sascha A. Stoeter, Donald G. Krantz, Kemal B. Yesin, Maria Gini, Richard Voyles, Dean F. Hougen, Brad Nelson, and Michael D. Erickson. "Enlisting Rangers and Scouts for Reconnaissance and Surveillance." IEEE Robotics and Automation Magazine, Vol. 7 Issue 4, December 2000, pages 14-24.

[47] Solorzano, Jose et al. leJOS: JAVA for the RCX. Updated 21 January 2003. Retrieved 23 January 2003. <http://lejos.sourceforge.net/>

[48] Texas Instruments, Inc. "General-Purpose Operational Amplifiers (UA741CD)." Datasheet. Copyright 2000. Retrieved 26 February 2003. < http://www-s.ti.com/sc/psheets/ slos094b/slos094b.pdf>

[49] Wallich, Paul. "Mindstorms™'s: Not Just a Kid's Toy." *IEEE Spectrum* September 2001: 52-57.

[50] Walter, W. Grey. The Living Brain. New York: W.W. Norton & Company, 1953.

[51] Ward, Mark. "Walter's World." New Scientist. Vol. 159 Issue 2144 (25 July 1998): p54.

[52] Yamamoto, Masaya, Tomonori Hashiyama, and Shigeru Okuma. "Reducing Computational Time on Evolution Under the Real Environment Using Fitness Estimation." In Proceedings of the Annual Conference of the IEEE Industrial Electronics Society, Nagoya, Japan, 22-28 October 2000, pages 2497-2500.

Appendix

Chapter 9 – Appendix

The following sections include photographs of the robot colony developed, the NQC source code used to control the individual robots, an explanation of how images were acquired during experimentation, and a comprehensive parts list.

Section 9.1 – LEGO® Robot Colony

This section will offer different views of each agent that comprises the robot colony. Following the photographs of the robots, pictures of the field and pen will be shown.

Section 9.1.1 – Sheepdog

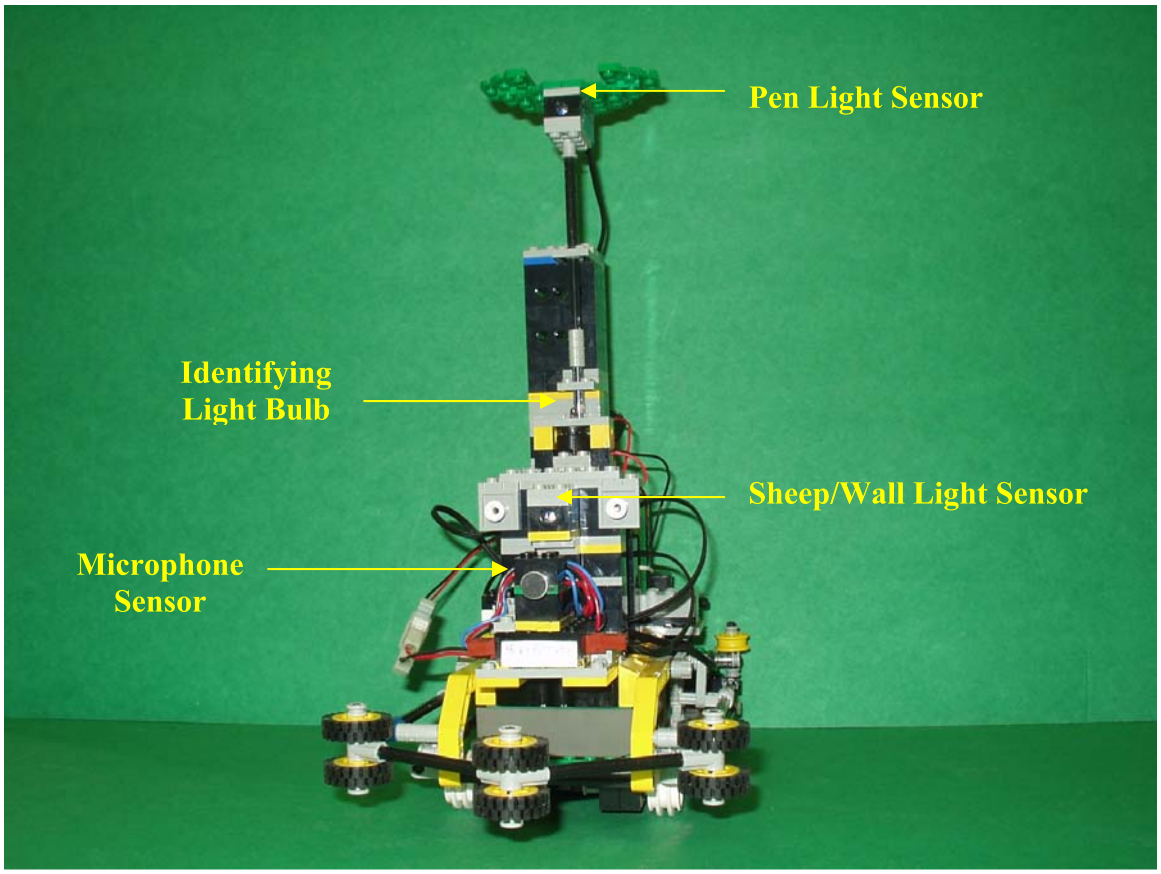


Figure 9.1 Front side of the sheepdog

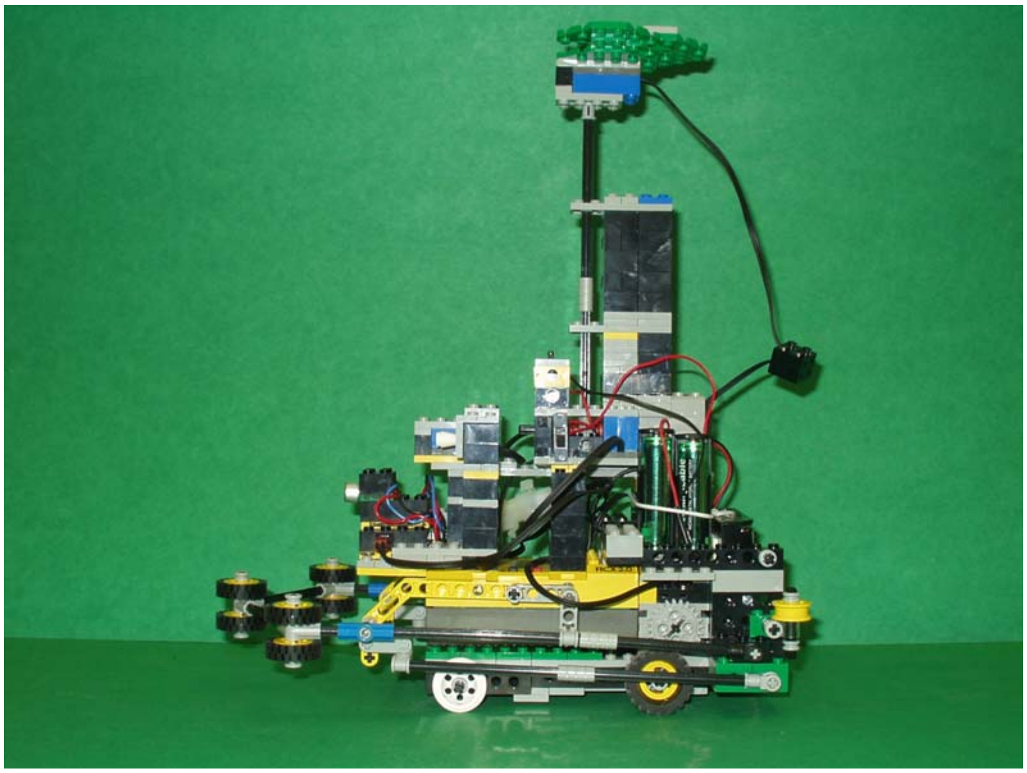


Figure 9.2 Left side of the sheepdog

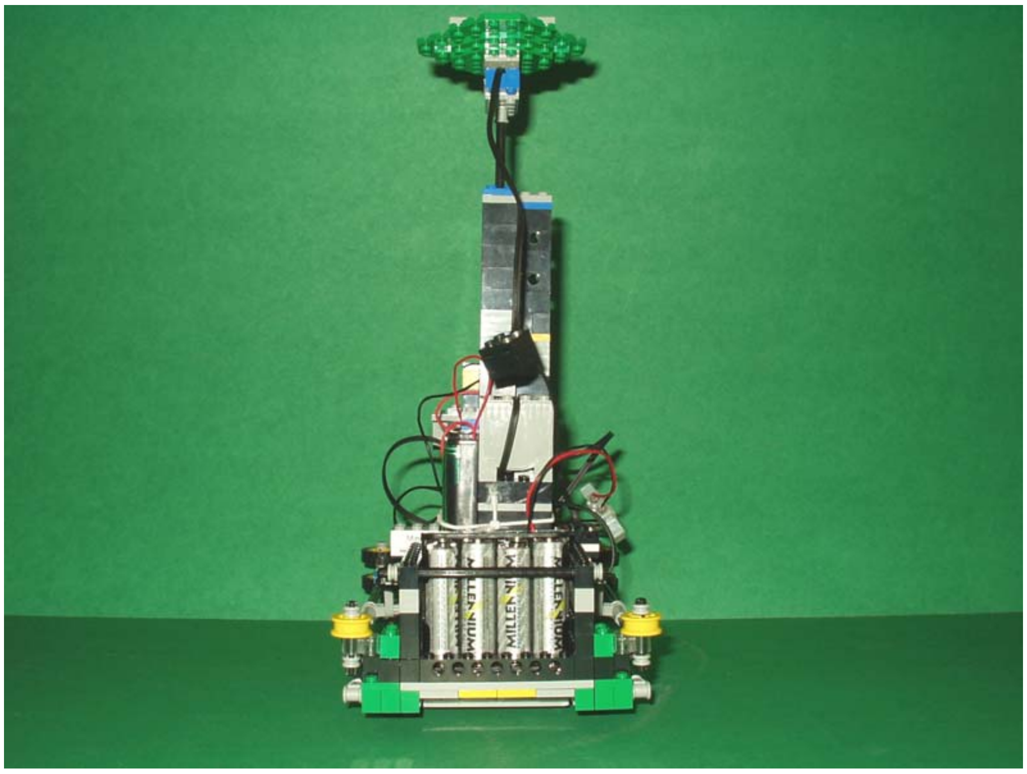


Figure 9.3 Backside of the sheepdog

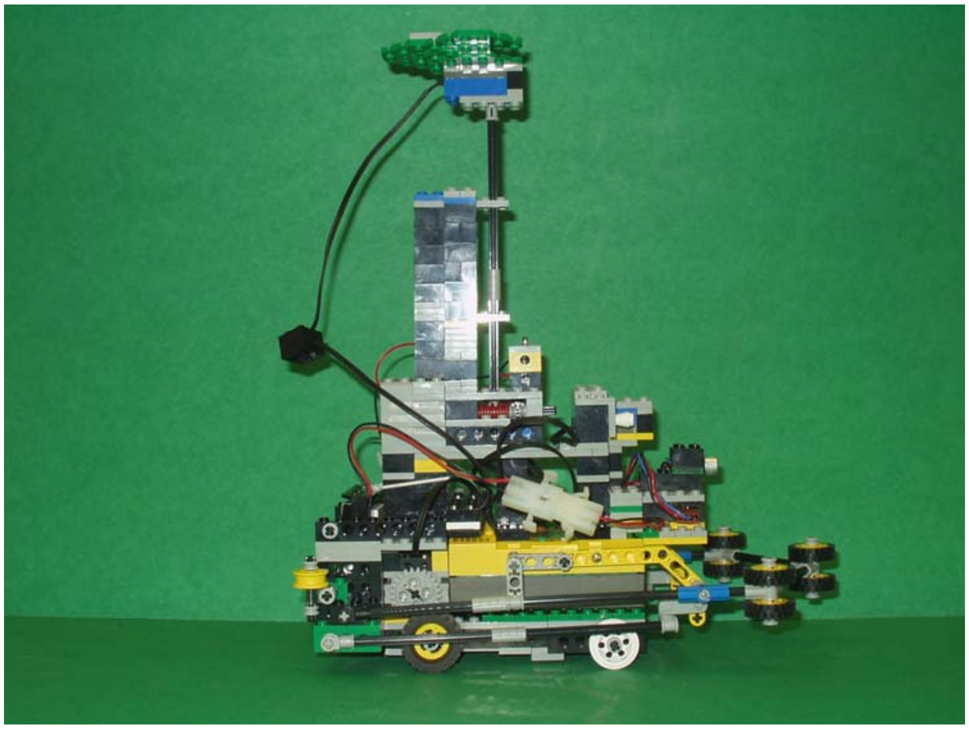


Figure 9.4 Right side of the sheepdog

Section 9.1.2 – Sheep

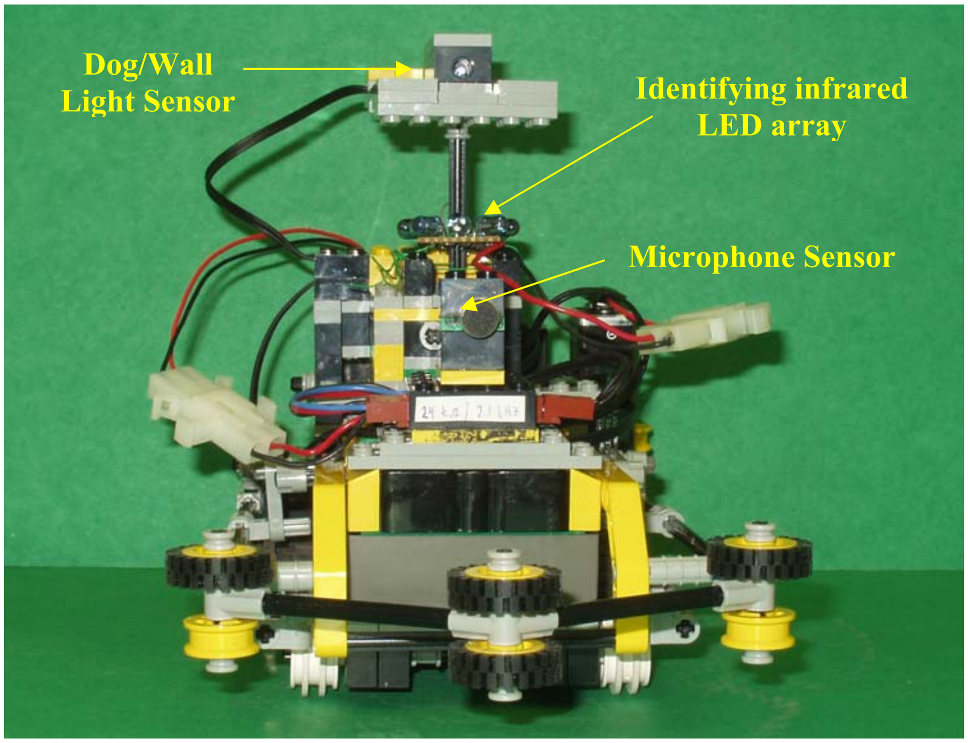


Figure 9.5 Front side of the sheep

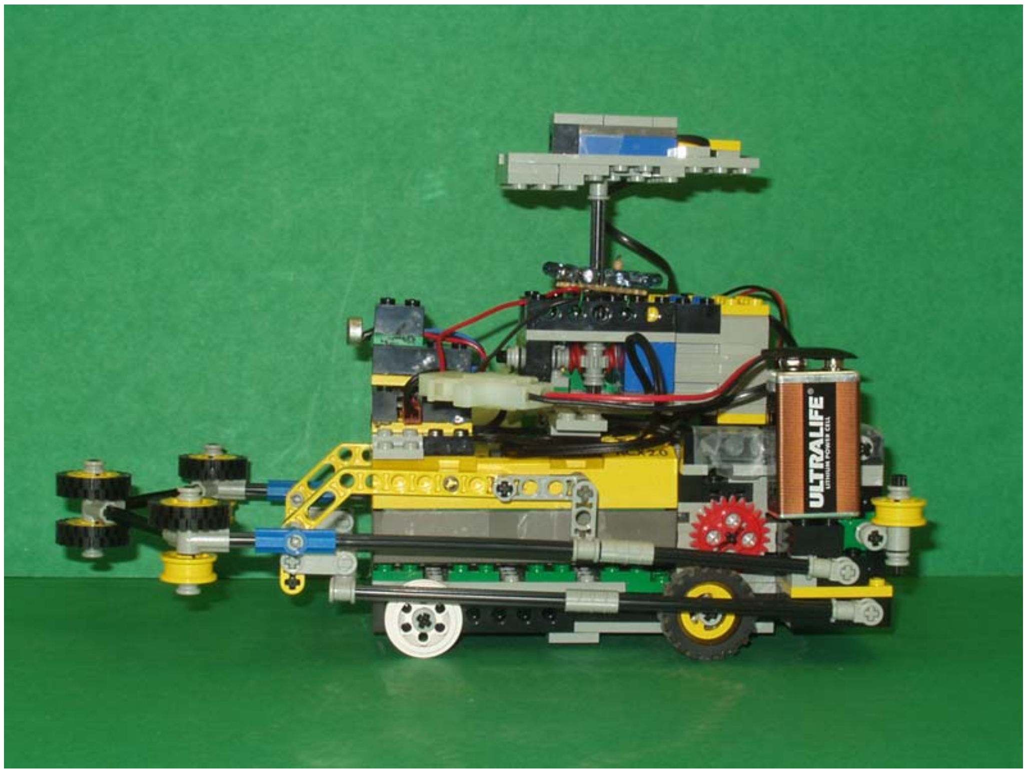


Figure 9.6 Left side of the sheep

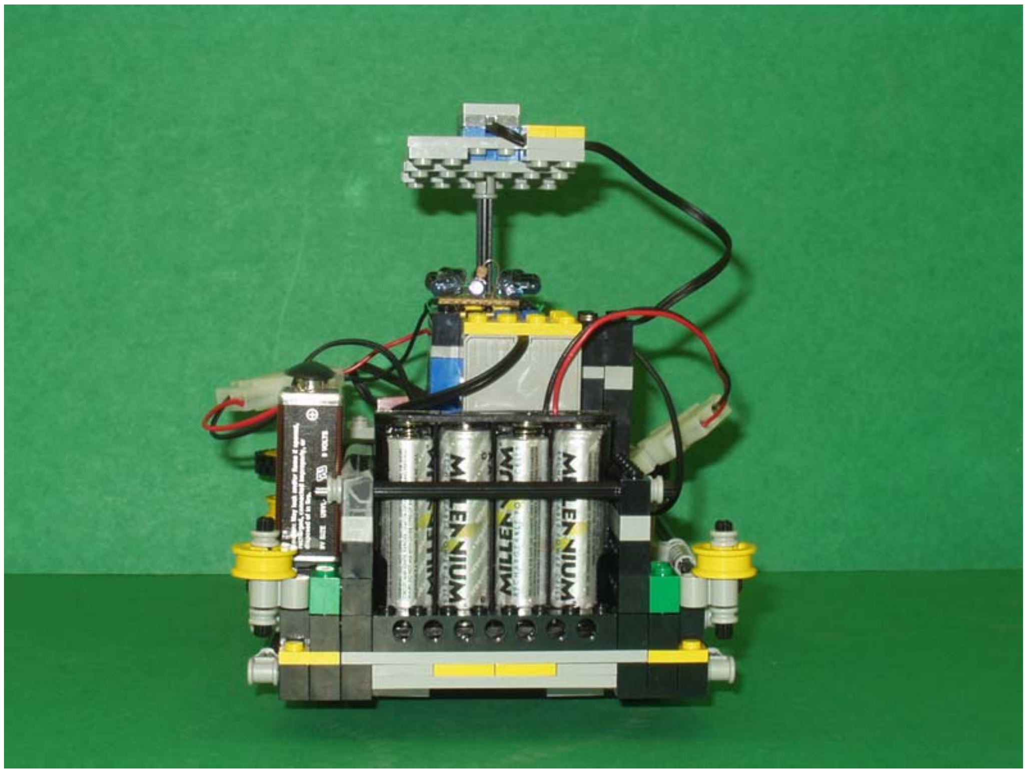


Figure 9.7 Backside of the sheep

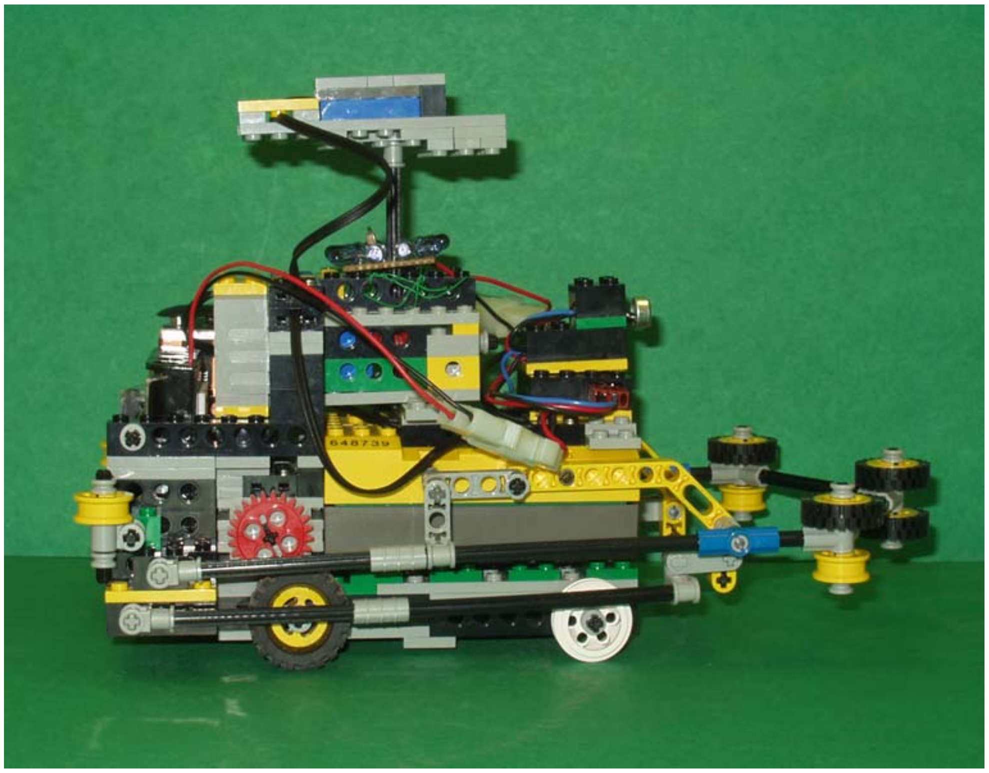


Figure 9.8 Right side of the sheep

Section 9.1.3 – Helper Dog

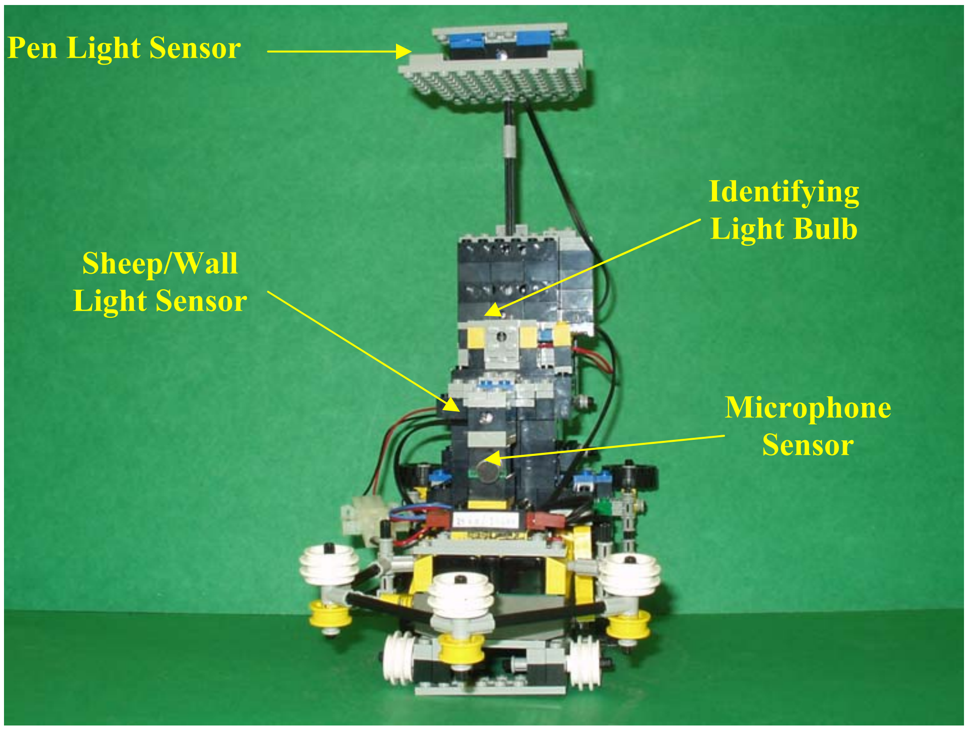


Figure 9.9 Front side of the helper dog

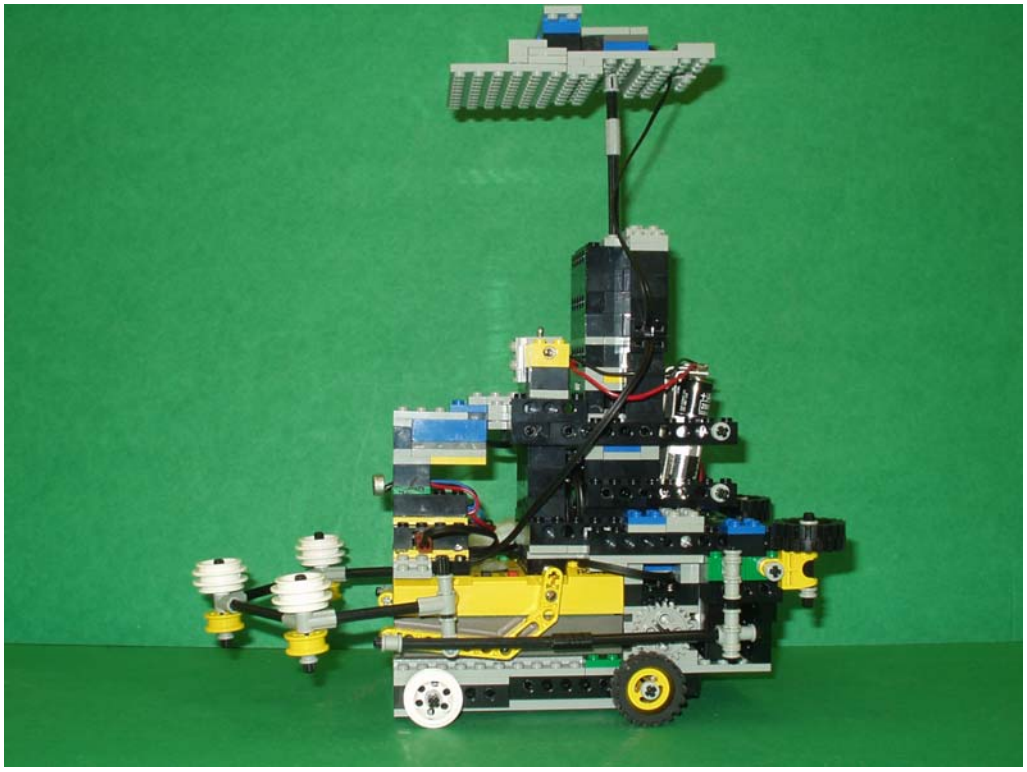


Figure 9.10 Left side of the helper dog

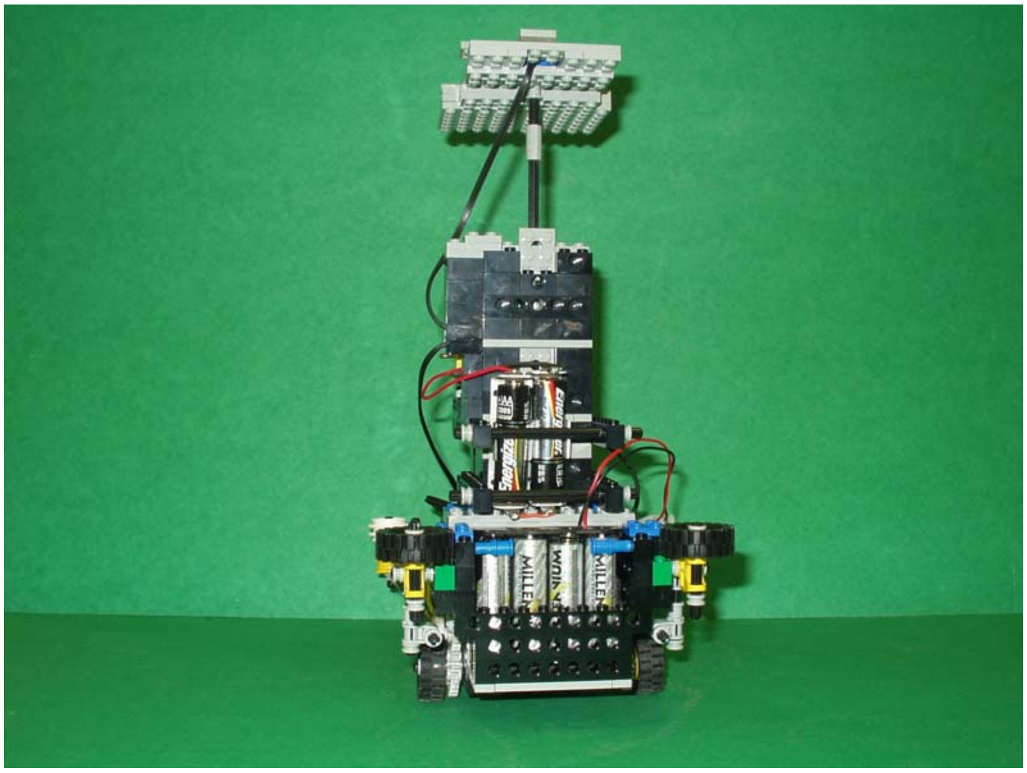


Figure 9.11 Backside of the helper dog

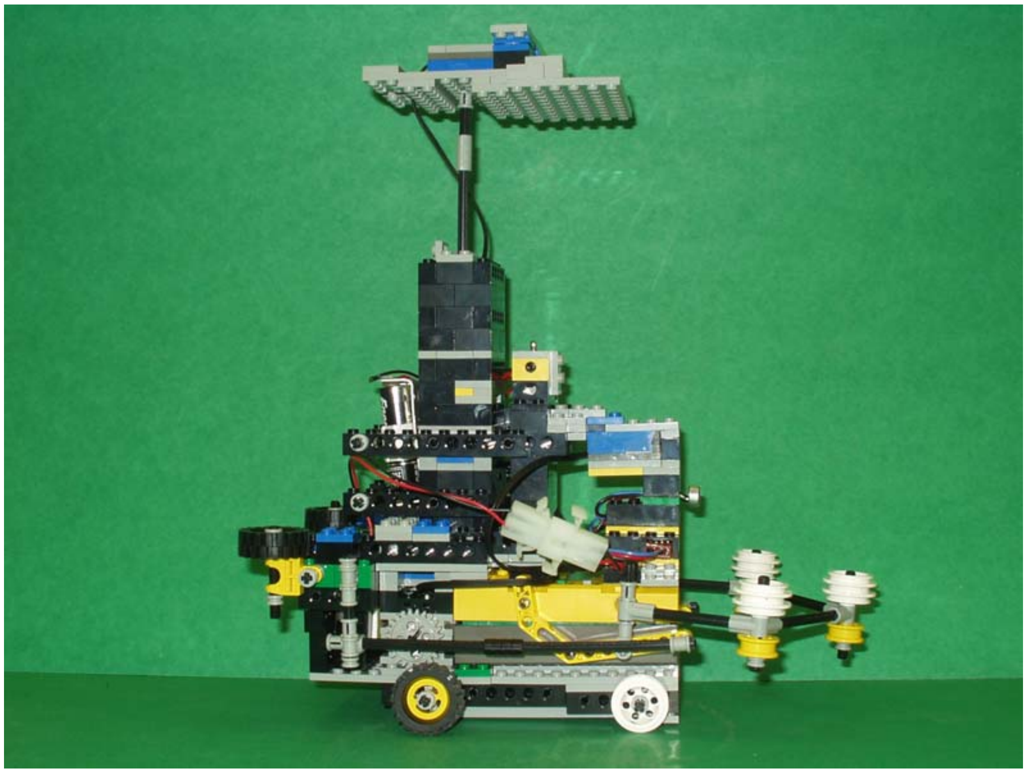


Figure 9.12 Right side of the helper dog

Section 9.1.4 – Field

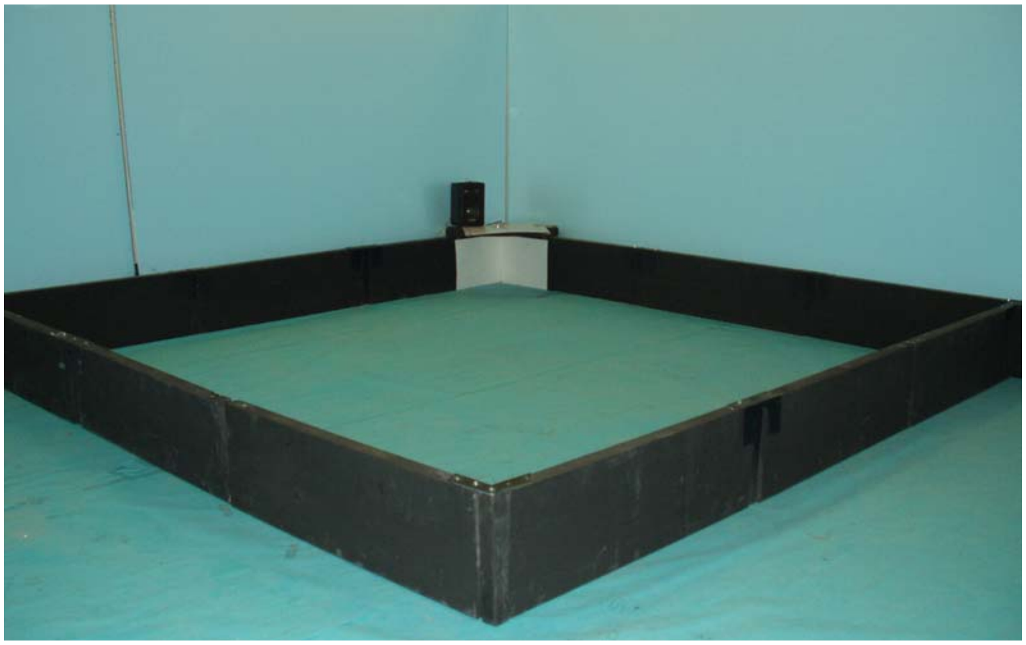


Figure 9.13 The 8' x 8' field used to conduct experiments

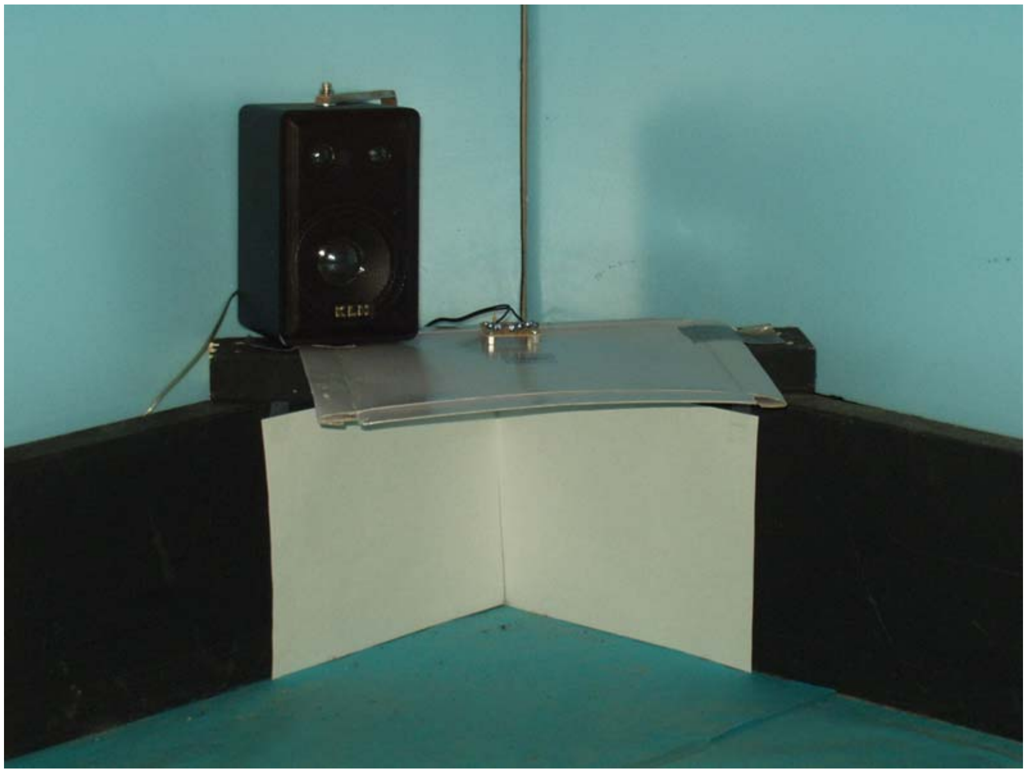


Figure 9.14 The pen used for experiments

Section 9.2 – NQC Source Code

This section includes the NQC source code used to control the sheepdog, sheep, and the helper dog.

Section 9.2.1 – Sheepdog NQC Source Code

```
#define LEFT OUT_A                  //define motor output A as LEFT motor
#define HEAD OUT_B                  //define motor output B as HEAD motor
#define RIGHT OUT_C                 //define motor output C as RIGHT motor
#define EAR SENSOR_1                //define sensor input 1 as EYE
#define ANGLE SENSOR_2              //define sensor input 2 as ANGLE
#define MUX SENSOR_3                //define sensor input 3 as MUX

int TURN = 7;                       //LEFT and RIGHT motor power for turning
int FWD = 7;                        //LEFT and RIGHT motor power for going forward
int PEN = 780;                      //threshold for detecting the pen
int SHEEP = 815;            //threshold for detecting the sheep from far away
int SHEEPCLOSE = 600;               //threshold for detecting the sheep up close
int WALL = 920;                     //threshold for detecting the wall
int LDIR = 1;                       //1 = FWD MOTION of the tread; 0 = REV MOTION of the tread;
int RDIR = 1;                       //used in task trackPen if the EYE loses the pen
int FOUND = 0;                      //flag used for detecting the pen;  see function init()
int SDIR = 0;       //1 = Left; 0 = Right; flag used to determine which way to spin when looking for the sheep
int direction;                      // -1 = right; 1 = left; this flag is used for scanning purposes
int distance;                       //distance from the dawggy to the sheep
int decapitation = 590;             //threshold for determing if the dog is too close to the pen
int sound = 50;                     //threshold for detecting sound
int HELP = 1;                                    //flag used to tell the sheepdog to stop and start
int angleLast = 0;                       //variable used in stuck task

//This task is used to corral the sheep towards the pen.
task corral()
 {
  while(1)
   {
    findSheep();
    getDistance();
    findPen();
    turn();
    go();
    Eye();
    Wait(30);

    if (MUX <= decapitation)        //If the dog is too close to the pen,
    {                                             //go in reverse for one second
     GoRev();
     OnFor(LEFT+RIGHT, 100);
    }
   }
 }

//This function will cause the dog to look for the sheep by spinning around
//until the NOSE locates the sheep.
void findSheep()
 {
  Nose();
  Wait(30);

  while(MUX > SHEEP)
   {
    if (SDIR == 1)                    //dog will spin left to look for the sheep
     {
      TurnLeft();
      On(LEFT + RIGHT);
     }
    else if (SDIR == 0)               //dog will spin right to look for the sheep
     {
      TurnRight();
      On(LEFT + RIGHT);
     }
   }
  Off(LEFT+RIGHT);
 }

//This function is used for finding the pen once the sheep has been located
//The EYE is used to look for the pen
void findPen()
 {
  Eye();
  Wait(30);

  while(!FOUND)
   {
   On(HEAD);

   if (MUX < PEN)
   {
    FOUND = 1;
    Off(HEAD);
   }

   if ((ANGLE <= -66) && (direction == -1) && (!FOUND))
   {
    OnRev(HEAD);
    direction = 1;
   }
  if ((ANGLE >= 69) && (direction == 1) && (!FOUND))
   {
    OnFwd(HEAD);
    direction = -1;
   }
  }
 FOUND = 0;
 }

//Determines the distance from the dog to the sheep
void getDistance()
 {
 Nose();
 Wait(30);

 if ((MUX >= 802) && (MUX < SHEEP))
  distance = 13;
 else if ((MUX >= 780) && (MUX < 802))
  distance = 11;
 else if ((MUX >= 760) && (MUX < 780))
  distance = 10;
 else if ((MUX >= 743) && (MUX < 760))
  distance = 8;
 else if ((MUX >= 735) && (MUX < 753))
  distance = 6;
 else if ((MUX >= 705) && (MUX < 735))
  distance = 5;
 else if ((MUX >= 687) && (MUX < 705))
  distance = 4;
 else if ((MUX >= 655) && (MUX < 687))
  distance = 3;
 else if ((MUX >= 580) && (MUX < 655))
  distance = 2;
 else if ((MUX >= 440) && (MUX < 580))
  distance = 1;
 else distance = 0;
 }

//This function turns the dog a specific direction depending on
//its position relative to the pen before moving towards the sheep
void turn()
 {
 int timeAngle = 30;
 if (ANGLE > 5)
   {
    TurnLeft();
    OnFor(LEFT + RIGHT, timeAngle);
    SDIR = 0;
    }
 else if (ANGLE < -5)
   {
    TurnRight();
    OnFor(LEFT + RIGHT, timeAngle);
    SDIR = 1;
   }
 }

//This function moves the sheep dog towards the sheep
void go()
 {
 Nose();
 Wait(30);
 GoFwd();
 start sheepDistance;
 OnFor(LEFT + RIGHT, 20*distance);
 }

//This task is used to prevent the dog from running into the sheep
//If the dog is too close to the sheep, it will stop
task sheepDistance()
```

```
 {
 while(1)
  {
  if (MUX <= SHEEPCLOSE)
   {
   Off(LEFT+RIGHT);
   stop sheepDistance;
   }
  }
 }

//This function sets up the motors to turn right
void TurnRight()
 {
 SetDirection(LEFT, OUT_REV);
 SetDirection(RIGHT, OUT_FWD);
 SetPower(LEFT + RIGHT, TURN);
 LDIR = 1;
 RDIR = 0;
 }

//This function sets up the motors to turn left
void TurnLeft()
 {
 SetDirection(LEFT, OUT_FWD);
 SetDirection(RIGHT, OUT_REV);
 SetPower(LEFT + RIGHT, TURN);
 LDIR = 0;
 RDIR = 1;
 }

//This function sets up the motors to move forward
void GoFwd()
 {
 SetDirection(RIGHT, OUT_REV);
 SetDirection(LEFT, OUT_REV);
 SetPower(LEFT + RIGHT, FWD);
 LDIR = 1;
 RDIR = 1;
 }

//This function sets up the motors to move in reverse
void GoRev()
 {
 SetDirection(RIGHT, OUT_FWD);
 SetDirection(LEFT, OUT_FWD);
 SetPower(LEFT + RIGHT, FWD);
 LDIR = 0;
 RDIR = 0;
 }

//Changes the MUX to Channel 1 to detect the pen
void Eye()
 {
 SetSensorType(MUX,SENSOR_TYPE_TOUCH);
 Wait(2);
 SetSensorType(MUX,SENSOR_TYPE_LIGHT);
 SetSensorMode(MUX,SENSOR_MODE_RAW);
 }

//Changes the MUX to Channel 2 to detect the sheep
void Nose()
 {
 SetSensorType(MUX,SENSOR_TYPE_TOUCH);
 Wait(2);
 SetSensorType(MUX,SENSOR_TYPE_LIGHT);
 Wait(2);
 SetSensorType(MUX,SENSOR_TYPE_TOUCH);
 Wait(2);
 SetSensorType(MUX,SENSOR_TYPE_LIGHT);
```

```
  SetSensorMode(MUX,SENSOR_MODE_RAW);
 }

//This task will cause the dog to back up and turn around if it gets
//too close to a wall.  The NOSE is used to detect the wall
task avoidWall()
 {
  SetPower(LEFT + RIGHT, TURN);
  while(true)
  {
    while (MUX >= WALL)
		{
		    stop corral;
		    SetDirection(LEFT + RIGHT, OUT_FWD);
		    OnFor(LEFT + RIGHT, 50);
		    SetDirection(LEFT, OUT_REV);
		    SetDirection(RIGHT, OUT_FWD);
		    OnFor(LEFT + RIGHT, 50);
		    Nose();
		    Wait(30);
		    SDIR = 0;
		    start corral;
		}
  }
 }

//This task prevents the dog from getting stuck often.  If the rotation sensor does not detect movement in
//20 seconds, the sheepdog will move in reverse
task stuck()
 {
  while(true)
  {
   angleLast = ANGLE;
   Wait(2000);
   if (ANGLE == angleLast)
   {
    PlayTone(600,100);
    stop corral;
    stop avoidWall;
    Off(LEFT + RIGHT);
    GoRev();
    OnFor(LEFT + RIGHT, 200);
    OnFor(HEAD, 10);

    start corral;
    start avoidWall;
   }
  }
 }

task main()
 {
  direction = -1;
  SetSensor(EAR, SENSOR_LIGHT);                    //Defines EAR as a light sensor
  SetSensorMode(EAR, SENSOR_MODE_RAW);                    //Sensor outputs RAW (0-1023) values
  SetSensor(ANGLE, SENSOR_ROTATION);                     //Defines ANGLE as a rotation sensor
  SetDirection(HEAD, OUT_FWD);
  SetPower(LEFT + RIGHT, TURN);

  Wait(1000);

  start corral;
  start avoidWall;
  start stuck;

  while(true)                    //corral sheep unless a sound is detected
  {
   if ((EAR < sound) && (HELP == 1))        //If sound is detected, stop
   {
    stop corral;
   stop stuck;
   stop avoidWall;
   Off(LEFT + RIGHT + HEAD);
   HELP = 0;
  }

  if ((EAR < sound) && (HELP == 0))        //Resume corralling the sheep if a sound is detected
  {
   start corral;
   start stuck;
   start avoidWall;
   HELP = 1;
  }
 }
}
```

Section 9.2.2 – Sheep NQC Source Code

```
#define ANGLE SENSOR_1                    //Define sensor input 1 as ANGLE (rotation sensor)
#define EAR SENSOR_2                      //Define sensor input 2 as EAR (mic)
#define EYE SENSOR_3                      //Define sensor input 3 as EYE (light sensor)
#define RIGHT OUT_C                       //Define motor output C as LEFT motor
#define HEAD OUT_B                        //Define motor output B as HEAD motor
#define LEFT OUT_A                        //Define motor output A as RIGHT motor

int time = 0;                             //variable used in wander()
int black = 930;                          //threshold for detecting a wall
int detect = 0;                           //flag used when sheep sees either the dog or wall
int dog =710;                             //threshold for detecting the dog
int run = 4;                              //motor power for when sheep is running from the dog or wall
int fwd = 2;                              //motor power for when sheep is moving forward while grazing
int turn = 4;                             //motor power for when sheep is turning while grazing
int sound = 50;                           //threshold for detecting sound

//For this task, the sheep is continuously scanning and looking for the dog or a wall.
//When it sees either of the two, it will react accordingly.
task scan()
 {
  while(true)
  {
   On(HEAD);
   while((ANGLE > -68) && (ANGLE <= 0))   //scans counterclockwise for 180 degrees
   {
    OnFwd(HEAD);
    if((EYE >= black) || (EYE <= dog))
     detect = 1;
   }
   Off(HEAD);
   Wait(20);
   while (ANGLE < 0)   //scan clockwise -- returns to zero degree position
   {
    OnRev(HEAD);
    if((EYE >= black) || (EYE <= dog))
     detect = 1;
   }
   Off(HEAD);
   Wait(20);
   while((ANGLE > 0) && (ANGLE < 68))  //scans clockwise for 180 degrees
   {
    OnRev(HEAD);
    if((EYE >= black) || (EYE <= dog))
     detect = 1;
   }
   Off(HEAD);
   Wait(20);
   while(ANGLE > 0)  //returns EYE to zero degree position
   {
    OnFwd(HEAD);
    if((EYE >= black) || (EYE <= dog))
     detect = 1;
   }
   Off(HEAD);
   Wait(20);
  }
 }

//This funciton is called when a wall detected.  The sheep will move away from the wall.
void wall()
 {
  SetPower(LEFT + RIGHT, run);
  while (EYE >= black)
  {
   Off(HEAD);
   if ((ANGLE < -3) && (ANGLE >= -60))   //robot turns RIGHT
   {
    SetDirection(LEFT, OUT_REV);
    SetDirection(RIGHT, OUT_FWD);
    OnFor(LEFT + RIGHT, 75);
    SetDirection(RIGHT, OUT_REV);
    OnFor(LEFT + RIGHT, 50);
   }
   else if ((ANGLE >= -3) && (ANGLE <= 3))  //robot moves backward and spin away from wall
   {
    SetDirection(LEFT + RIGHT, OUT_FWD);
    OnFor(LEFT + RIGHT, 75);
    SetDirection(LEFT, OUT_REV);
    SetDirection(RIGHT, OUT_FWD);
    OnFor(LEFT + RIGHT, 75);
   }
   else if ((ANGLE > 3) && (ANGLE <= 60))   //robot turns LEFT
   {
    SetDirection(LEFT, OUT_FWD);
    SetDirection(RIGHT, OUT_REV);
    OnFor(LEFT + RIGHT, 75);
    SetDirection(LEFT, OUT_REV);
    OnFor(LEFT + RIGHT, 50);
   }
   else if ((ANGLE < -60) || (ANGLE > 60))  //robot moves forward
   {
    SetDirection(LEFT + RIGHT, OUT_REV);
    OnFor(LEFT + RIGHT, 75);
   }
  }
  start scan;
  start wander;
 }

//This function is called when the sheep sees the dog.  The sheep will run away from the dog.
void panic()
 {
  SetPower(LEFT + RIGHT, run);
  while (EYE <= dog)
  {
   Off(HEAD);
   if ((ANGLE < -10) && (ANGLE > -25))   //robot turns RIGHT
   {
    SetDirection(LEFT, OUT_REV);
    SetDirection(RIGHT, OUT_FWD);
    OnFor(LEFT + RIGHT, 80);
    SetDirection(RIGHT, OUT_REV);
    OnFor(LEFT + RIGHT, 60);
   }
   else if ((ANGLE <= -25) && (ANGLE >= -35))   //robot turns RIGHT
   {
    SetDirection(LEFT, OUT_REV);
    SetDirection(RIGHT, OUT_FWD);
    OnFor(LEFT + RIGHT, 70);
```

```
     SetDirection(RIGHT, OUT_REV);
     OnFor(LEFT + RIGHT, 60);
    }
    else if ((ANGLE < -35) && (ANGLE >= -60))   //robot turns RIGHT
    {
     SetDirection(LEFT, OUT_REV);
     SetDirection(RIGHT, OUT_FWD);
     OnFor(LEFT + RIGHT, 50);
     SetDirection(RIGHT, OUT_REV);
     OnFor(LEFT + RIGHT, 60);
    }
    else if ((ANGLE >= -10) && (ANGLE <= 10))  //robot moves backward and spin away from dog and then runs
    {
     SetDirection(LEFT + RIGHT, OUT_FWD);
     OnFor(LEFT + RIGHT, 50);
     SetDirection(LEFT, OUT_REV);
     SetDirection(RIGHT, OUT_FWD);
     OnFor(LEFT + RIGHT, 100);
     SetDirection(RIGHT, OUT_REV);
     OnFor(LEFT + RIGHT, 60);
    }
    else if ((ANGLE > 10) && (ANGLE < 25))   //robot turns LEFT
    {
     SetDirection(LEFT, OUT_FWD);
     SetDirection(RIGHT, OUT_REV);
     OnFor(LEFT + RIGHT, 80);
     SetDirection(LEFT, OUT_REV);
     OnFor(LEFT + RIGHT, 60);
    }
    else if ((ANGLE >= 25) && (ANGLE <= 35))   //robot turns RIGHT
    {
     SetDirection(LEFT, OUT_FWD);
     SetDirection(RIGHT, OUT_REV);
     OnFor(LEFT + RIGHT, 70);
     SetDirection(LEFT, OUT_REV);
     OnFor(LEFT + RIGHT, 60);
    }
    else if ((ANGLE > 35) && (ANGLE <= 60))   //robot turns RIGHT
    {
     SetDirection(LEFT, OUT_FWD);
     SetDirection(RIGHT, OUT_REV);
     OnFor(LEFT + RIGHT, 50);
     SetDirection(LEFT, OUT_REV);
     OnFor(LEFT + RIGHT, 60);
    }
    else if ((ANGLE < -60) || (ANGLE > 60))  //robot moves forward
    {
     SetDirection(LEFT + RIGHT, OUT_REV);
     OnFor(LEFT + RIGHT, 40);
    }
   }
   Wait(100);
   start scan;
   start wander;
  }

//This task simulates a sheep grazing.
task wander()
  {
   while(true)
   {
    SetDirection(LEFT + RIGHT, OUT_REV);
    SetPower(LEFT + RIGHT, fwd);
    On(RIGHT + LEFT);
    time = Random(100);
    Wait(time);
    Off(LEFT + RIGHT);
    time = Random(250);
    Wait(time);
    if((time%2) == 0)
    {
     SetPower(LEFT+RIGHT,turn);
     OnRev(LEFT);
     OnFwd(RIGHT);
    }
    else
    {
     SetPower(LEFT+RIGHT,turn);
     OnRev(RIGHT);
     OnFwd(LEFT);
    }
    time = Random(75);
    Wait(time);
    Off(LEFT + RIGHT);
    Wait(1500);
   }
  }

task main()
  {
   SetSensorType(EYE, SENSOR_TYPE_LIGHT);        //Defines EYE as a light sensor
   SetSensorMode(EYE, SENSOR_MODE_RAW);
   SetSensorType(EAR, SENSOR_TYPE_LIGHT);        //Defines EAR as a light sensor
   SetSensorMode(EAR, SENSOR_MODE_RAW);
   SetSensor(ANGLE, SENSOR_ROTATION);                          //Defines ANGLE as a rotation sensor
   SetPower(HEAD, 1);

   SetPower(LEFT, fwd);
   SetPower(RIGHT, fwd);

   Wait(1000);

   while (true)
   {
     start scan;
     start wander;

     while(true)
     {
     if (detect == 1)
     {
      stop wander;
      Off(LEFT + RIGHT);

      if (EYE >= black)
      {
        PlayTone(500, 50);
        wall();
      }
      else if ( EYE <= dog)
      {
        stop scan;
        Off(HEAD);
        PlayTone(3000, 50);
               panic();
      }
     detect = 0;
     start scan;
     }
     }
    }
  }
```

Section 9.2.3 – Helper Dog NQC Source Code

```
#define PEN_EYE SENSOR_1            //Define sensor input 1 as PEN_EYE (light sensor)
#define EAR SENSOR_2                //Define sensor input 2 as EAR (mic)
#define EYE SENSOR_3                //Define sensor input 3 as EYE (light sensor)
#define LEFT OUT_C                  //Define motor output C as LEFT motor
#define LIGHT OUT_B                 //Define motor output B as LIGHT (used to power lightbulb)
#define RIGHT OUT_A                 //Define motor output A as RIGHT motor

int black = 889;                    //threshold for detecting a wall
int detect = 0;                     //flag used when helper dog sees either the sheep or wall
int run = 6;                        //motor power for when helper dog is helping the dog
int turn = 5;                       //motor power for when helper dog is turning while monitoring
int sound = 50;                     //threshold for detecting sound
int pen = 780;                      //threshold for detecting the pen
int sheep = 815;                    //threshold for detecting the sheep
int sheepclose = 700;               //too close to sheep when under this value
int decapitation = 590;             //too close to the pen when under this value

//This funciton is called when a wall detected.  The helper dog will move away from the wall.
task avoidWall()
 {
  while (EYE >= black)
  {
    GoRev();
    OnFor(LEFT + RIGHT, 100);
    TurnRight();
    OnFor(LEFT + RIGHT, 100);
   }
 }

//This function sets up the motors to turn right
void TurnRight()
 {
  SetDirection(LEFT, OUT_REV);
  SetDirection(RIGHT, OUT_FWD);
  SetPower(LEFT + RIGHT, turn);
 }

//This function sets up the motors to turn left
void TurnLeft()
 {
  SetDirection(LEFT, OUT_FWD);
  SetDirection(RIGHT, OUT_REV);
  SetPower(LEFT + RIGHT, turn);
 }

//This function sets up the motors to move forward
void GoFwd()
 {
  SetDirection(RIGHT, OUT_REV);
  SetDirection(LEFT, OUT_REV);
  SetPower(LEFT + RIGHT, run);
 }

//This function sets up the motors to move in reverse
void GoRev()
 {
  SetDirection(RIGHT, OUT_FWD);
  SetDirection(LEFT, OUT_FWD);
  SetPower(LEFT + RIGHT, run);
 }

//This function is called when the helper dog must find the sheep
void findSheep()
 {
   while(EYE > sheep)
   {
    TurnLeft();
    On(LEFT + RIGHT);
   }
   Off(LEFT + RIGHT);
 }
task main()
 {
  SetSensorType(EYE, SENSOR_TYPE_LIGHT);         //Defines EYE as a light sensor
  SetSensorMode(EYE, SENSOR_MODE_RAW);                          //Sensor outputs RAW (0-1023) values
  SetSensorType(PEN_EYE, SENSOR_TYPE_LIGHT);                //Defines PEN_EYE as a light sensor
  SetSensorMode(PEN_EYE, SENSOR_MODE_RAW);                //Sensor outputs RAW (0-1023) values
  SetSensorType(EAR, SENSOR_TYPE_LIGHT);         //Defines EAR as a light sensor
  SetSensorMode(EAR, SENSOR_MODE_RAW);                          //Sensor outputs RAW (0-1023) values

  start avoidWall;

    while(true)                              //Wait until a sound is heard, and then move to a position
    {                                        //where the sheep is between the helper dog and the pen
     if (EAR <= sound)
     {
       stop avoidWall;

       while(EYE >= black)
       {
              GoRev();
              OnFor(LEFT + RIGHT, 100);
              TurnRight();
              OnFor(LEFT + RIGHT, 100);
       }

       findSheep();

       while(PEN_EYE > pen)
       {
         start avoidWall;
         TurnRight();
         OnFor(LEFT + RIGHT, 50);
         GoFwd();
         OnFor(LEFT + RIGHT, 150);
              findSheep();
       }
     }
    }
 }
```

Section 9.3 – Experiment Image Acquisition

This section consists of a detailed explanation of how the images were acquired during the experiments. The steps taken to process the images, as well as how these images were made into a movie will be discussed.

Section 9.3.1 – Image Acquisition

The images taken during experimentation were obtained using a camera that is fixed over the mobile robot test bed located in the CRIM laboratory. In order to be able to view the entire test bed, the camera is centered above the area of interest and has a fish-eye lens. Figure 9.15 shows a sample image acquired during experimentation. Notice the field in the upper right quadrant of the picture and the pen in the upper right-hand corner of the image. Using MATLAB, the distortion in this image can be removed. The processed image is shown in Figure 9.16. If desired, the image can be cropped such that only the area of interest is shown. For this application, only the field is shown in the final image. Notice how the defished image has been flipped over the horizon. This algorithm is used to correct

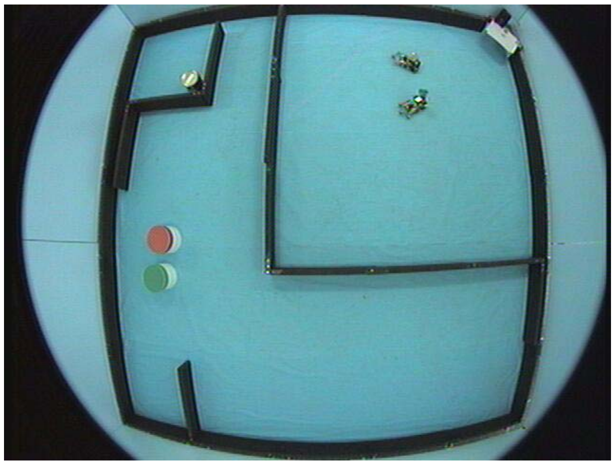


Figure 9.15 Image acquired while conducting experiments

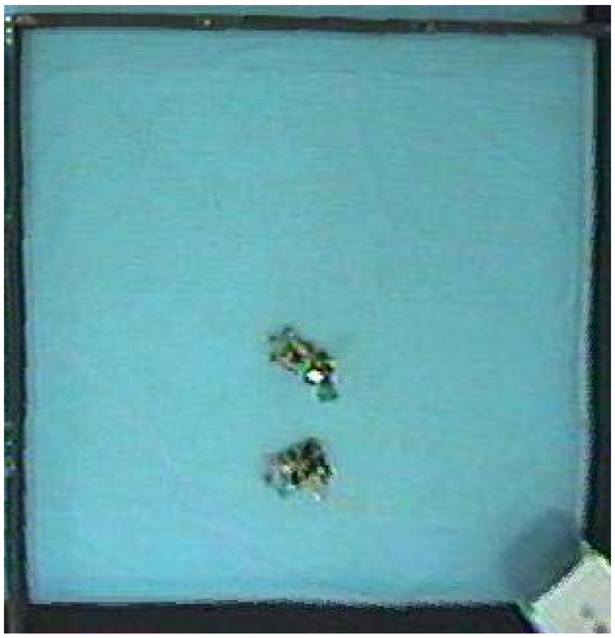


Figure 9.16 Defished and cropped image

images of various experiments that were taken with this camera. For a different application, the axes of a simulated world are oriented differently than the test bed. When the image is corrected, it is flipped so the axes of the defished image and those in the simulated world are aligned. Flipping the image does not serve any purpose for this application.

After all of the original images have been acquired, the robots used in the experiment can be tracked, and their individual paths can be plotted on the images (see Figure 9.17). For each image, the user clicks on the robot with the mouse. These points are stored in MATLAB and then connected for the sequence of images. Once the robot path information has been created, a MATLAB movie can then be created. When the movie is created, the image is flipped back to its original position. If desired, the MATLAB movie can be converted to a format (Audio Video Interleaved or AVI) that can be viewed using another software application, such as Windows Media Player.

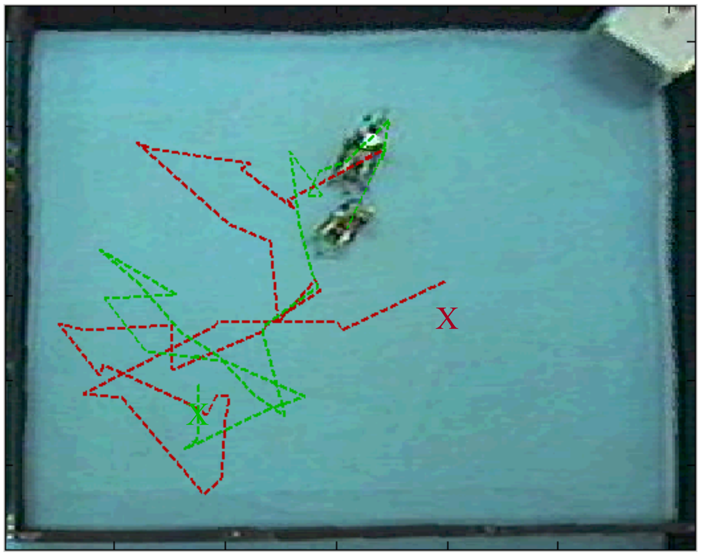


Figure 9.17 One frame of the MATLAB movie showing two robots and the paths they have taken

Section 9.3.2 – MATLAB Source Code

In the following section, the MATLAB source code used to process the images and make a movie with these processed images is listed. Andrew L. Nelson of the CRIM laboratory wrote all pieces of code, and each is included with his permission.

Section 9.3.2.1 defish_image_set.m

This file takes the original "fishy" image (see Figure 11.1) and removes this distortion (see Figure 11.2). The user must specify a path in the MATLAB working directory where the images are located. The distortion is removed from the images, and then saved into the same location with a different name. In order for this segment of code to run, the function *defish_image2.m* must be included in the MATLAB working directory also.

```
%defish_image_set.m                                                          %no comments past here *
%
%        Purpose:   Generate and save a set of defished images.
%
%
%        Record of revisions:
%                       Date                 Programmer                  Changes Made
%                       ====                 ==========                  ===========
%                       5-30-02              A. L. Nelson                Original Code
%
%        Notes:    1)   defish_image_set
%
%                  2)   Slow: takes about 30 sec per image.
%

clear all

%Set beginning and final image to be processed
start_index = 5; end_index = 62;

number_images = 0;

%Set file name strings so MATLAB can find where the images
%that need to be processed are located
%Note Matlab doesn't like dots (.) in path names
file_name_str = '3_11_03/test5';

from_file_name_string = [file_name_str '/image'];          %preprocessed image
to_file_name_string = [file_name_str '/defished_image'];    %processed image

%Processes images
for i = start_index:end_index
   number_images = number_images + 1;

   %plot top view camera image
   image_name_string = [from_file_name_string, num2str(i)];

   %Input file might be .jpg, .jpeg, or .bmp
   %Use one ot the following lines of code:

   %top_view_image = double(imread([image_name_string  '.bmp']));
   top_view_image = double(imread([image_name_string  '.jpg']));
   %top_view_image = double(imread([image_name_string  '.jpeg']));

   %Format image for plotting
   %function --> defish_image2.m must be in working directory
   top_view_image=defish_image2(top_view_image);

   %Plot real world
   subplot(1,1,1)
   image(top_view_image/255)
   axis xy
```

```
    axis square
    pause(0.001)

    %.bmp files are about 850K while .jpg files are about 30K
    %choose the type of file that is desired to be created
    %imwrite(top_view_image/255, [to_file_name_string, num2str(i), '.bmp']); disp(['Writing ', to_file_name_string, num2str(i), '.bmp'])
    imwrite(top_view_image/255, [to_file_name_string, num2str(i), '.jpg']); disp(['Writing ', to_file_name_string, num2str(i), '.jpg'])
end
```

Section 9.3.2.2 – defish_image2.m

This function performs the procedure of removing the fish-eye distortion from the original image. It also crops the image to the desired size of the user. This function must be in the MATLAB working directory, along with *defish_image_set.m* for the image processing to succeed.

```
%defish_image2.m
%
%        Purpose:   Version2: Remove fish-eye distortion from an over-head maze image.
%                            Image is also sized, cropped and fliped to by 540 by 540. (size is
%                            a result of the defishing process)
%
%        Record of revisions:
%                Date                     Programmer                    Changes Made
%                ====                     ==========                    ============
%                5-20-02            A. L. Nelson            Origonal Code
%                5-28-02            A. L. Nelson            Curve extrapolation method
%
%        Notes   1)      Called by video_get_range with defish_image2(640_by_480_image);
%
%                2)      Only for use with 480 by 640 .bpm images from overhead winTV cam
%
%                3)      defish_image2(white_grid_top_view_image);
%
%                4)       white_grid_top_view_image = double(imread(['calib_new_images/white_grid_on_maze.bmp']));
%
%                5)       this function is now set to crop the image around the "field" located in the upper
%                          right hand quadrant of the preprocessed fishy image

function [unfished_image]=y(fishy_image)

%Crop image so it is square
image_width = size(fishy_image, 2);
image_height = size(fishy_image, 1);

%Crop image x dimention = cols. (y is already 480)
%        size should be 480 by 480
center_error = 16;
fishy_image = fishy_image(:, 81+center_error:560+center_error, :);

%size(fishy_image) %debug

%Flip image over hoizon so axes match those of the simulated world (origin in lower left)
%        Note: flipud requires a 2D matrix...
fishy_image(:,:,1) = flipud(fishy_image(:,:,1));
fishy_image(:,:,2) = flipud(fishy_image(:,:,2));
fishy_image(:,:,3) = flipud(fishy_image(:,:,3));

max_offset = 100;

%pad array
for i = 1:3
```

```
    col_padded_fishy_image(:,:,i) = [zeros(image_height,max_offset) fishy_image(:,:,i) zeros(image_height,max_offset)];
end

new_image_width = size(col_padded_fishy_image, 2);

for i = 1:3
    padded_fishy_image(:,:,i) = [zeros(max_offset,new_image_width); col_padded_fishy_image(:,:,i);
zeros(max_offset,new_image_width)];
end

new_image_height = size(padded_fishy_image, 1);

center_index = round(new_image_height/2);

%  The index of is the radial distance of the point
%i.e. new 67 reads its element from 60...
shift_points = [1 2
    60 67
    115 134
    160 201
    196 269
    225 335
    275 480];
shift_points(:,1) = shift_points(:,1)*1.2;

max_radius = (2^0.5)*(center_index)+1;

shift_vector = []; %This is one of the few times where you must init a matrix var

%This loop creats a vector of linear extraplation points from shift_points
for count = 2:size(shift_points,1)
    start_ind = shift_points(count,2);
    prev_start_ind = shift_points(count-1,2);
    max_shift = shift_points(count,2) - shift_points(count,1);
    prev_max_shift = shift_points(count-1,2) - shift_points(count-1,1);
    new_elems = prev_max_shift:(max_shift-prev_max_shift)/(start_ind-prev_start_ind):max_shift;
    shift_vector = [shift_vector, new_elems];
end

%fill out any undefined elements in the linear extraplation shift curve vector
shift_vector(length(shift_vector):round(max_radius)) = 0;

x_dim = size(padded_fishy_image,2); %Columns
y_dim = size(padded_fishy_image,1);          %Rows

for quad_count = 1:4
    %Select a quadrant of the image. Note that there is a row-column to Cartesian coord change
    switch quad_count
    case 1
        padded_fishy_image_quad(:,:,1:3) = padded_fishy_image(ceil(y_dim/2)+1:y_dim,ceil(x_dim/2)+1:x_dim,1:3);

    case 2
        padded_fishy_image_quad(:,:,1:3) = padded_fishy_image(ceil(y_dim/2)+1:y_dim,1:ceil(x_dim/2),1:3);
        padded_fishy_image_quad(:,:,1) = fliplr(padded_fishy_image_quad(:,:,1));
        padded_fishy_image_quad(:,:,2) = fliplr(padded_fishy_image_quad(:,:,2));
        padded_fishy_image_quad(:,:,3) = fliplr(padded_fishy_image_quad(:,:,3));

    case 3
        padded_fishy_image_quad(:,:,1:3) = padded_fishy_image(1:ceil(y_dim/2),1:ceil(x_dim/2),1:3);
        padded_fishy_image_quad(:,:,1) = flipud(fliplr(padded_fishy_image_quad(:,:,1)));
        padded_fishy_image_quad(:,:,2) = flipud(fliplr(padded_fishy_image_quad(:,:,2)));
        padded_fishy_image_quad(:,:,3) = flipud(fliplr(padded_fishy_image_quad(:,:,3)));

    case 4
        padded_fishy_image_quad(:,:,1:3) = padded_fishy_image(1:ceil(y_dim/2),ceil(x_dim/2)+1:x_dim,1:3);
        padded_fishy_image_quad(:,:,1) = flipud(padded_fishy_image_quad(:,:,1));
        padded_fishy_image_quad(:,:,2) = flipud(padded_fishy_image_quad(:,:,2));
        padded_fishy_image_quad(:,:,3) = flipud(padded_fishy_image_quad(:,:,3));

    end %case quad_count
    for x_index = 1:center_index
        for y_index = 1:center_index
            radial_dist = (x_index^2 + y_index^2)^.5;

            if radial_dist < 1 %remove devide by zero errors
                radial_dist = 1;
            end

            cos_theta = x_index/radial_dist;
            sin_theta = y_index/radial_dist;

            x_offset = cos_theta*shift_vector(ceil(radial_dist));
            x_offset = round(x_offset);

            y_offset = sin_theta*shift_vector(ceil(radial_dist));
            y_offset = round(y_offset);


            x_read_from_index = x_index-x_offset;
            if x_read_from_index < 1 %remove zero index errors
                x_read_from_index = 1;
            end
            y_read_from_index = y_index-y_offset;
            if y_read_from_index < 1 %remove zero index errors
                y_read_from_index = 1;
            end
            new_image_matrix_quad(x_index, y_index,:) = padded_fishy_image_quad(x_read_from_index, y_read_from_index,:);
        end %for y_index
    end %for x_index

    %Un-flip and set resulting quadrant sub-matrixes
    switch quad_count
    case 1
        sub_image_quad1 = new_image_matrix_quad;
    case 2
        new_image_matrix_quad(:,:,1) = fliplr(new_image_matrix_quad(:,:,1));
        new_image_matrix_quad(:,:,2) = fliplr(new_image_matrix_quad(:,:,2));
        new_image_matrix_quad(:,:,3) = fliplr(new_image_matrix_quad(:,:,3));
        sub_image_quad2 = new_image_matrix_quad;

    case 3
        new_image_matrix_quad(:,:,1) = flipud(fliplr(new_image_matrix_quad(:,:,1)));
        new_image_matrix_quad(:,:,2) = flipud(fliplr(new_image_matrix_quad(:,:,2)));
        new_image_matrix_quad(:,:,3) = flipud(fliplr(new_image_matrix_quad(:,:,3)));
        sub_image_quad3 = new_image_matrix_quad;

    case 4
        new_image_matrix_quad(:,:,1) = flipud(new_image_matrix_quad(:,:,1));
        new_image_matrix_quad(:,:,2) = flipud(new_image_matrix_quad(:,:,2));
        new_image_matrix_quad(:,:,3) = flipud(new_image_matrix_quad(:,:,3));
        sub_image_quad4 = new_image_matrix_quad;

    end %case quad_count
end %for quad_count

%combine sub-images
uncroped_unfished_image = [sub_image_quad3 sub_image_quad4; sub_image_quad2 sub_image_quad1];

%Crop the image to only include the field (upper right-hand quadrent)
%           Std is 480 by 480.
%           Origonal was 640 by 480
crop_error = -5;
unfished_image(:,:,1:3)  = uncroped_unfished_image(81+crop_error:620+crop_error,81+crop_error:620+crop_error,1:3);
cropped_image(:,:,1:3)  = unfished_image(220:540,210:520,:);
unfished_image = cropped_image;
```

Section 9.3.2.3 – make_real_robot_paths.m

This segment of code is used to plot a robot path over a sequence of images. The user defines a sequence of images to use, and is then prompted to enter the position of the robot on the image by using the mouse.

```
%make_real_robot_paths.m
%
%	Purpose:	Show a sequence of images and collect a corresponding set of
%			robot path points generated from user input via the mouse...
%
%	Record of revisions:
%			Date		Programmer		Changes Made
%			====		==========		============
%			8-8-2001	A. L. Nelson		Origonal Code
%			2-28-2002	A. L. Nelson		Modified for multiple robots
%			2-19-2003	A. L. Nelson		Altered to make movie info for version 4 games
%
%	Notes	1)		Order of path following: red green red green.

clear robot_path;

axis on

%Set the file name strings so MATLAB can find the images to use
file_name_str = '3_11_03/test5';
%frame_prefix_string = [file_name_str, '/image'];		%use fish-eye images
frame_prefix_string = [file_name_str, '/defished_image'];	%use Defished images

%Set start and end image numbers
start_image_fraim_number = 5; end_image_fraim_number = 62;

%Use one or the other of these lines depending on file type
file_type_str = '.jpg';
%file_type_str = '.jpeg';
%file_type_str = '.bmp';	%big. try not to use

incr_intreval = 1;

%set number of robots that are used
number_of_robots = 2;

%are numbers padded or not...
padded = 0; %1 for winTV, 0 for server
pad_level = 4

for i = 1:pad_level
  pad_str(i) = '0';
end

disp('Order of path following: red green.');

%Collect path vectors
for robot_count = 1:number_of_robots
  path_index = 0;
  evalc(['clear robot', num2str(robot_count), '_path']);
  for wii = start_image_fraim_number:incr_intreval:end_image_fraim_number

    if padded
	count_str = pad_str;
      count_str(length(pad_str) - length(num2str(wii))+1:length(pad_str)) = num2str(wii)
    else
      count_str = wii;
    end
    path_index = path_index + 1;
              disp(['frame number' '  ' num2str(wii)]);
    image(imread([frame_prefix_string num2str(count_str) file_type_str]));

    if path_index ~= 1
            evalc(['line(robot', num2str(robot_count), '_path(:,1), robot', num2str(robot_count), '_path(:,2))']);
    end

    switch robot_count
      case 1
        disp('click on RED robot #1 center')
      case 2
        disp('click on GREEN robot #2 center')
    end

    evalc(['robot', num2str(robot_count), '_path(path_index, :) = GINPUT(1);']);

  end
  disp(['Done with robot'  num2str(robot_count), ' path: hit any key to continue'])
  pause
end

%Display last image with path lines
image(imread([frame_prefix_string num2str(count_str) file_type_str]));

%Set colors of the various paths that will be plotted
color_array(1,:) = [.7 0 0];
color_array(2,:) = [0 .7 0];

for robot_count = 1:number_of_robots
            evalc(['Hndl = line(robot', num2str(robot_count), '_path(:,1), robot', num2str(robot_count), '_path(:,2))']);
  set(Hndl, 'LineStyle', '--', 'Color', color_array(robot_count,:), 'LineWidth', 2.5);
end

%Save path data for figures and movies
%  Note Matlab doesn't like dots (.) in path names
save([file_name_str '/path_and_movie_data'],  'robot1_path', 'robot2_path', 'start_image_fraim_number', 'end_image_fraim_number',
'file_type_str', 'frame_prefix_string', 'padded');
```

Section 9.3.2.4 – robot_movie_maker_ver4.m

Using the sequence of images and the robot paths created for the images by *make_real_robot_paths.m*, a MATLAB movie is created. In order for a movie to be made, the file *path_and_movie_data.m* that was created when the robot paths were defined must be in the same location as the images that are being used to create the movie. Once a MATLAB movie has been created, it can be played using MATLAB or converted to AVI movie using the MATLAB function *movie2avi*. Once in AVI format, the movie can be played using Windows Media Player or another application.

```
%robot_movie_maker_ver4.m
%
%          Purpose:    Show a sequence of images and collect a corresponding set of
%                                          robot path points generated from make_real_robot_paths.m
%
```

```
%
%       Record of revisions:
%               Date                    Programmer              Changes Made
%               ====                    ==========              ============
%               8-8-2001                A. L. Nelson            Origonal Code
%               2-28-2002                A. L. Nelson           Modified for multiple robots
%               6-3-2002                A. L. Nelson            Modified to play movies
%                2-20-2003              A. L. Nelson            Modified for Version 4 movies
%
%       Notes   1)      Call with robot_movie_player_ver4
%
%     2)  A file '/path_and_movie_data' must exist at file_name_str,
%        generated by make_real_robot_paths.m

clear robot_path;

axis on
draw_paths = 1

%Set the file name string so MATLAB can find the images and the path information to use
file_name_str = '3_11_03/test5';
load([file_name_str, '/path_and_movie_data'])

%These variables are all now set by loading /path_and_movie_data (all saved from path tracking)
%frame_prefix_string =
%path_data_string =
%start_image_fraim_number =
%end_image_fraim_number =
%file_type_str =
%padded =

%Sets color of robot paths
color_array(1,:) = [.7 0 0];
color_array(2,:) = [0 .7 0];

incr_intreval = 1
number_of_robots = 2
pad_level = 4

for i = 1:pad_level
  pad_str(i) = '0';
end

  path_index = 0;
  for wii = start_image_fraim_number:incr_intreval:end_image_fraim_number

    if padded
        count_str = pad_str;
      count_str(length(pad_str) - length(num2str(wii))+1:length(pad_str)) = num2str(wii)
    else
      count_str = wii;
    end

    path_index = path_index + 1;
    disp(['Making frame number' '  ' num2str(wii)]);
    image(imread([frame_prefix_string num2str(count_str) file_type_str]));
    title(['frame number' '  ' num2str(wii)])

    if draw_paths
        for robot_count = 1:number_of_robots
            evalc(['Hndl = line(robot', num2str(robot_count), '_path(1:', num2str(path_index), ',1), robot', num2str(robot_count),
'_path(1:', num2str(path_index), ',2))']);
            set(Hndl, 'LineStyle', '--', 'Color', color_array(robot_count,:), 'LineWidth', 1.5);
      end
    end

    axis xy
    pause(.01)
    movie_frames(path_index) = getframe;
  end
```

```
clf
frames_per_second = 1;
times_to_play = 2;
disp(['Done making movie.  The movie will play ', num2str(times_to_play), ' times at ', num2str(frames_per_second), ' frames per second']);

movie(movie_frames,times_to_play,frames_per_second);

save new_movie movie_frames

movie(movie_frames);   %plays MATLAB movie

break

%Display last image with path lines
image(imread([frame_prefix_string num2str(count_str) file_type_str]));

color_array(1,:) = [.7 0 0];
color_array(2,:) = [0 .7 0];

for robot_count = 1:number_of_robots
   evalc(['Hndl = line(robot', num2str(robot_count), '_path(:,1), robot', num2str(robot_count), '_path(:,2))']);
   set(Hndl, 'LineStyle', '--', 'Color', color_array(robot_count,:), 'LineWidth', 2.5);
end
title(['frame number' '  ' num2str(wii)])

axis equal
axis tight
axis off
axis xy
```

Section 9.4 – Parts List

An extensive parts list and datasheets for several major components used in the microphone sensor are included in this section. Specifications of the LEGO® multiplexer used on the sheepdog are also included.

Section 9.4.1 – Microphone Sensor Parts

Listed below are the components used to make one microphone sensor. Resistors and capacitors were chosen based on their value and size, not ratings. For example, the 10000 pF capacitor used for the tone decoder board is rated at 100V. This high voltage rating is not needed for this application, but the capacitor was used because it was in stock and it met the size and value requirements. All components were purchased from Digikey Corporation, and the part numbers listed are the Digikey part numbers.

Section 9.4.1.1 – Component List

Table 9.1 Components used for the microphone circuit board

Microphone Circuit Board Components (see Figure 5.16)		
Quantity	Part Description	Digikey Part Number
1	Panasonic Omnidirectional Electret Condenser Microphone Cartridge (WM-52B)	P9970-ND
1	680 Ω Resistor (⅛ W, 1%, 0805 SMD)	311-680CCT-ND
1	3.3 µF Capacitor (6.3V ceramic X5R 0805)	PCC1925CT-ND

Table 9.2 Components used for the amplifier circuit board

Amplifier Circuit Board Components (see Figure 5.16)		
Quantity	Part Description	Digikey Part Number
1	µA741 General Purpose Operational Amplifier (8-SOIC)	296-11106-5-ND
2	1 kΩ Resistor (⅛ W, 1%, 0805 SMD)	311-1.00KCCT-ND
1	100 Ω Resistor (⅛ W, 1%, 0805 SMD)	311-100CCT-ND
1	10 kΩ Resistor (⅛ W, 1%, 0805 SMD)	311-10.0KCCT-ND
1	10 µF Capacitor (6.3V ceramic X5R 0805)	PCC2225CT-ND

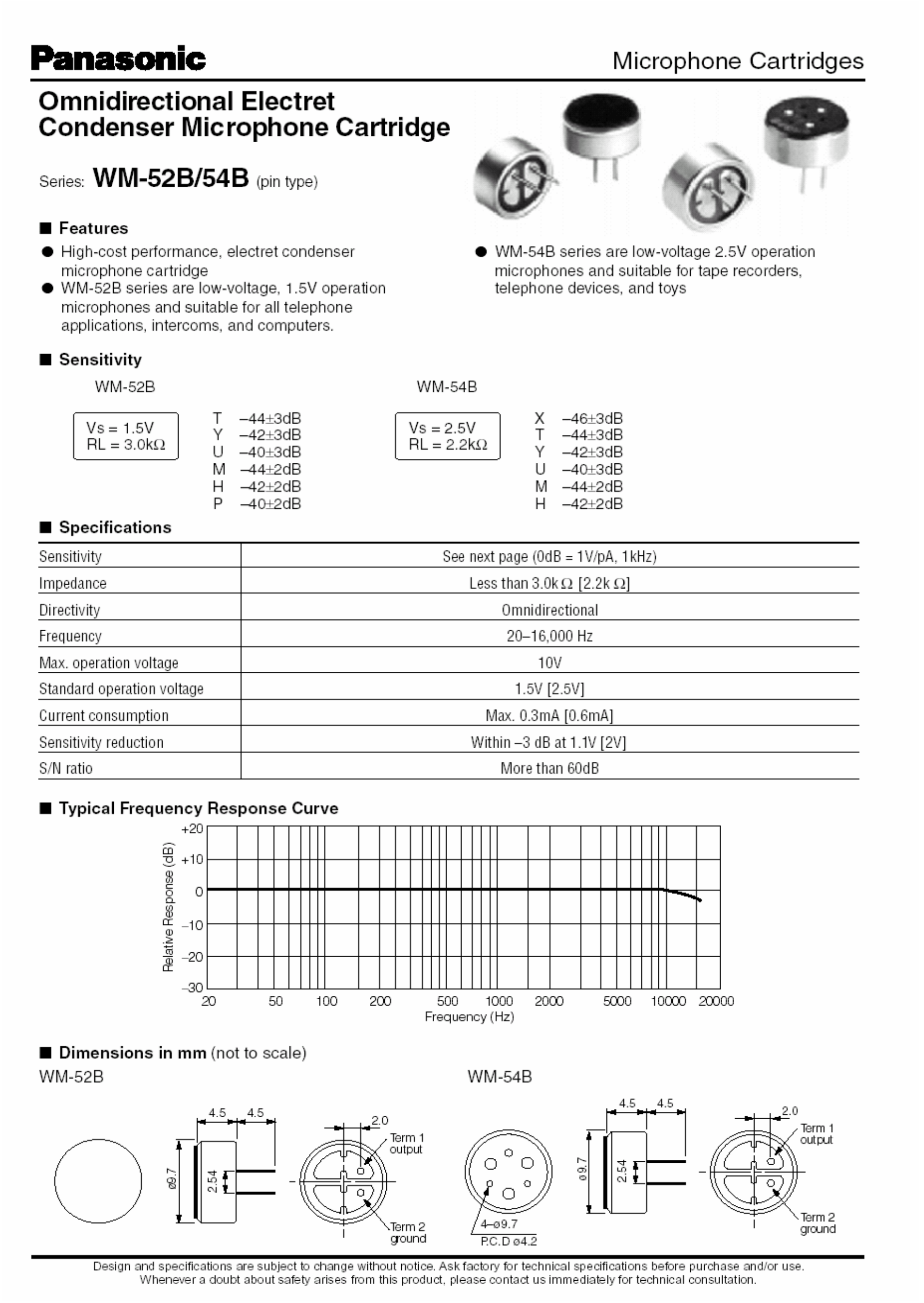
Table 9.3 Components used for the tone detector circuit board

Tone Decoder Circuit Board Components (see Figure 5.16)		
Quantity	Part Description	Digikey Part Number
1	FSK Demodulator/Tone Decoder (SO14)	NJM2211M
2	0.1 µF Capacitor (25V ceramic X7R 0805)	PCC1828CT-ND
1	0.47 µF Capacitor (16V ceramic X7R 0805)	PCC1818CT-ND
1	56000 pF Capacitor (16V ceramic X7R 0805)	PCC1809CT-ND
1	10000 pF Capacitor (100V ceramic X7R 0805)	399-1159-1-ND
1	5 kΩ Potentiometer	CT20P502-ND
1	470 kΩ Resistor (⅛ W, 1%, 0805 SMD)	311-470KCCT-ND
1	10 kΩ Resistor (⅛ W, 1%, 0805 SMD)	311-10.0KCCT-ND
1	1 MΩ Resistor (⅛ W, 1%, 0805 SMD)	311-1.00MCCT-ND
1	XXX Ω Resistor used to set center frequency (⅛ W, 1%, 0805 SMD)	TBD

Section 9.4.1.2 – Component Datasheets

Portions of the datasheets for the major components used in the microphone sensor are included below.

Section 9.4.1.2.1 – Panasonic Microphone [39]



Panasonic Microphone Cartridges

Omnidirectional Electret Condenser Microphone Cartridge

Series: **WM-52B/54B** (pin type)

■ **Features**

- High-cost performance, electret condenser microphone cartridge
- WM-52B series are low-voltage, 1.5V operation microphones and suitable for all telephone applications, intercoms, and computers.
- WM-54B series are low-voltage 2.5V operation microphones and suitable for tape recorders, telephone devices, and toys

■ **Sensitivity**

WM-52B

Vs = 1.5V
RL = 3.0kΩ

T	−44±3dB
Y	−42±3dB
U	−40±3dB
M	−44±2dB
H	−42±2dB
P	−40±2dB

WM-54B

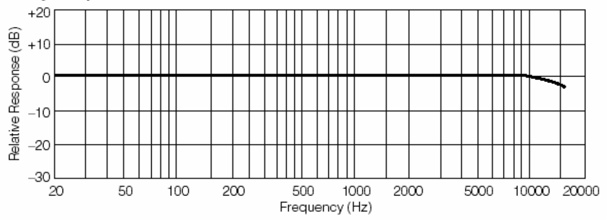
Vs = 2.5V
RL = 2.2kΩ

X	−46±3dB
T	−44±3dB
Y	−42±3dB
U	−40±3dB
M	−44±2dB
H	−42±2dB

■ **Specifications**

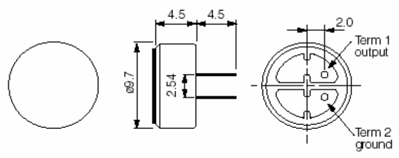
Sensitivity	See next page (0dB = 1V/pA, 1kHz)
Impedance	Less than 3.0k Ω [2.2k Ω]
Directivity	Omnidirectional
Frequency	20–16,000 Hz
Max. operation voltage	10V
Standard operation voltage	1.5V [2.5V]
Current consumption	Max. 0.3mA [0.6mA]
Sensitivity reduction	Within −3 dB at 1.1V [2V]
S/N ratio	More than 60dB

■ **Typical Frequency Response Curve**

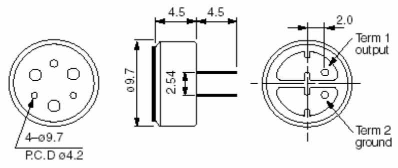


■ **Dimensions in mm** (not to scale)

WM-52B



WM-54B



Design and specifications are subject to change without notice. Ask factory for technical specifications before purchase and/or use.
Whenever a doubt about safety arises from this product, please contact us immediately for technical consultation.

Section 9.4.1.2.2 – Texas Instruments Op-Amp [48]

μA741, μA741Y GENERAL-PURPOSE OPERATIONAL AMPLIFIERS

SLOS094B – NOVEMBER 1970 – REVISED SEPTEMBER 2000

- **Short-Circuit Protection**
- **Offset-Voltage Null Capability**
- **Large Common-Mode and Differential Voltage Ranges**
- **No Frequency Compensation Required**
- **Low Power Consumption**
- **No Latch-Up**
- **Designed to Be Interchangeable With Fairchild μA741**

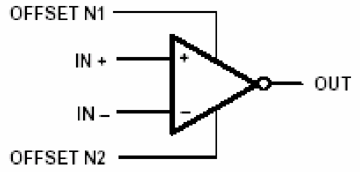
description

The μA741 is a general-purpose operational amplifier featuring offset-voltage null capability.

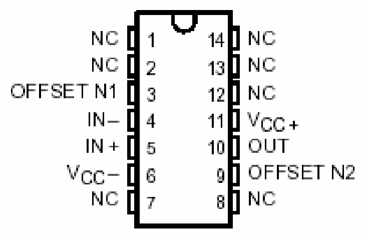
The high common-mode input voltage range and the absence of latch-up make the amplifier ideal for voltage-follower applications. The device is short-circuit protected and the internal frequency compensation ensures stability without external components. A low value potentiometer may be connected between the offset null inputs to null out the offset voltage as shown in Figure 2.

The μA741C is characterized for operation from 0°C to 70°C. The μA741I is characterized for operation from −40°C to 85°C.The μA741M is characterized for operation over the full military temperature range of −55°C to 125°C.

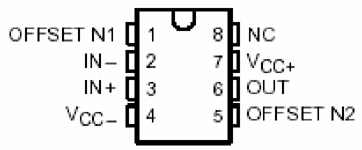
symbol



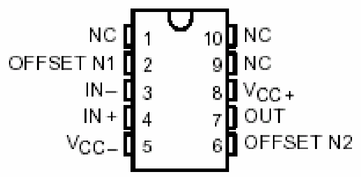
μA741M . . . J PACKAGE
(TOP VIEW)



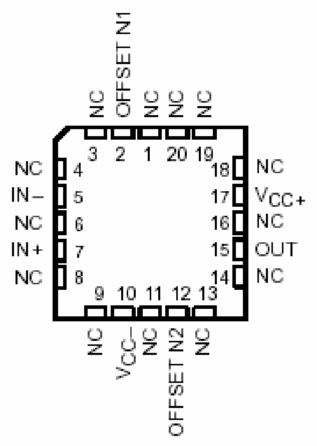
μA741M . . . JG PACKAGE
μA741C, μA741I . . . D, P, OR PW PACKAGE
(TOP VIEW)



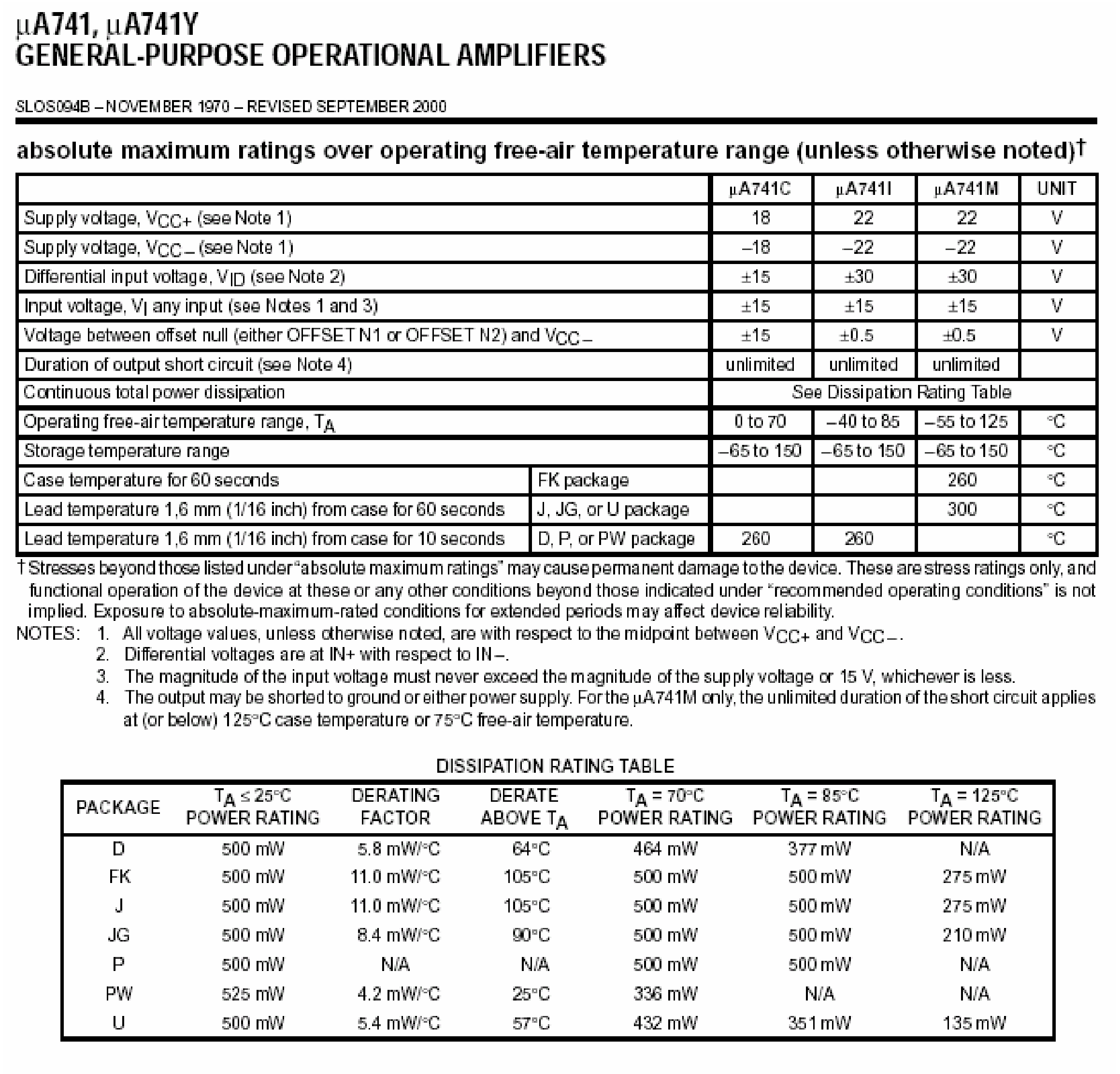
μA741M . . . U PACKAGE
(TOP VIEW)



μA741M . . . FK PACKAGE
(TOP VIEW)



NC – No internal connection



absolute maximum ratings over operating free-air temperature range (unless otherwise noted)†

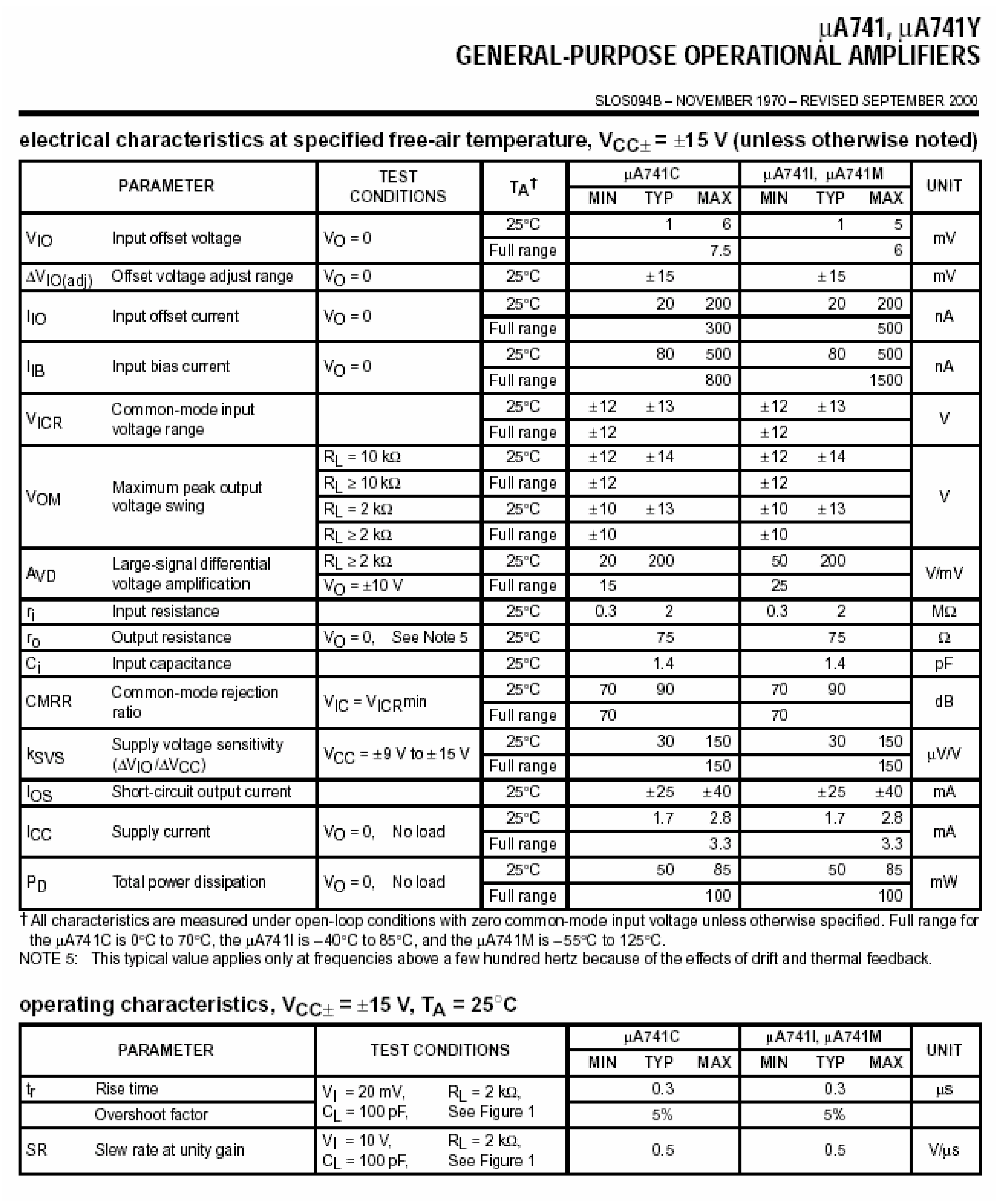
		μA741C	μA741I	μA741M	UNIT
Supply voltage, V_{CC+} (see Note 1)		18	22	22	V
Supply voltage, V_{CC-} (see Note 1)		–18	–22	–22	V
Differential input voltage, V_{ID} (see Note 2)		±15	±30	±30	V
Input voltage, V_I any input (see Notes 1 and 3)		±15	±15	±15	V
Voltage between offset null (either OFFSET N1 or OFFSET N2) and V_{CC-}		±15	±0.5	±0.5	V
Duration of output short circuit (see Note 4)		unlimited	unlimited	unlimited	
Continuous total power dissipation		See Dissipation Rating Table			
Operating free-air temperature range, T_A		0 to 70	–40 to 85	–55 to 125	°C
Storage temperature range		–65 to 150	–65 to 150	–65 to 150	°C
Case temperature for 60 seconds	FK package			260	°C
Lead temperature 1,6 mm (1/16 inch) from case for 60 seconds	J, JG, or U package			300	°C
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds	D, P, or PW package	260	260		°C

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTES: 1. All voltage values, unless otherwise noted, are with respect to the midpoint between V_{CC+} and V_{CC-}.
2. Differential voltages are at IN+ with respect to IN–.
3. The magnitude of the input voltage must never exceed the magnitude of the supply voltage or 15 V, whichever is less.
4. The output may be shorted to ground or either power supply. For the μA741M only, the unlimited duration of the short circuit applies at (or below) 125°C case temperature or 75°C free-air temperature.

DISSIPATION RATING TABLE

PACKAGE	$T_A \leq 25°C$ POWER RATING	DERATING FACTOR	DERATE ABOVE T_A	$T_A = 70°C$ POWER RATING	$T_A = 85°C$ POWER RATING	$T_A = 125°C$ POWER RATING
D	500 mW	5.8 mW/°C	64°C	464 mW	377 mW	N/A
FK	500 mW	11.0 mW/°C	105°C	500 mW	500 mW	275 mW
J	500 mW	11.0 mW/°C	105°C	500 mW	500 mW	275 mW
JG	500 mW	8.4 mW/°C	90°C	500 mW	500 mW	210 mW
P	500 mW	N/A	N/A	500 mW	500 mW	N/A
PW	525 mW	4.2 mW/°C	25°C	336 mW	N/A	N/A
U	500 mW	5.4 mW/°C	57°C	432 mW	351 mW	135 mW



electrical characteristics at specified free-air temperature, $V_{CC\pm}$ = ±15 V (unless otherwise noted)

PARAMETER		TEST CONDITIONS	T_A†	µA741C			µA741I, µA741M			UNIT
				MIN	TYP	MAX	MIN	TYP	MAX	
V_{IO}	Input offset voltage	V_O = 0	25°C		1	6		1	5	mV
			Full range			7.5			6	
$\Delta V_{IO(adj)}$	Offset voltage adjust range	V_O = 0	25°C		±15			±15		mV
I_{IO}	Input offset current	V_O = 0	25°C		20	200		20	200	nA
			Full range			300			500	
I_{IB}	Input bias current	V_O = 0	25°C		80	500		80	500	nA
			Full range			800			1500	
V_{ICR}	Common-mode input voltage range		25°C	±12	±13		±12	±13		V
			Full range	±12			±12			
V_{OM}	Maximum peak output voltage swing	R_L = 10 kΩ	25°C	±12	±14		±12	±14		V
		R_L ≥ 10 kΩ	Full range	±12			±12			
		R_L = 2 kΩ	25°C	±10	±13		±10	±13		
		R_L ≥ 2 kΩ	Full range	±10			±10			
A_{VD}	Large-signal differential voltage amplification	R_L ≥ 2 kΩ	25°C	20	200		50	200		V/mV
		V_O = ±10 V	Full range	15			25			
r_i	Input resistance		25°C	0.3	2		0.3	2		MΩ
r_o	Output resistance	V_O = 0, See Note 5	25°C		75			75		Ω
C_i	Input capacitance		25°C		1.4			1.4		pF
CMRR	Common-mode rejection ratio	$V_{IC} = V_{ICR}min$	25°C	70	90		70	90		dB
			Full range	70			70			
k_{SVS}	Supply voltage sensitivity ($\Delta V_{IO}/\Delta V_{CC}$)	V_{CC} = ±9 V to ±15 V	25°C		30	150		30	150	µV/V
			Full range			150			150	
I_{OS}	Short-circuit output current		25°C		±25	±40		±25	±40	mA
I_{CC}	Supply current	V_O = 0, No load	25°C		1.7	2.8		1.7	2.8	mA
			Full range			3.3			3.3	
P_D	Total power dissipation	V_O = 0, No load	25°C		50	85		50	85	mW
			Full range			100			100	

† All characteristics are measured under open-loop conditions with zero common-mode input voltage unless otherwise specified. Full range for the µA741C is 0°C to 70°C, the µA741I is –40°C to 85°C, and the µA741M is –55°C to 125°C.

NOTE 5: This typical value applies only at frequencies above a few hundred hertz because of the effects of drift and thermal feedback.

operating characteristics, $V_{CC\pm}$ = ±15 V, T_A = 25°C

PARAMETER		TEST CONDITIONS	µA741C			µA741I, µA741M			UNIT
			MIN	TYP	MAX	MIN	TYP	MAX	
t_r	Rise time	V_I = 20 mV, R_L = 2 kΩ, C_L = 100 pF, See Figure 1		0.3			0.3		µs
	Overshoot factor			5%			5%		
SR	Slew rate at unity gain	V_I = 10 V, R_L = 2 kΩ, C_L = 100 pF, See Figure 1		0.5			0.5		V/µs

electrical characteristics at specified free-air temperature, $V_{CC\pm} = \pm15$ V, $T_A = 25°C$ (unless otherwise noted)

PARAMETER		TEST CONDITIONS	μA741Y			UNIT
			MIN	TYP	MAX	
V_{IO}	Input offset voltage	$V_O = 0$		1	6	mV
$\Delta V_{IO(adj)}$	Offset voltage adjust range	$V_O = 0$		±15		mV
I_{IO}	Input offset current	$V_O = 0$		20	200	nA
I_{IB}	Input bias current	$V_O = 0$		80	500	nA
V_{ICR}	Common-mode input voltage range		±12	±13		V
V_{OM}	Maximum peak output voltage swing	$R_L = 10$ kΩ	±12	±14		V
		$R_L = 2$ kΩ	±10	±13		
A_{VD}	Large-signal differential voltage amplification	$R_L \geq 2$ kΩ	20	200		V/mV
r_i	Input resistance		0.3	2		MΩ
r_o	Output resistance	$V_O = 0$, See Note 5		75		Ω
C_i	Input capacitance			1.4		pF
CMRR	Common-mode rejection ratio	$V_{IC} = V_{ICR}$min	70	90		dB
k_{SVS}	Supply voltage sensitivity ($\Delta V_{IO}/\Delta V_{CC}$)	$V_{CC} = \pm9$ V to ±15 V		30	150	μV/V
I_{OS}	Short-circuit output current			±25	±40	mA
I_{CC}	Supply current	$V_O = 0$, No load		1.7	2.8	mA
P_D	Total power dissipation	$V_O = 0$, No load		50	85	mW

† All characteristics are measured under open-loop conditions with zero common-mode voltage unless otherwise specified.
NOTE 5: This typical value applies only at frequencies above a few hundred hertz because of the effects of drift and thermal feedback.

operating characteristics, $V_{CC\pm} = \pm15$ V, $T_A = 25°C$

PARAMETER		TEST CONDITIONS	μA741Y			UNIT
			MIN	TYP	MAX	
t_r	Rise time	$V_I = 20$ mV, $R_L = 2$ kΩ, $C_L = 100$ pF, See Figure 1		0.3		μs
	Overshoot factor			5%		
SR	Slew rate at unity gain	$V_I = 10$ V, $R_L = 2$ kΩ, $C_L = 100$ pF, See Figure 1		0.5		V/μs

Section 9.4.1.2.3 – New Japan Radio Tone Decoder [38]

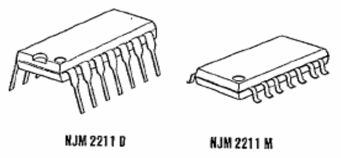
JRC **NJM2211**

FSK DEMODULATOR/TONE DECODER

■ GENERAL DESCRIPTION

The NJM2211 is a monolithic phase-locked loop (PLL) system especially designed for data communications. It is particularly well suited for FSK modem applications, and operates over a wide frequency range of 0.01Hz to 300kHz. It can accommodate analog signals between 2mV and 3V, and can interface with conventional DTL, TTL and ECL logic families. The circuit consists of a basic PLL for tracking an input signal frequency within the passband, a quadrature phase detector which provides carrier detection, and an FSK voltage comparator which provides FSK demodulation. External components are used to independently set carrier frequency, bandwidth, and output delay.

■ PACKAGE OUTLINE



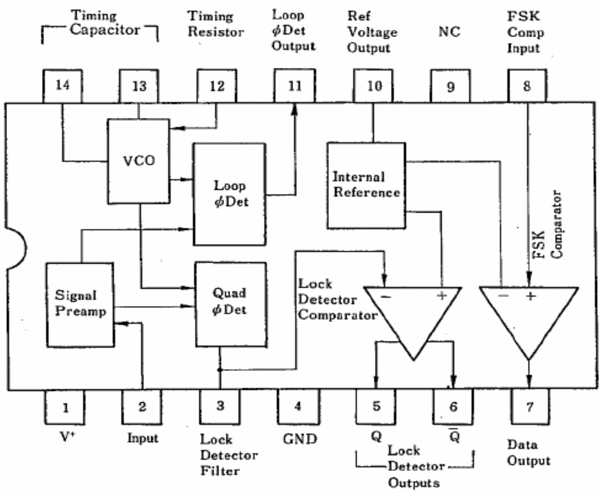
■ FEATURES

● Wide Operating Voltage	(4.5V to 20V)
● Wide frequency range	(0.01Hz to 300 kHz)
● DTL/TTL/ECL logic compatibility	
● FSK demodulation with carrier-detector	
● Wide dynamic range	(2mV to $3V_{rms}$)
● Adjustable tracking range	(±1% to ±80%)
● Excellent temperature stability	(20ppm/°C typical)
● Package Outline	DIP14, DMP14
● Bipolar Technology	

■ APPLICATIONS

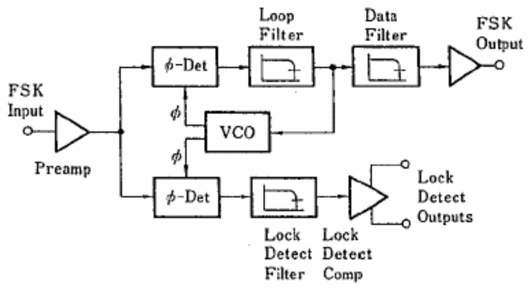
- FSK demodulation
- Data synchronization
- Tone decoding
- FM detection
- Carrier detection

■ PIN CONFIGURATION



NJM2211D
NJM2211M

■ BLOCK DIAGRAM



■ ABSOLUTE MAXIMUM RATINGS

(Ta=25℃)

PARAMETER	SYMBOL	RATINGS	UNIT
Supply Voltage	V^+	20	V
Input Signal Level	V_{IN}	3	V_{rms}
Power Dissipation	P_D	(DIP14) 700	mW
		(DMP14) 300	mW
Operating Temperature Range	T_{opr}	−40～+85	℃
Storage Temperature Range	T_{stg}	−40～+125	℃

■ ELECTRICAL CHARACTERISTICS

($V^+=+12V$, $Ta=25°C$)

PARAMETER	SYMBOL	TEST CONDITION	MIN.	TYP.	MAX.	UNIT
Operating Voltage	V^+		4.5	—	20	V
Operating Current	I_{CC}	$R_0 \geq 10k\Omega$	—	5	11	mA
Oscillator						
Frequency Accuracy	Δf_0		—	±1.0	—	%
Frequency Stability Temp. Coefficient	$\Delta f_0/\Delta T$	$R_1=\infty$	—	±20	—	ppm/°C
Power Supply Rejection	PSRR	$V^+=12\pm1V$ $V^+=5\pm0.5V$	—	±0.05 ±0.2	±1.5	%/V %/V
Upper Frequency Limit	$f_{0\,MAX}$	$R_0=8.2k\Omega$, $C_0=400pF$	—	300	—	kHz
Lowest Operating Frequency	$f_{0\,MIN}$	$R_0=2M\Omega$, $C_0=50\mu F$	—	0.01	—	Hz
Timing Resistor						
Timing Resistor	R_0	Operating Range	5	—	2000	kΩ
		Recommended Range	15	—	100	kΩ
Loop Phase Detector						
Peak Output Current	I_0	Meas. at pin 11	±100	±200	±300	μA
Output Offset Current	I_{OS}		—	±2.0	—	μA
Output Impedance	Z_0		—	1.0	—	MΩ
Maximum Voltage Swing	V_{OM}	Ref. to pin 10	±4.0	±5.0	—	V
Quadrature Phase Detector						
Peak Output Current	I_0	Meas. at Pin 3	—	150	—	μA
Output Impedance			—	1.0	—	MΩ
Maximum Voltage Swing			—	11	—	$V_{p\text{-}p}$
Input Preamp						
Input Impedance	R_{IN}	Meas. at Pin 2	—	20	—	kΩ
Input Signal Voltage Required to Cause Limiting	V_{IN}		—	2	—	mV_{rms}

Voltage Comparator

Input Impedance	R_{IN}	Measure at Pin 3 & 8	—	2	—	MΩ
Input Bias Current	I_B		—	100	—	nA
Voltage Gain	G_V	R_L=5.1kΩ	—	70	—	dB
Output Voltage Low	V_{SAT}	5, 6, 7_{PIN} I_C=3mA	—	0.3	1.0	V
Output Leakage Current	I_{LEAK}	V_0=12V	—	0.01	11	μA

Internal Reference

Output Voltage	V_{REF}	Measure at Pin 10	4.75	5.30	5.85	V
Output Impedance	Z_0		—	100	—	Ω

■ CIRCUIT FUNCTION

• Singal Input (Pin 2)

The input signal is AC coupled to this terminal. The internal impedance at pin 2 is 20kΩ. Recommended input signal leveles in the range of $10mV_{rms}$ to $3V_{rms}$.

• Quadrature Phase Detector Output (Pin 3)

This is the high-impedance output of the quadrature phase detector, and is internally connected to the input of lock-detect voltage comparator. In tone detection applications, pin 3 is connected to ground through a parallel combination of R_D and C_D (see Figure 1) to eliminate chatter at the lock-detect outputs. If this tone-detect section is not used, pin 3 can be left open circuited.

• Lock-Detect Output, Q (Pin 5)

The output at pin 5 is at a "high" state when the PLL is out of lock and goes to a "low" or conducting state when the PLL is locked. It is an open collector type output and required a pull-up resistor, R_L, to V^+ for proper operation. In the "low" state it can sink up to 5mA of load current.

• Lock-Detect Complement, Q (Pin 6)

The output at pin 6 is the logic complement of the lock-detect output at pin 5. This output is also an open collector type stage which can sink 5mA of load current in the low or "on" state.

• FSK Data Output (Pin 7)

This output is an open collector logic stage which requres a pull-up resistor, R_L, to V^+ for proper operation. It can sink 5mA of load current. When decoding FSK signals the FSK data output will switch to a "high" or off state for low input frequency, and will switch to a "low" or on state for high input frequency. If no input signal is present, the logic state at pin 7 is indeterminate.

• FSK Comparator Input (Pin 8)

This is the high-impedance input to the FSK voltage comparator. Normally, an FSK post-detection or data filter is connected between this terminal and the PLL phase-detector output (pin 11). This data filter is formed by R_F and C_F of Figure 1. The threshold voltage of the comparator is set by the internal reference voltage, V_R, available at pin 10.

• Reference Voltage, V_R (Pin 10)

This pin is internally biased at the reference voltage level, V_R; $V_R=V+/2-650mV$. The DC voltage level at this pin forms an internal reference for the voltage levels at pin 3, 8, 11, and 12. Pin 10 must be bypassed to ground with a 0.1μF capacitor.

• Loop Phase Detector Output (Pin 11)

This terminal provides a high impedance output for the loop phase-detector. The PLL loop filter is formed by R1 and C1 connected to pin 11 (see Figure 1). With no input signal, or with no phase error within the PLL, the DC level at pin 11 is very nearly equal to V_{REF}. The peak voltage swing available at the phase detector output is equal to $\pm V_{REF}$.

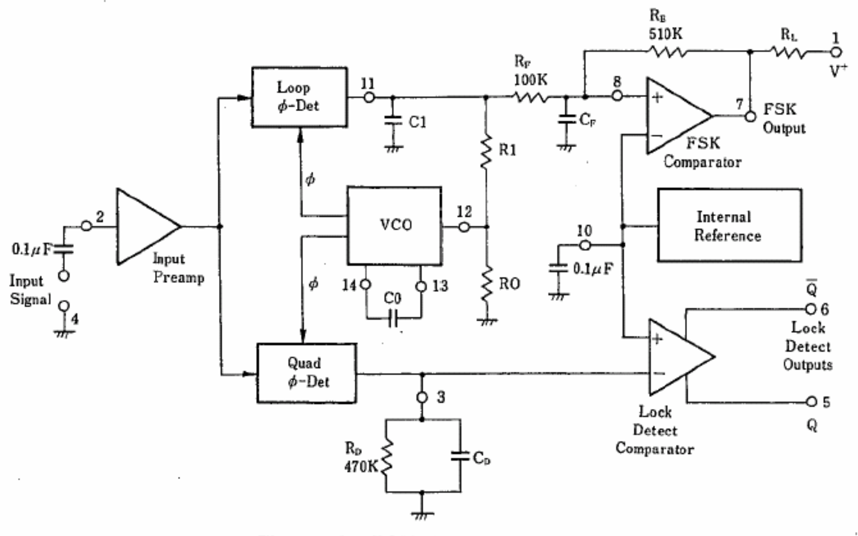


Figure 1 FSK & Tone Detection

- **VCO Control Input (Pin 12)**

VCO free-running frequency is determined by external timing resistor, R0, connected from this terminal to ground. The VCO free-running frequency, f_0, is given by:

$$f_0(\mathrm{Hz}) = \frac{1}{R0C0}$$

where C0 is the timing capacitor across pins 13 and 14. For optimum temperature stability R0 must be in the range of 10kΩ to 100kΩ (see Typical Electrical Characteristics).
This terminal is a low impedance point, and is internally biased at a DC level equal to V_R. The maximum timing current drawn from pin 12 must be limited to ≦3mA for proper operation of the circuit.

- **VCO Timing Capacitor (Pins 13 and 14)**

VCO frequency is inversely proportional to the external timing capacitor, C0, connected across these terminals. C0 must be non-polarized, and in the range of 200pF to 10μF.

- **VCO Frequncy Adjustment**

VCO can be fine tuned by connecting a potentiometer, R_X, in series with R0 at pin 12 (see Figure 2)

- **VCO Free-Running Frequency, F_0**

The NJM2211 does not have a separate VCO output terminal. Instead, the VCO outputs are internally connected to the phase-detector sections of the circuit. However, for setup or adjustment purposes, the VCO free-running frequency can be measured at pin 3 (with C_D disconnected) with no input and also pin 2 shorted to pin 10.

■ DESIGN EQUATIONS

See Figure 1 for Definitions of Components.

1. VCO Center Frequency, f_0:

$$f_0(\mathrm{Hz}) = \frac{1}{R0C0}$$

2. Internal Reference Voltage, V_R (measured at pin 10):

$$V_R = \left(\frac{+V_S}{2}\right) - 650\mathrm{mV}$$

3. Loop Lowpass Filter Time Constant, τ:

$$\tau = R1C1$$

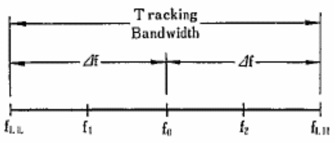
4. Loop Damping, ζ:

$$\zeta = \left(\sqrt{\frac{C0}{C1}}\right)\left(\frac{1}{4}\right)$$

5. Loop Tracking Bandwidth, $\pm\Delta f/f_0$:

$$\Delta f/f_0 = R0/R1$$

$\Delta f/f_0 = R0/R1$



6. FSK Date Filter Time Constant, τ_F:

$$\tau_F = R_F C_F$$

7. Loop Phase Detector Conversion Gain, K_ϕ:
(K_ϕ is the differential DC voltage across pins 10 and 11, per unit of phase error at phase-detector input):

$$K\varphi\ (\text{in volts per radian}) = \frac{(-2)(V_{REF})}{\pi}$$

8. VCO conversion Gain, K0, is the amount of change in VCO frequency per unit of DC voltage change at pin 11:

$$K0\ (\text{in Hertz per volt}) = \frac{-1}{C0R1V_{REF}}$$

9. Total Loop Gain K_T:
K_T (in radians per second per volt) $= 2\pi K\phi K0$
$= 4/C0R1$

10. Peak Phase-Detector Current, I_A:

$$I_A(\mathrm{mA}) = \frac{V_{REF}}{25}$$

Tone Detection

Figure 4 shows the generalized circuit connection for tone detection. The logic outputs, Q and $\overline{Q}$ at pins 5 and 6 are normally at "high" and "low" logic states, respectively. When a tone is present within the detection band of the PLL, the logic state at these outputs becomes reversed for the duration of the input tone. Each logic output can sink 5mA of load current.

Both logic outputs at pins 5 and 6 are open-collector type stages, and require external pull-up resistors R_{L1} and R_{L2} as shown in Figure 4.

With reference to Figure 1 and 4, the function of the external circuit components can be explained as follows: R0 and C0 set VCO center frequency, R1 sets the detection bandwidth, C1 sets the lowpass-loop filter time constant and the loop damping factor, and R_{L1} and R_{L2} are the respective pull-up resistors for the Q and $\overline{Q}$ logic outputs.

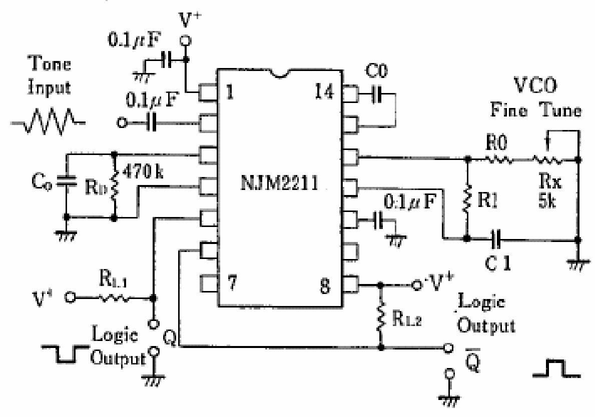


Figure 4. Tone Detection

Design Instructions

The circuit of Figure 4 can be optimized for any tone-detection application by the choice of five key circuit components: R0, R1 C0, C1, and C_D. For a given input tone frequency, f_S, these parameters are calculated as follows:

1. Choose R0 to be in the range of 15kΩ to 100kΩ. This choice is arbitrary.
2. Calculate C0 to set center frequency, f_0 equal to f_S: $C0=1/R0f_S$.
3. Calculate R1 to set bandwidth $\pm\Delta f$ (see Design Equation No. 5): $R1=R0\ (f_0/\Delta f)$

Note: The total detection bandwidth covers the frequency range of $f_0=\Delta f$.

4. Calculate value of C1 for a given loop damping factor:
   $C1=C0/16\zeta^2$
   Normally $\zeta\approx 1/2$ is optimum for most tone-detector applications, giving $C1=0.25\ C0$.
   Increasing C1 improves the out-of-band signal rejection, but increases the PLL capture time.
5. Calculate value of filter capacitor C_D. To avoid chatter at the logic output, with $R_D=470k\Omega$, C_D must be:
   $C_D\ (\mu F)\geqq (16/\text{capture range in Hz})$
   Increasing C_D slows the logic output response time.

Design Examples

Tone detector with a detection band of 1kHz±20Hz:

Step 1: Choose R0=20kΩ (18kΩ in series with 5kΩ potentiometer).
Step 2: Choose C0 for f_0=1kHz: C0=0.05μF.
Step 3: Calculate R1: R1=(R0) (1000/20)=1MΩ.
Step 4: Calculate C1: for $\zeta=1/2$, C1=0.25μF, C0=0.013μF.
Step 5: Calculate C_D: C_D=16/38=0.42μF.
Step 6: Fine tune the center frequency with the 5kΩ potentiometer, R_X.

Section 9.4.2 – LEGO® Multiplexer

The multiplexer used on the sheepdog was purchased from Mindsensors Robotics [1]. This three-channel active multiplexer allows for up to three LEGO® sensors to be plugged into a single input port on the RCX. The circuit diagram of the multiplexer is shown in

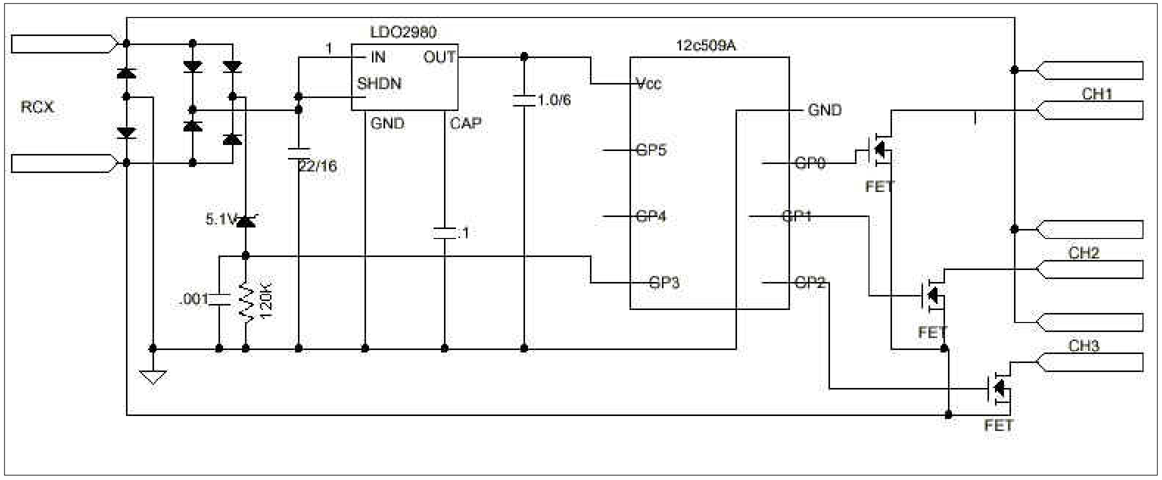


Figure 9.18 Circuit diagram of the 3-channel active multiplexer [1]

Figure 9.18. There are some limitations to using this multiplexer. Only sensors that require power can be connected to the multiplexer. For example, a light sensor or angle sensor can be used with the multiplexer, but a touch sensor cannot. Also, if the multiplexer is not connected to the RCX in the correct polarity, it will not work. When power the RCX is turned on, only CH1 should receive power. Take the following steps to check for correct polarity [“Active”]:

1. Connect the multiplexer to the RCX
2. Connect a light sensor to CH2
3. Switch on the RCX
4. If the light sensor is ON (red LED will be on), reverse the polarity of the RCX connection by rotating it 180º

The technical specifications of the multiplexer are included in Table 9.4.

Table 9.4 Technical specifications of the three-channel multiplexer [1]

Total power consumption	4 mW (< 3% of total power available)
Current consumption (with no sensor connected)	500 mA
Voltage drop (with Light sensor connected)	50 mV
Channel selection time	75 ms
Channel access logic	Random access to any channel
Size (W x L x H)	2 by 8 by 2 plate with 2 by 4 block with plate at
Connector	Standard Mindstorms electric connector plate on top of a 2 by 8 for sensor connection
Devices used	High reliability solid state SMT and low power microcontroller