The University of Southern Mississippi

Alignment of Robot Orientations Using Limited Sensing Capabilities

by

Daniel Lucas Thompson

April 2014

Approved by

Bikramjit Banerjee

School of Computing

Chaoyang Zhang, Chair

School of Computing

Abstract

The process of making robots cooperate presents many challenges. These challenges become even greater as we transition from the current high quality robots that include many advanced sensors and equipment such as video and global positioning to the more simple and cost effective models that have less and lower quality equipment. When dealing with the simple robots, performing simple tasks can often need complex solutions. This thesis deals with a solution to one of these simple tasks. Often when robots are cooperating, each robot needs to know the orientation of its partner robot. With a simple robot, finding this orientation can be a difficult task. When we add obstacles to the environment, this task becomes even more complex. In this thesis, a solution will be designed and implemented to bring two robots from an unknown orientation to an alignment where they are both facing each other with an environment that contains obstacles.

Key Terms: robotics, alignment, multiple robots, infrared, orientation

Table of Contents

List of Illustrations

Chapter 1 Introduction to the Problem

When working with multiple robots, the need for them to cooperate can often arise. The more complex the robot is, the simpler this task can become. Modern day robots are often equipped with advanced equipment and sensors such as video or global positioning. This advanced equipment simplifies complex tasks and makes a simple task's complexity almost insignificant. When a simple robot with low quality equipment tries to perform these same simple tasks, the complexity can be huge.

There are many difficulties to overcome when robots try to work together. When multiple robots must cooperate, the task they are trying to perform becomes easier to accomplish if they are able to start from a known alignment, such as two robots directly facing each other. Getting two robots to align is a problem with many difficulties. When we add obstacles to the environment that the robots are attempting to align in, even more complications arise.

After doing research into various robotic alignment techniques, I came across many solutions to similar problems, but none of the the solutions fit the problem being solved in this thesis. While all of the various papers listed specific implementations for robotic alignment, all of them had some aspect that disqualified them for use in this project.

These papers covered research into the alignment and docking of modular robots. Modular robots are a single robot made of many smaller pieces working together. These robots are able to disconnect from each other to perform multiple tasks. The robots are then able to reconnect in order to form itself back into a single robotic state. During the reconnecting process, the various pieces of the robots must align their docking ports in order to connect. The alignment of these docking ports is very similar to the alignment needed for the problem being solved in this thesis. However, for various other reasons, the implementations designed in these papers were not applicable to this problem.

In Roufas [1], they make use of advanced equipment that is not available to the robots being used for this thesis. This paper makes use of four infrared emitting diodes placed on the opposing corners of a docking plate on one robot and two infrared receivers on another robot. These sensors include the ability to read the intensity of a signal. By having the diodes emit in sequence, they are able to use the intensity of each signal to triangulate an estimated position. The ability to read the intensity of multiple emitters to triangulate an estimated position is far beyond the capabilities of the sensors built into our robots.

In Quiňonez [2], they use an approach of having an active robot seek out and align with an inactive stationary robot. They also make use of proximity sensors which work similar to the infrared sensors contained within our robot to help the active robot seek out the inactive robot. However, they rely heavily on a video camera during the process. The robots used in this thesis are not equipped with a video camera.

In Shen [3], multiple infrared transmitters and receivers are used to implement a docking solution. The robots used in this paper have separate infrared transmitters and receivers attached to each of its three docking pins. Similar to Roufas [1], the sensors include the ability to measure signal strength. When the modules come in range of alignment, it measures the signal strength and uses the measurements to make estimates on the position and alignment. This, like the other papers, offers more capabilities than the robots used in this thesis have access to.

In Delrobaei [4], they make use of multiple infrared transmitters and a rotating infrared beam collector. Once again, this equipment is far more advanced than the equipment available to the robots used in this thesis, so this implementation is not applicable.

The purpose of this project is to implement a program to allow two robots with simple sensors to line up in an orientation facing each other in an environment that contains obstacles. To get two robots to align, they must have some form of sensor that can be used to locate another robot. The types of sensors can vary, but in this implementation, the robots will be using an infrared transmitter and receiver for detection and a microphone and speaker for communication.

The use of infrared sensors are not completely accurate, and this is the main problem that this project must address. When an infrared beam is transmitted into an obstacle located in the environment, that beam will reflect from that obstacle which can lead to false readings. In this implementation, the robots must find a way to distinguish between a direct signal and a reflected signal.

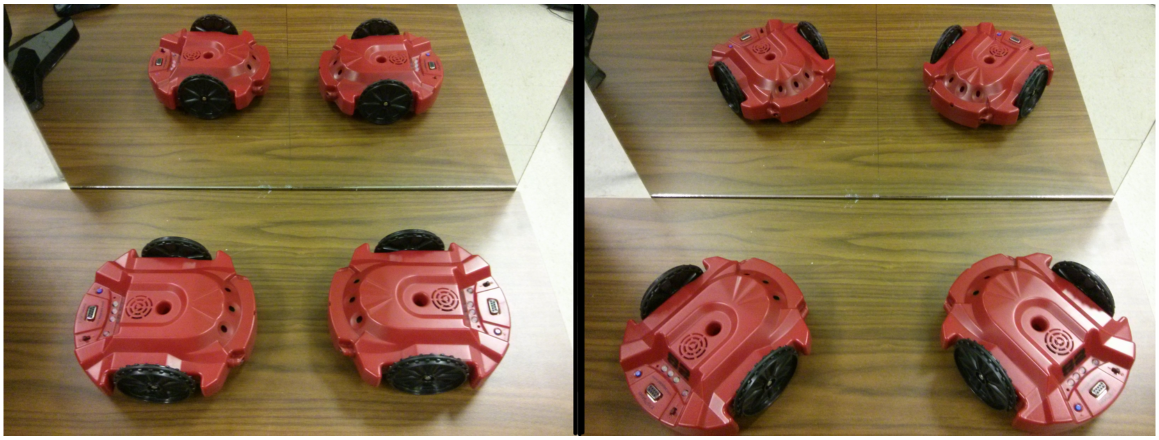


Illustration 1: Problem Demonstration

In the problem demonstration in Illustration 1, the problem will be explained more clearly. In the left image, we have two robots that are in alignment. They are directly facing each other. This is the desired outcome of this project. At this point, the robots know they are facing each other, and it will make any future coordinated movements easier because each robot knows where the other started from. However, getting to that outcome can prove problematic. In the right image, the major problem of this project is demonstrated. Each robot is directly facing the opposite robot's reflection in the mirror. This means that any infrared signals being emitted from one robot are being reflected from the mirror towards the opposite robot. This means a robot is receiving an infrared signal even though he is not in an alignment directly facing the opposite robot. Many obstacles in the real world reflect infrared just like a mirror reflects visible light, so although a mirror is used to illustrate the challenge of this problem, the challenge persists in the real world with other kinds of obstacles. This is the problem this thesis must solve.

Chapter 2 Introduction to the Robots

2.1 Scribbler S2 Robot

The implementation for this problem will be done using the S2 Scribbler robots created and sold by Parallax Inc. The S2 Scribbler robot is a low cost, fully assembled, robot that is capable of performing many tasks. It includes a built in serial port that allows for the installation and execution of custom programs. The robot contains the Parallax P8X32A Propeller micro-controller which contains eight separate cores known as cogs. This micro-controller is programmed using the SPIN programming language which contains a mixture of high level programming and assembly programming.

The S2 Scribbler also contains several sensors and other equipment that can be used in custom programs. It contains an infrared transmitter and receiver, a microphone, a speaker, a light sensor, four color LED lights, and two independent wheeled motors with stall sensors. When combined with the Propeller micro-controller and its multiple cogs, these sensors and equipment can be used simultaneously to allow the S2 Scribbler robot to perform many complex tasks.

2.2 Robot Limitations

The S2 Scribbler robot has a wide array of equipment and sensors, but due to its low cost, the sensors and equipment are not the most accurate. The speaker and microphone transmission and detection frequencies can often vary by large values when playing notes that should be identical. The infrared sensors are designed for obstacle detection for a single robot. The robot has an infrared receiver located on the center of its face in the forward direction. It has two infrared transmitters on its right and left side. The transmitters are aligned at a slight inward angle. This slight angle allows reflected infrared beams to bounce off of an object in front of the robot and be directed at the receiver located on the center of the robots face. This presents a problem with accuracy when trying to transmit to another robot since the signals will not be coming from a single direct source. Another problem, which is not common to the S2 Scribbler, is the accuracy of the wheeled motors. Any robot, regardless of the motor and wheel type, will have trouble making precisely measured turns due to the friction of the surface it is turning on. The effect of friction on the movement of a robot is often referred to as slippage. Solutions to these inaccuracies will need to be developed in order to solve the problem presented.

Chapter 3 Plan to Solve the Problem

To solve this problem, two S2 Scribbler robots will be used. They will begin in a random orientation within sensor distance of each other. The robots will be designated as a master robot and a slave robot. The master robot will handle all calculations, and it will send instructions to the slave robot. The slave robot will wait for instructions from the master robot, and once received, perform that action.

The robots will be using infrared scanning and signaling in a 360 degree area. This area will be broken up into smaller sections referred to as slices. Each robot must use the same slice size, but different sizes can be used. Theoretically, a large number of slices would give us more data points to perform our calculations, but with an increase in the number of slices, the number of turns the robot must make also increases. As the number of turn the robot makes increases, larger amounts of error due to slippage are introduced to the problem. To keep the amount of slippage error to a minimum, a slice size of 45 degrees, or 8 slices, will be used for this project.

To begin, the slave robot will constantly emit bursts from the infrared transmitter. The master robot will begin rotating and making scans through each of the slices until it makes a complete 360 degree turn. Once it has achieved a complete rotation, it will signal to the slave robot. The slave robot will turn through a single slice and continue emitting the infrared signals. The master robot will then make another complete rotation as it scans through each slice. The robots will continue this pattern until the slave robot has made a complete 360 degree rotation through all of it's slices. At this point, the master robot will have infrared data from every possible alignment scenario in each slice. It will calculate which slice it received the strongest signal from, and it will rotate back to that slice. Next, it will calculate which slice the slave robot sent the strongest signal from. It will signal the slave robot to rotate back to that slice. At this point, the robots should be in a forward facing orientation.

Chapter 4 Robot Communications

During the various steps of the solution to this problem. The robots need a method to communicate with each other. As discussed earlier, the poor quality of the frequency readings using the microphone and speaker represents the largest obstacle to this project. If the robots are unable to communicate reliably, the implementation of this project will not work.

The design of this messaging system was inspired by the Morse code system of sending messages. Morse code was chosen due to its representation of many messages using only a small number of symbols. With the frequency range of these robots, very few tones could be accurately included in that range and still allow room for error checking. Since Morse code uses a system of only two symbols, dots and dashes, it was a good starting point.

Due to the limitations of the speaker and microphone, the messages being sent will be played in a continuous tone. This decision was made due to the fact that the speaker introduces distortion into the beginning and end of the tone when the speaker turns on and off. The continuous tone is a collection of notes that will be played from beginning to end with no pauses between notes.

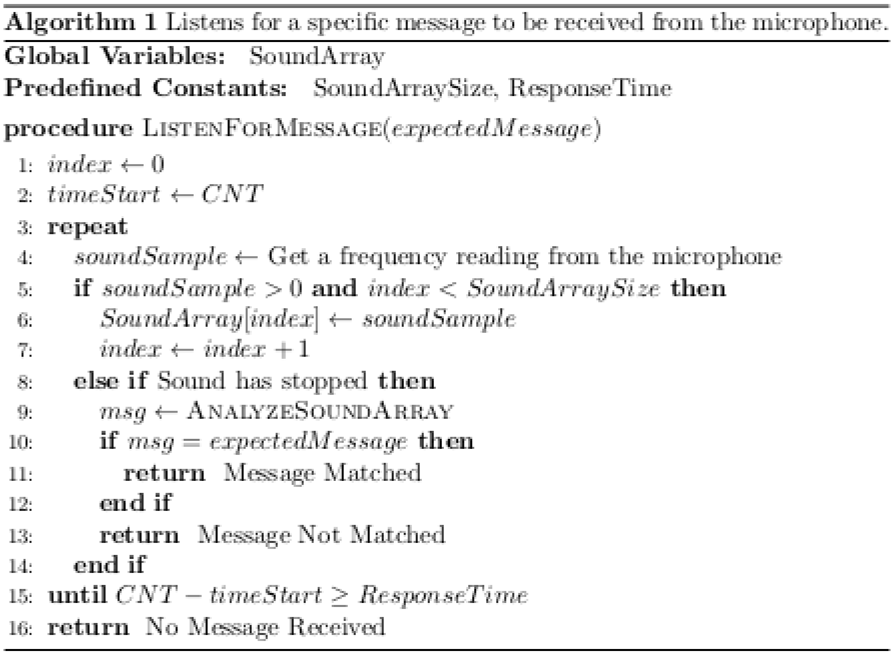
Due to this solution not allowing pauses between notes, the Morse code idea needed to be modified to work with this project. The modification was to represent the pauses in Morse code with a tone which is why this system uses a three tone system. The three tones used in this project are represented by the names DOT, BREAK, and DASH. The messaging system has the ability to store a large number of message types using varying sequences of the three different tones. However, the robots only use a total of 13 messages. Each message is represented as an array of frequencies that contain sequences of the DOT, BREAK, and DASH tones.

When the master robot is required to send a message, it plays the tones from the designated array. It continues transmitting this message until it receives a response from the slave robot that verifies the command was received and executed. After the master robot receives this response, it replies with its own response to let the slave robot know that the confirmation was received and the scanning routine has continued..

The slave robot handles this in a similar way. It waits for a command to be sent. Once the slave robot receives the message, it performs the command associated with that message. After the action has been completed, the slave robot sends a response to confirm the command has been executed. It continues sending this response until the master robot replies that it has received the response. At this point, both robots know all actions have been completed, and they can continue with the routine.

4.1 Capturing the Message

The method for capturing a message differs slightly for the slave and master robot. The overall concept is the same, but there are two main differences between the two methods. First, the master robot is only expecting a single confirmation message, so it only checks for the message it is expecting. Second, it must resend its original message if no response is received, so it has a time limit. The slave robot uses this method to listen for the master robots reply to its confirmation message, but it must also receive a command before it can perform an action. Due to this, the slave robot makes use of a second listening method that doesn't have a timeout, so it continues listening until it receives some message value.

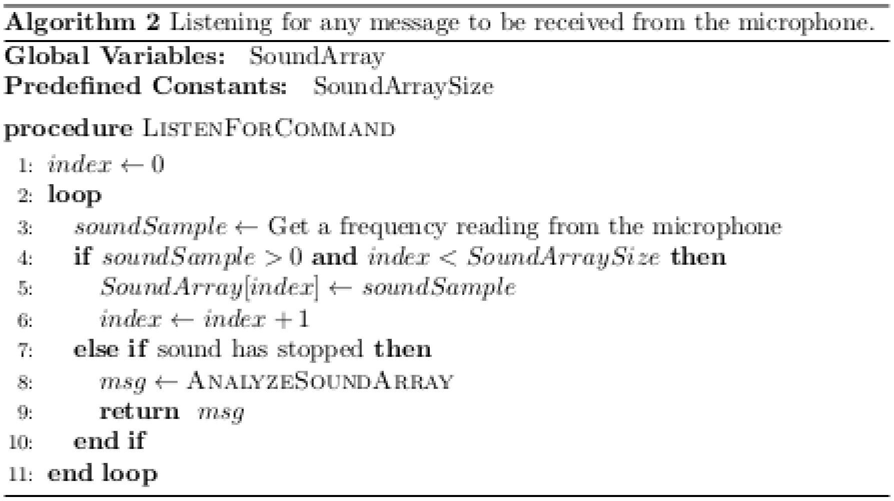


Algorithm 1 Listens for a specific message to be received from the microphone.

```
Global Variables: SoundArray
Predefined Constants: SoundArraySize, ResponseTime
procedure ListenForMessage($expectedMessage$)
$index \leftarrow 0$
$timeStart \leftarrow CNT$
repeat
    $soundSample \leftarrow$ Get a frequency reading from the microphone
    if $soundSample > 0$ and $index < SoundArraySize$ then
        $SoundArray[index] \leftarrow soundSample$
        $index \leftarrow index + 1$
    else if Sound has stopped then
        $msg \leftarrow$ AnalyzeSoundArray
        if $msg = expectedMessage$ then
            return Message Matched
        end if
        return Message Not Matched
    end if
until $CNT - timeStart \geq ResponseTime$
return No Message Received
```

Illustration 2: Algorithm 1

Algorithm 1 shows the procedure the robots use to listen for the confirmation message. Line 1 initializes the index variable which is used to save the frequencies received into an array. Line 2 initializes the variable timeStart to the value received from CNT. CNT is the internal clock time for the robot. Lines 3 through 15 are the loop used to constantly sample and store the frequencies being read from the microphone. For lines 4 through 7, the robot samples a frequency from the microphone and stores it into the sound array. Lines 8 through 14 check the message received for a match once the sounds have stopped. Line 9 analyzes the sound array and returns a message or a failure to receive a message. The procedure AnalyzeSoundArray is explained in the next section. Lines 10 through 13 return if the message was a match for the expected message. Line 15 continues the loop listening for messages for a set amount of time. If no matching message is received within the time limit, line 16 returns that nothing was received.



Algorithm 2 Listening for any message to be received from the microphone.

Global Variables: SoundArray
Predefined Constants: SoundArraySize

```
procedure ListenForCommand
 1: $index \leftarrow 0$
 2: loop
 3:    $soundSample \leftarrow$ Get a frequency reading from the microphone
 4:    if $soundSample > 0$ and $index < SoundArraySize$ then
 5:       $SoundArray[index] \leftarrow soundSample$
 6:       $index \leftarrow index + 1$
 7:    else if sound has stopped then
 8:       $msg \leftarrow$ AnalyzeSoundArray
 9:       return $msg$
10:    end if
11: end loop
```

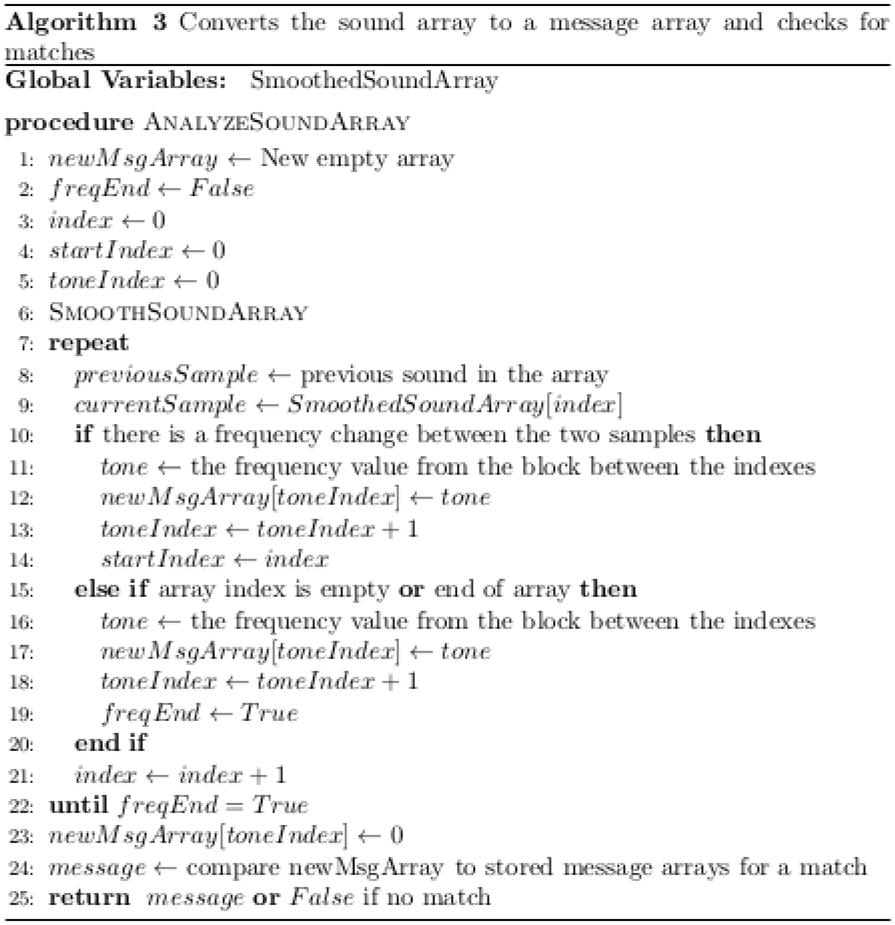
Illustration 3: Algorithm 2

Algorithm 2 shows the procedure used by the slave robot to wait for a command.

Line 1 initializes an index variable for accessing the sound array. Line 2 through 11 contains the continuous loop for listening for a command. Lines 3 through 6 read in frequencies from the microphone and store them in the sound array. Lines 7 through 10 check the sound array for a match when the sound has stopped playing. The AnalyzeSoundArray procedure is then run on the array and that message is returned.

4.2 Analyzing the Sound Array

To get a message from the stored sound array, it must be analyzed to check for a match. The array is analyzed by breaking the array into blocks. The blocks are the sections of an array that contain the same tones. The tones used are designated DOT, BREAK, and DASH which have frequencies of 1000, 3500, and 5000 respectively. Each block of of tones is recorded as a single tone into a new array, and that array is compared to the stored predefined messages for a match.



Algorithm 3 Converts the sound array to a message array and checks for matches

Global Variables: SmoothedSoundArray

```
procedure AnalyzeSoundArray
 1: $newMsgArray \leftarrow$ New empty array
 2: $freqEnd \leftarrow False$
 3: $index \leftarrow 0$
 4: $startIndex \leftarrow 0$
 5: $toneIndex \leftarrow 0$
 6: SmoothSoundArray
 7: repeat
 8:    $previousSample \leftarrow$ previous sound in the array
 9:    $currentSample \leftarrow SmoothedSoundArray[index]$
10:    if there is a frequency change between the two samples then
11:       $tone \leftarrow$ the frequency value from the block between the indexes
12:       $newMsgArray[toneIndex] \leftarrow tone$
13:       $toneIndex \leftarrow toneIndex + 1$
14:       $startIndex \leftarrow index$
15:    else if array index is empty or end of array then
16:       $tone \leftarrow$ the frequency value from the block between the indexes
17:       $newMsgArray[toneIndex] \leftarrow tone$
18:       $toneIndex \leftarrow toneIndex + 1$
19:       $freqEnd \leftarrow True$
20:    end if
21:    $index \leftarrow index + 1$
22: until $freqEnd = True$
23: $newMsgArray[toneIndex] \leftarrow 0$
24: $message \leftarrow$ compare newMsgArray to stored message arrays for a match
25: return $message$ or $False$ if no match
```

Illustration 4: Algorithm 3

Algorithm 3 contains the procedure for analyzing the sound array and returning the received message. Lines 1 through 5 initialize a new empty array and various index variables. Line 6 runs a smoothing procedure on the stored global sound array in order to remove some error and saves the new resulting sound array into the global variable SmoothedSoundArray which will be used for processing. The details of this smoothing procedure is discussed in the next section. Lines 7 through 22 analyze the array until it reaches and empty value or the end of the array is reached. Lines 8 and 9 assign the current and previous sample values in the sound array. Line 10 through 14 checks for a change in frequency between these two samples. If a change in frequency is detected, line 11 assigns the frequency value from the previous samples to the tone variable. The values in the array between the previous samples will all contain the same DOT, BREAK, or DASH values with a small variance in the actual frequency values. A single DOT, BREAK, or DASH is assigned to the tone variable depending on which frequencies were stored in the sample. Line 12 stores this tone in the new array that was declared earlier. Lines 15 through 20 are a special case to store the last block in the array if there is no change in the frequencies. It performs the same storage operation as the previous if block. After the the new array has been filled with DOT, BREAK, or DASH tones, line 23 adds a terminating 0 to mark the end of the array. Line 24 and 25 compares the newly created array to the predefined values for each of the 13 messages the robots use and returns the message type that was received or false if there is no match.

4.3 Smoothing the Sound Array

Before the array can be compared to find which message it matches, a smoothing process needs to be run on it. Due to the quality of the speaker and microphone, the frequency of a particular note can have some variance. Also, ambient noise can be recorded while the notes are playing. Both of these can have an effect on the values stored in the array of frequencies, so some form of correction must be applied to the array.

When a robot is playing one of the tones contained in a message, that tone is played for a distinct amount of time, so it can be expected the array should contain many consecutive indexes that contain the same frequency. To smooth the frequencies in the array, a method of analyzing the array similar to algorithm 3 can be used. Similar to algorithm 3, the array can be broken into blocks of indexes that contain each frequency type. The size of these indexes will then be considered. Since a tone will be played for a distinct amount of time, blocks with a small size can be discarded as noise which will leave the larger blocks of frequencies which should represent the actual tones.

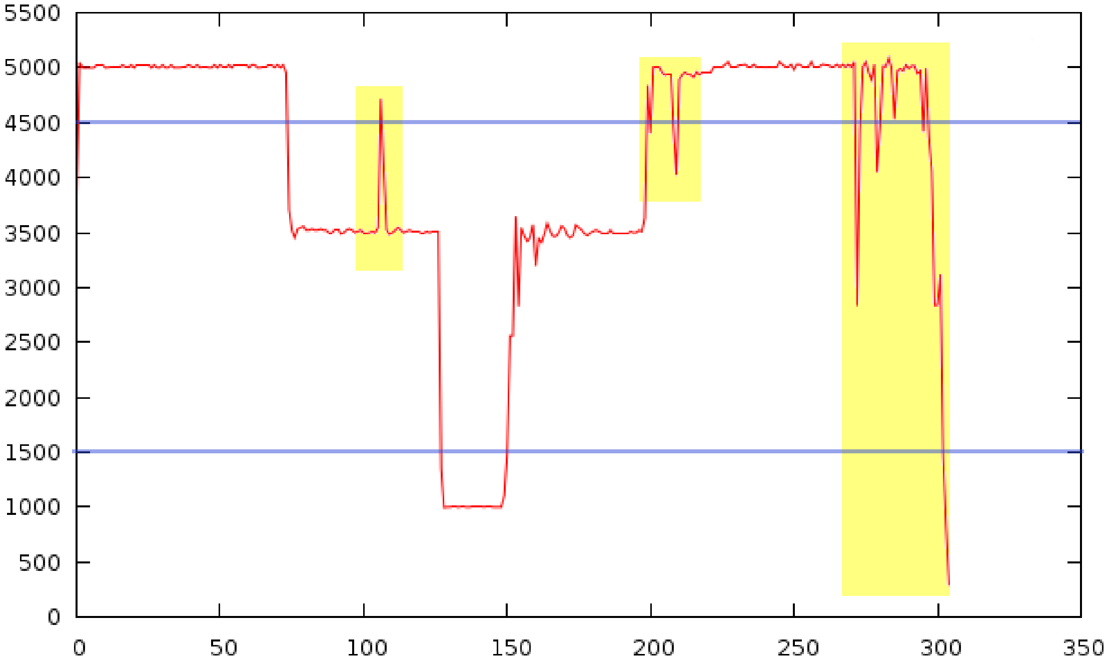


Illustration 5: Non-Smoothed Frequency Graph

The non-smoothed frequency graph in Illustration 5 demonstrates this process. The red line in the graph represents the frequency values stored in the array. The y values are the frequencies, and the x values represent the index number the value was stored in. The blue horizontal lines represent the frequency changes between the DOT, BREAK, and DASH tones. The lower blue line is the frequency check between the DOT and BREAK tones. Anything below the lower blue line is a DOT frequency, and anything above the lower blue line is a BREAK frequency. The DASH frequency is represented by anything above the upper blue line, and anything below the upper blue line falls back into the BREAK frequency range.

The highlighted sections in graph 1 represent the noise areas. For example, the highlighted area to the far left crosses the upper blue frequency line, so when those frequencies were read during the analyzing of the array, it would be recorded as a DASH frequency and added to the array. However, it can be seen that the DASH frequency in the left highlighted area only lasts for very few indices, so this frequency spike can be attributed to noise in the room or distortion from the speakers, and it should be removed from the array. The same logic applies to the other highlighted examples.

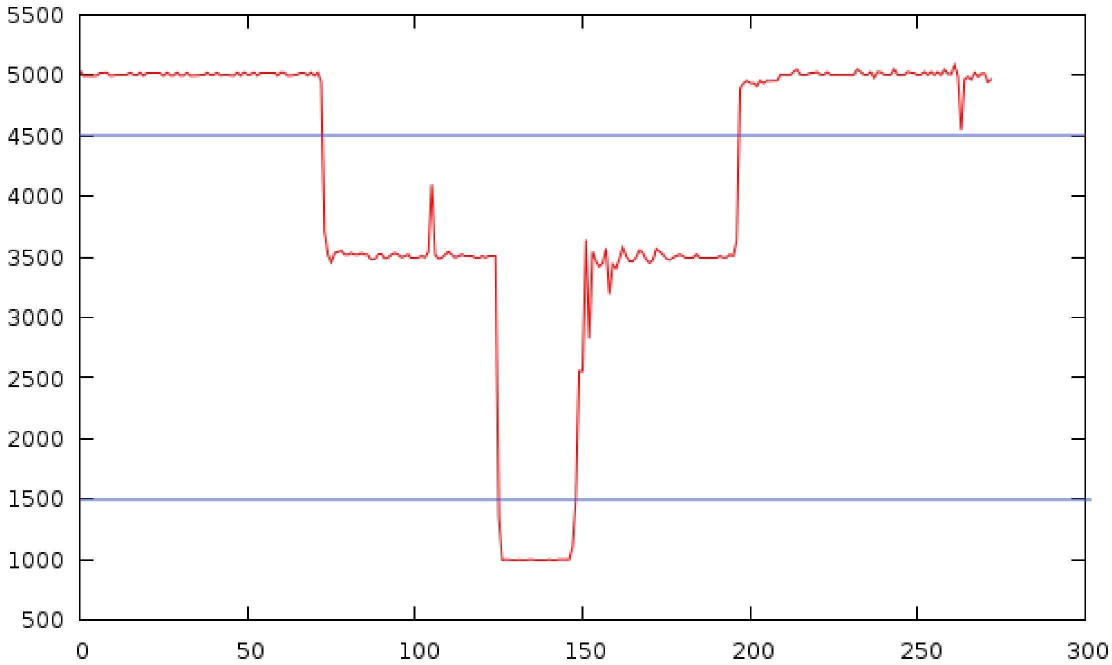
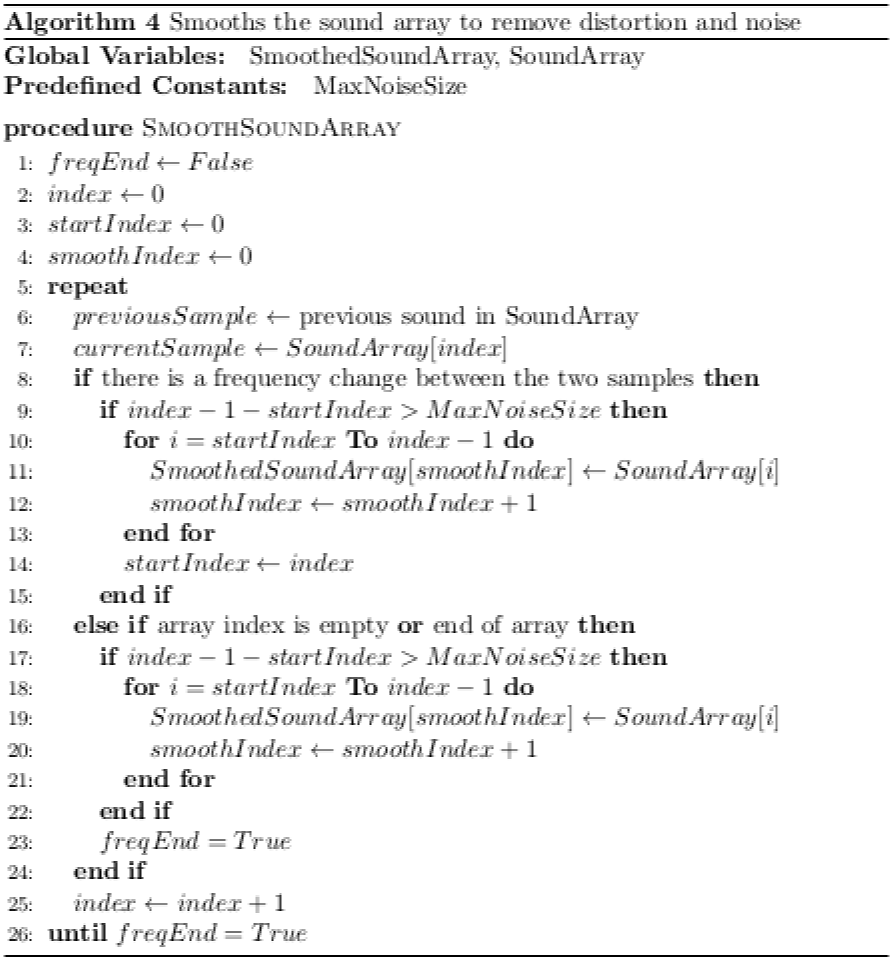


Illustration 6: Smoothed Frequency Graph

The resulting procedure of the array smoothing can be seen in the smoothed frequency graph in Illustration 6. The graph shows that all the previous highlighted areas have been smoothed to values that do not cross the blue frequency comparison lines. This new smoothed array will be used in place of the old non-smoothed sound array in the AnalyzeSoundArray procedure in algorithm 3.



```
Algorithm 4 Smooths the sound array to remove distortion and noise
Global Variables:   SmoothedSoundArray, SoundArray
Predefined Constants:   MaxNoiseSize

procedure SmoothSoundArray
 1: $freqEnd \leftarrow False$
 2: $index \leftarrow 0$
 3: $startIndex \leftarrow 0$
 4: $smoothIndex \leftarrow 0$
 5: repeat
 6:   $previousSample \leftarrow$ previous sound in SoundArray
 7:   $currentSample \leftarrow SoundArray[index]$
 8:   if there is a frequency change between the two samples then
 9:     if $index - 1 - startIndex > MaxNoiseSize$ then
10:       for $i = startIndex$ To $index - 1$ do
11:         $SmoothedSoundArray[smoothIndex] \leftarrow SoundArray[i]$
12:         $smoothIndex \leftarrow smoothIndex + 1$
13:       end for
14:       $startIndex \leftarrow index$
15:     end if
16:   else if array index is empty or end of array then
17:     if $index - 1 - startIndex > MaxNoiseSize$ then
18:       for $i = startIndex$ To $index - 1$ do
19:         $SmoothedSoundArray[smoothIndex] \leftarrow SoundArray[i]$
20:         $smoothIndex \leftarrow smoothIndex + 1$
21:       end for
22:     end if
23:     $freqEnd = True$
24:   end if
25:   $index \leftarrow index + 1$
26: until $freqEnd = True$
```

Illustration 7: Algorithm 4

Algorithm 4 contains the procedure for smoothing the array. Lines 1 through 4 initialize the various index variables used. Lines 5 through 26 contain the loop to process the whole sound array. Lines 6 and 7 store the previous and current values of the sound array. Line 8 checks for a frequency change between the two samples. If a frequency change is detected, lines 9 through 15 check the size of the frequency block to determine if it is large enough to be a message tone, and if it is, it is added to the array. If it too small, those frequencies are discarded. Line 16 through 24 is the special case to capture the message tones at the end of the array. It performs the same operations as the previous if statement in lines 8 through 15.

4.4 Message Passing

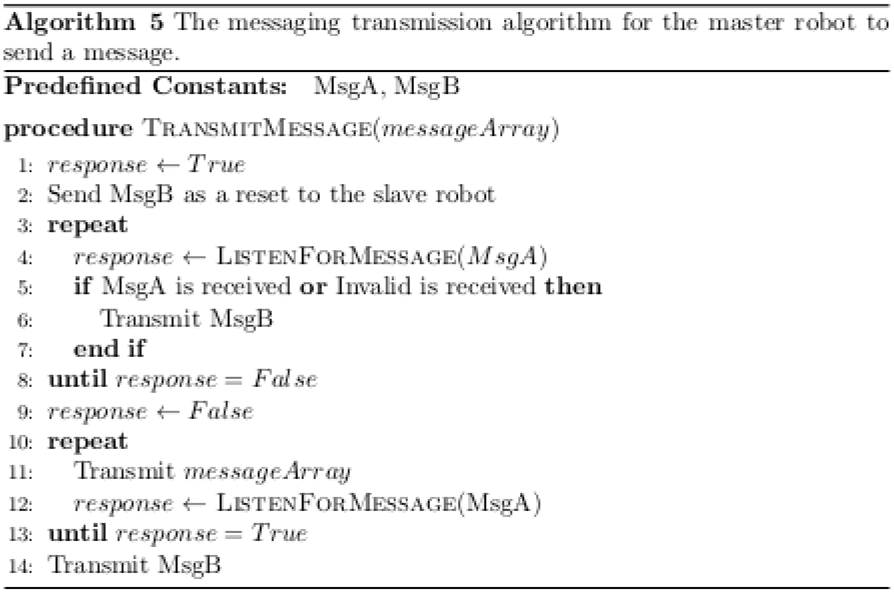
The robots now have the ability to send messages, but the receipt of a message is not guaranteed. Some of the possible errors have been corrected for, but other errors could still be introduced. In testing, the robots were able to transmit and receive a message the majority of the time, but on occasion, they would fail to receive one. Though these errors are rare, they must still be accounted for.

To account for these errors, the robots will not send a single message, but they will send a series of messages. These messages will represent the command the robot is to execute and a series of responses to confirm the message was received and the command was executed.

The messaging function for the robots is based on a send and receive loop. The robot sends a message tone and waits for a response tone. If a response tone is not received within a time limit, the message tone is transmitted again. This loop introduces a problem to transmitting the last tone. If using the loop, when the last tone has been transmitted, the robot is stuck in a state of waiting for a response. To counter this, rather than using a loop for the last tone message to be transmitted, a single message tone is sent. While this message is received the majority of the time, rare instances arise where the tone is missed. If the tone is missed, the robot expecting the tone is stuck in its final listening state where it is transmitting its message tone and listening for a response tone.

To account for this, when a robot is ready to transmit a new message, its first action it to send a reset response to break the listening robot out of its loop. This reset response is sent, and the robot listens to be sure the robot is no longer transmitting sounds, which means the robot has transitioned to its state of waiting for a command. If a sound is still received, the reset command is transmitted until the robot is no longer transmitting sound.

The master and slave robot use the same procedures for capturing a message as described in previous section. However, they have slightly different procedures for actually sending and acting on that message. The master robot must transmit a message, listen for the slave robots response, and send a response back to the slave robot. The slave robot must receive a message, perform the action associated with that message, and send a confirmation to the master robot.



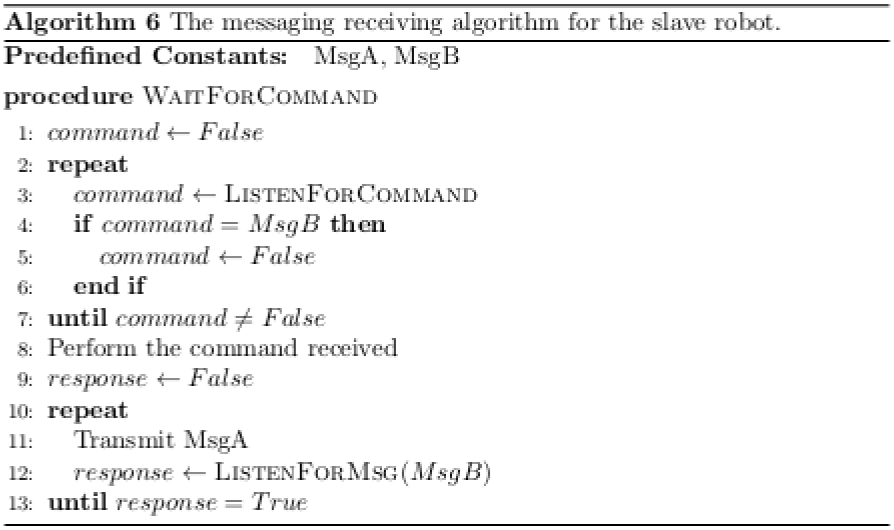
Algorithm 5 The messaging transmission algorithm for the master robot to send a message.

Predefined Constants: MsgA, MsgB

```
procedure TransmitMessage(messageArray)
 1: response ← True
 2: Send MsgB as a reset to the slave robot
 3: repeat
 4:     response ← ListenForMessage(MsgA)
 5:     if MsgA is received or Invalid is received then
 6:         Transmit MsgB
 7:     end if
 8: until response = False
 9: response ← False
10: repeat
11:     Transmit messageArray
12:     response ← ListenForMessage(MsgA)
13: until response = True
14: Transmit MsgB
```

Illustration 8: Algorithm 5

Algorithm 5 outlines the procedure the master robot uses to communicate with the slave robot. The master robot uses the predefined messages MsgA and MsgB as the response and confirmation tones. Line 2 sends a single MsgB reset tone to the slave robot in case the slave robot is stuck in his response loop as described previously. Lines 3 through 8 listens to be sure the slave robot is not transmitting and is now silently waiting for a command. In line 5, if MsgA or any sound is received, it is assumed that the robot is still emitting noise, and the reset message continues to be sent in line 6. Line 11 plays the passed in message through the speaker for the slave robot to receive. Line 12 listens for a MsgA response from the slave robot before continuing. If no MsgA response is received, the master robot continues the play message and listen for response loop from lines 10 through 13. Line 14 is the single response message that the master robot transmits to the slave robot to confirm that it has continued its scanning procedure. As discussed previously, it is only transmitted once. If the slave robot misses this message and gets stuck in its send and receive loop, lines 2 through 8 will reset the slave robot into its waiting state the next time a message is sent.



Algorithm 6 The messaging receiving algorithm for the slave robot.

```
Predefined Constants:   MsgA, MsgB
procedure WaitForCommand
 1: $command \leftarrow False$
 2: repeat
 3:    $command \leftarrow$ ListenForCommand
 4:    if $command = MsgB$ then
 5:       $command \leftarrow False$
 6:    end if
 7: until $command \neq False$
 8: Perform the command received
 9: $response \leftarrow False$
10: repeat
11:    Transmit MsgA
12:    $response \leftarrow$ ListenForMsg($MsgB$)
13: until $response = True$
```

Illustration 9: Algorithm 6

Algorithm 6 outlines the procedure for the slave robot waiting for a command to be received from the master robot. Line 1 initializes a bool variable used to check for messages. Lines 2 through 7 perform a loop to wait for the command sent from the master robot. Line 3 stores any message transmitted from the master robot using the ListenForCommand procedure. Line 4 through 6 are a check to ignore the MsgB reset that the master robot may be sending at the start of its message transmission procedure. Line 7 checks to see if a message value has been assigned to the command variable, and if it has, a command was received, and the loop can exit. Line 8 performs the command received from the previous loop. Lines 9 through 13 then begin the process to transmit the confirmation message to the master robot. Line 11 transmits the message, and line 12 waits for a response from the master robot. In line 13, the loop is repeated until a MsgB response is received from the master robot.

With all the previous procedures for analyzing and sending messages, the robots now have a communication system capable of transmitting and receiving messages. The message transmission system now has error checking built in using the smoothing algorithm and the message confirmation system.

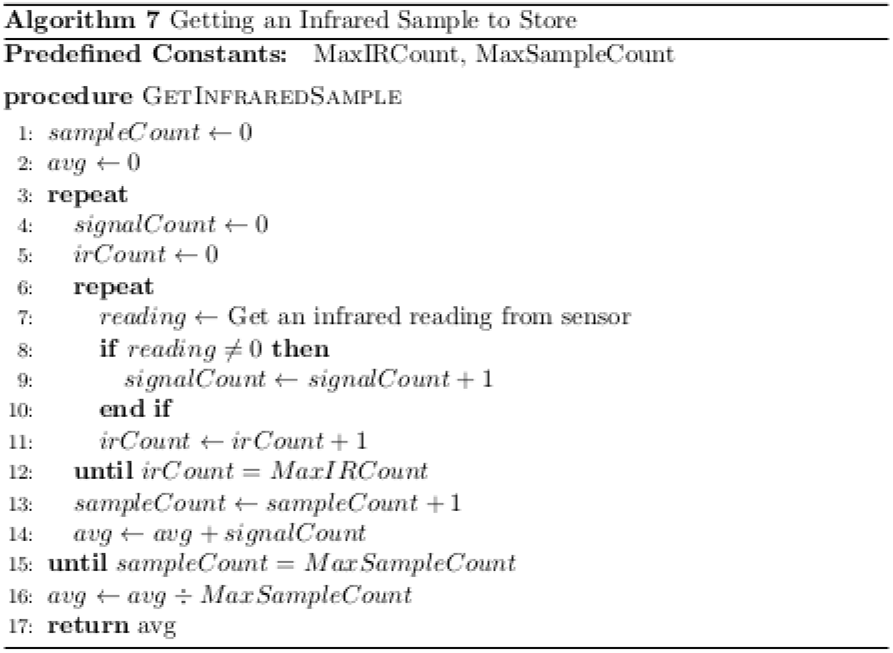
Chapter 5 Locating and Aligning the Robots

5.1 Infrared Sampling and Rotation

The next step in the solution to this problem is being able to detect the possible locations of a robot. This will be done using the infrared transmitter and receiver. As mentioned earlier, the infrared transmitters are aligned at slight angles, and are designed to aim infrared signals at itself rather than another robot. This could lead to some inaccuracies when trying to send infrared bursts to another robot. To deal with this inaccuracy, multiple samples will be taken.

When the sampling routine begins, the slave robot will constantly emit infrared signals. The master robot will take 1000 infrared readings and store the count. It will takes these 1000 readings a total of 20 times. Next, it will take the average of these 20 readings and store that value as the sample count for the current slice. Finally, it will rotate left to the next slice and repeat the sampling process.

Once it has completed the first complete 360 degree rotation, it now has infrared data for each direction that it rotated. Next, the master robot signals the slave robot to turn to a new slice degree. After the slave robot makes its rotation, the master robot begins it sampling routine again, but it rotations are to the right. This is done to offset the slippage problem that was discussed previously. If the robot made its rotations in the same direction each time, the effects of the slippage would be cumulative in one direction. By alternating the rotation directions, some of the slippage offsets. If the robot turns several extra degrees left on it's first rotation, any extra degrees that it turns on its right rotation will offset this to some amount and reduce the error in turning. The master and slave robot will continue this alternating sampling pattern until the slave robot has made a complete 360 degree turn. At that point, the master robot has the complete data set needed to make its decision.



Algorithm 7 Getting an Infrared Sample to Store

Predefined Constants: MaxIRCount, MaxSampleCount

```
procedure GetInfraredSample
$sampleCount \leftarrow 0$
$avg \leftarrow 0$
repeat
    $signalCount \leftarrow 0$
    $irCount \leftarrow 0$
    repeat
        $reading \leftarrow$ Get an infrared reading from sensor
        if $reading \neq 0$ then
            $signalCount \leftarrow signalCount + 1$
        end if
        $irCount \leftarrow irCount + 1$
    until $irCount = MaxIRCount$
    $sampleCount \leftarrow sampleCount + 1$
    $avg \leftarrow avg + signalCount$
until $sampleCount = MaxSampleCount$
$avg \leftarrow avg \div MaxSampleCount$
return avg
```

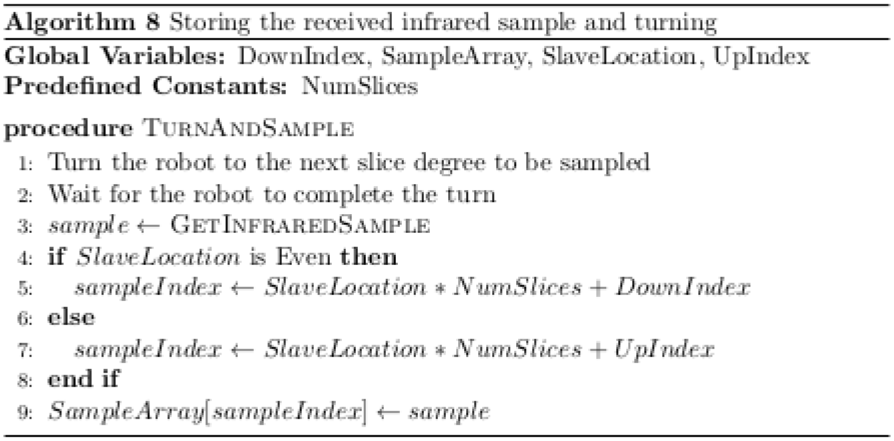
Illustration 10: Algorithm 7

Algorithm 7 outlines the process for taking an infrared sample. Lines 1 and 2 initialize the sample variables to 0. Lines 3 through 15 contain the loop to collect the 20 samples. Lines 6 through 12 take 1000 readings and total the number of IR signals that were detected. Line 14 adds the sample count of the infrared readings, and line 16 takes the average of those 20 samples before returning the value in line 17.

5.2 Storage of Data

As the robots perform their rotation and scanning, they need a way to store the data for future calculations. The SPIN programming language does not offer direct support for multi-dimensional arrays, so I chose to represent the data structure as a single continuous array. A 2D array will be simulated using a single array and math to manipulate the indexes.

As the master robot makes its first infrared sampling, it will begin storing the sample average in the array starting at the beginning. Since this implementation uses a slice size of 45 degrees, there will be 8 indexes for each slice the slave robot must rotate through. After it fills these 8 indexes, it will begin making its rotation in the alternate direction. The next 8 indexes will be filled in reverse order to make the data uniform and to make future calculations easier. With this order, the first index for a slice will always refer to 0 degrees, and the last index will always refer to 315 degrees.



Algorithm 8 Storing the received infrared sample and turning

Global Variables: DownIndex, SampleArray, SlaveLocation, UpIndex
Predefined Constants: NumSlices

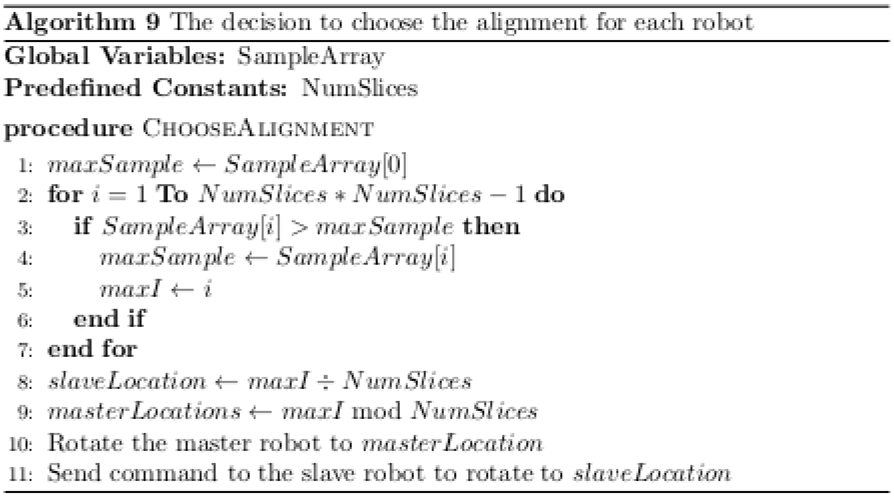
```
procedure TurnAndSample
1: Turn the robot to the next slice degree to be sampled
2: Wait for the robot to complete the turn
3: $sample \leftarrow$ GetInfraredSample
4: if $SlaveLocation$ is Even then
5:     $sampleIndex \leftarrow SlaveLocation * NumSlices + DownIndex$
6: else
7:     $sampleIndex \leftarrow SlaveLocation * NumSlices + UpIndex$
8: end if
9: $SampleArray[sampleIndex] \leftarrow sample$
```

Illustration 11: Algorithm 8

Algorithm 8 outlines the actual sampling procedure used to gather and store the data for a single slice. Line 1 turns the robot to the next slice to be sampled. Line 2 and 3 collects the infrared samples once the robot has completed the turn. Lines 4 through 8 determines what index in the infrared storage array the sample should be stored in. These lines make use of the global variables DownIndex, SlaveLocation, and UpIndex. These three variables are count variables that are incremented with values from other functions. UpIndex and DownIndex are incremented up and down respectively whenever the robot is turned. They are used to keep track of the next index a sample should be stored in. SlaveLocation is a variable that is incremented every time the slave robot is messaged to perform a turn. It tracks the current slice that the slave robot is in. If this slave location is even, as is checked in line 4, the master robot is performing its left turn and the sample can be stored in the array in order. If the slave location is odd, as is checked with the else in line 6, the master robot is performing its right turn, so the values must be stored in reverse order. This is done to make the data uniform and the alignment calculation easier to perform. Once the correct index has been calculated, the sample is stored in the sample array at that index in line 9.

5.3 Alignment Decision

After the sampling of all the data has been completed, the final step is for the master robot to make a decision on which directions the two robots should rotate to in order to complete the alignment. To make the decision, the master robot must process its data storage array. It simply scans the array to locate the largest infrared sample. Depending on the index, it now knows which slice each robot should face.



```
**Algorithm 9** The decision to choose the alignment for each robot
**Global Variables:** SampleArray
**Predefined Constants:** NumSlices
**procedure** ChooseAlignment
$maxSample \leftarrow SampleArray[0]$
**for** $i = 1$ **To** $NumSlices * NumSlices - 1$ **do**
    **if** $SampleArray[i] > maxSample$ **then**
        $maxSample \leftarrow SampleArray[i]$
        $maxI \leftarrow i$
    **end if**
**end for**
$slaveLocation \leftarrow maxI \div NumSlices$
$masterLocations \leftarrow maxI \bmod NumSlices$
Rotate the master robot to $masterLocation$
Send command to the slave robot to rotate to $slaveLocation$
```

Illustration 12: Algorithm 9

Algorithm 9 outlines the procedure to select the master and slave robot alignment.

In line 1, the max sample is set to the value in the first index of the sample array. In lines 1 through 7, the entire array is scanned making comparisons to find the highest value in the array. When that value is found, line 5 stores that index value. This index and value represent the largest infrared sample count that was stored during the scanning and rotation process. In line 8, the slice location the slave robot should rotate to is calculated. To get this slice, the index of the maximum sample is divided by the total number of slices. The integer value of this result is the slice the slave robot was located in when the maximum infrared sample was taken. In line 9, the slice location the master robot should rotate to is calculated. To get this slice, the modulus of the index and total number of slices is taken. The resulting value is the slice the master robot was located in when the maximum sample was taken. In line 10, the master robot is rotated to its corresponding slice. In line 11, the master robot sends a command to the slave robot instructing it to turn to its calculated slice. Once this algorithm is complete, the robots should be in an alignment facing each other, and the alignment routine is complete.

Chapter 6 Data and Results

To test this implementation. A total of 10 experiments were run and data collected. The tests were broken down into two groups. The first group of tests used a non-reflective cardboard box as an obstacle. The second group of test used a reflective mirror as an obstacle. The robots were then aligned in several different positions with the same positions being used for each reflective and non-reflective test. The starting position of each robot relative to the obstacle was recorded. The robots were then allowed to run their alignment routine. The infrared sample counts from the routine were recorded. After the robots completed the alignment routine, their final position was recorded. Images of the starting and ending positions are also included for the results of each test.

The following charts are the results of the experiments. The first section of the chart includes the master and slave robots starting alignment in reference to the obstacle. The second section of the chart includes a chart of the infrared samples that each robot collected in each slice. The third section includes images of both robots starting and ending alignments. The fourth and final section shows the robots finishing alignment in reference to their original starting position.

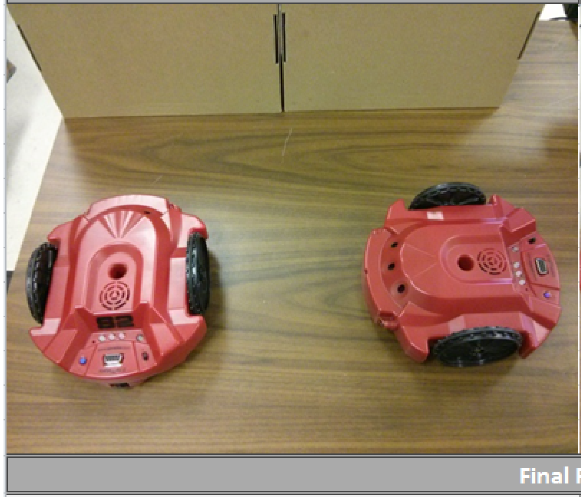
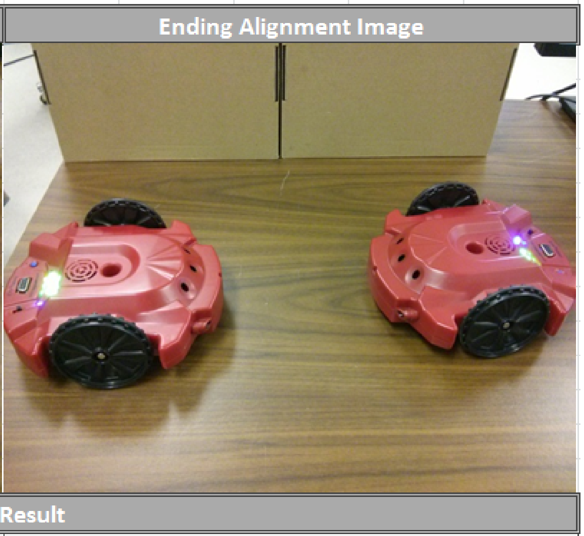
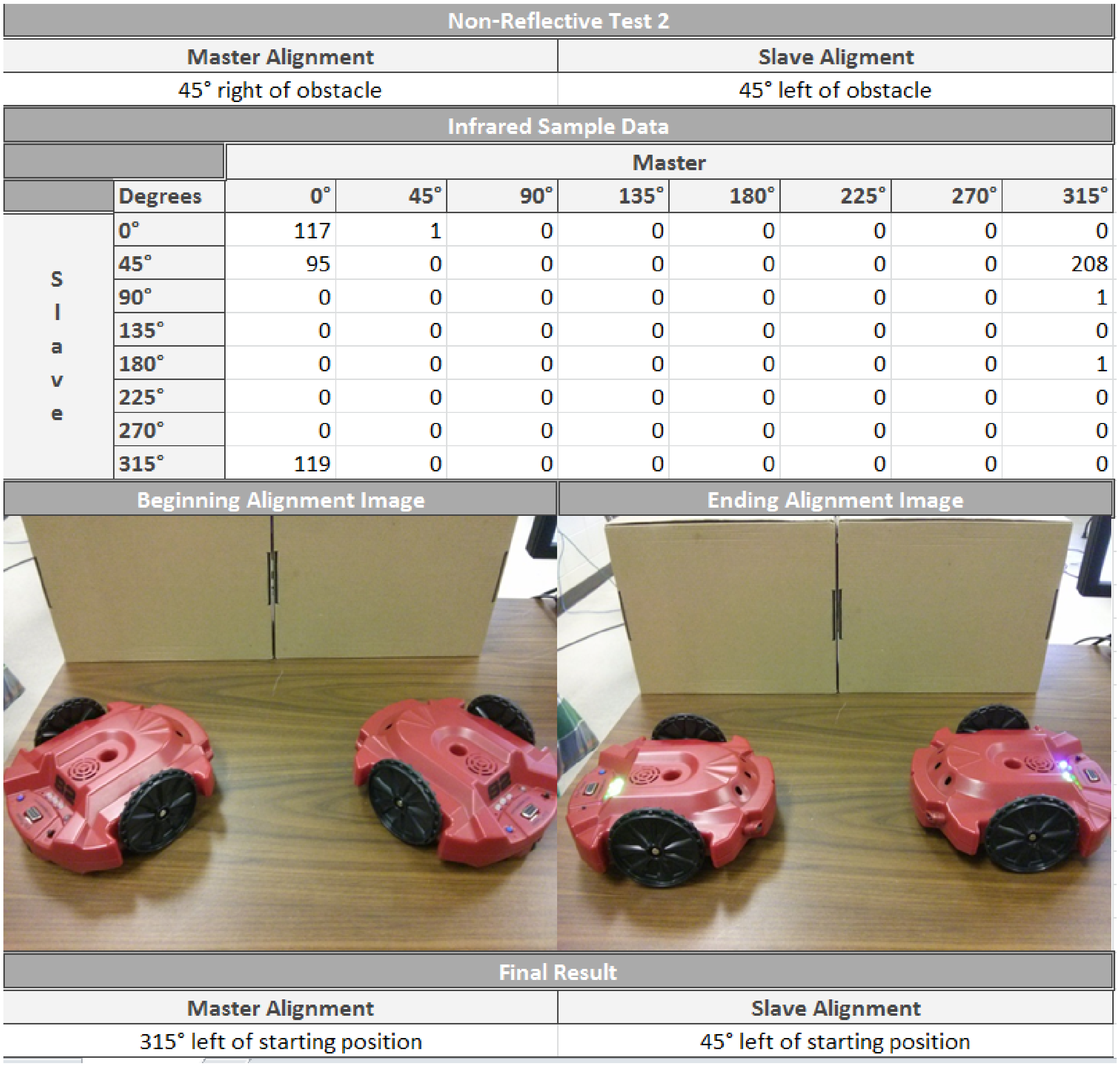
<table>
<tr><th colspan="10">Non-Reflective Test 1</th></tr>
<tr><td colspan="5">Master Alignment</td><td colspan="5">Slave Aligment</td></tr>
<tr><td colspan="5">0° facing obstacle</td><td colspan="5">90° left of obstacle</td></tr>
<tr><th colspan="10">Infrared Sample Data</th></tr>
<tr><td colspan="2"></td><td colspan="8">Master</td></tr>
<tr><td></td><td>Degrees</td><td>0°</td><td>45°</td><td>90°</td><td>135°</td><td>180°</td><td>225°</td><td>270°</td><td>315°</td></tr>
<tr><td rowspan="8">S
l
a
v
e</td><td>0°</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>84</td><td>221</td><td>0</td></tr>
<tr><td>45°</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>90°</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>8</td><td>0</td><td>0</td></tr>
<tr><td>135°</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>8</td><td>0</td></tr>
<tr><td>180°</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>2</td><td>0</td><td>0</td></tr>
<tr><td>225°</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>270°</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>111</td></tr>
<tr><td>315°</td><td>173</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><th colspan="5">Beginning Alignment Image</th><th colspan="5">Ending Alignment Image</th></tr>
<tr><td colspan="5"></td><td colspan="5"></td></tr>
<tr><th colspan="10">Final Result</th></tr>
<tr><td colspan="5">Master Alignment</td><td colspan="5">Slave Alignment</td></tr>
<tr><td colspan="5">270° left of starting position</td><td colspan="5">original starting position at 0°</td></tr>
</table>

Illustration 13: Non-Reflective Test Result 1



Non-Reflective Test 2

Master Alignment	Slave Aligment
45° right of obstacle	45° left of obstacle

Infrared Sample Data

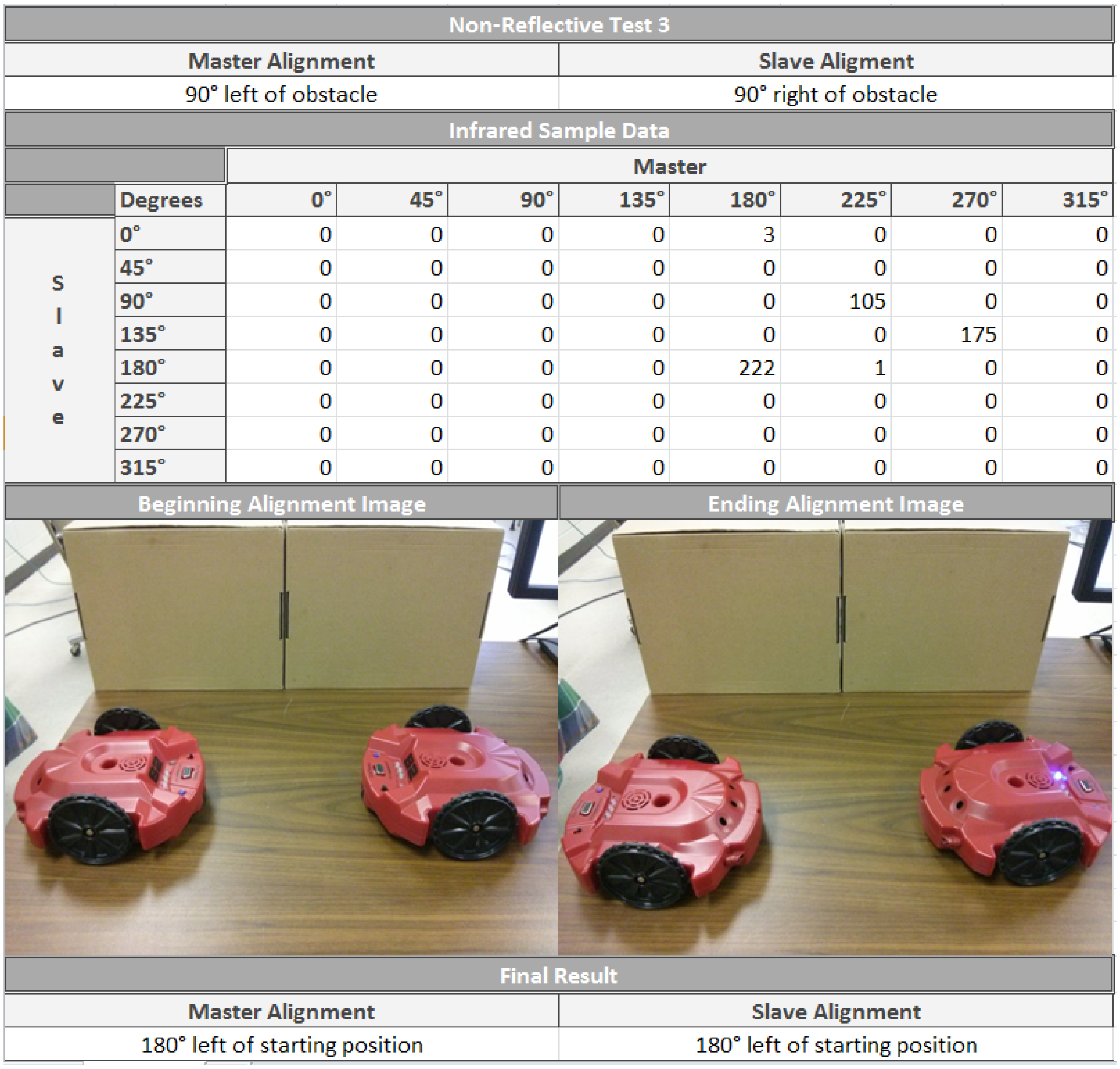
		Master							
	Degrees	0°	45°	90°	135°	180°	225°	270°	315°
Slave	0°	117	1	0	0	0	0	0	0
	45°	95	0	0	0	0	0	0	208
	90°	0	0	0	0	0	0	0	1
	135°	0	0	0	0	0	0	0	0
	180°	0	0	0	0	0	0	0	1
	225°	0	0	0	0	0	0	0	0
	270°	0	0	0	0	0	0	0	0
	315°	119	0	0	0	0	0	0	0

Beginning Alignment Image	Ending Alignment Image

Final Result

Master Alignment	Slave Alignment
315° left of starting position	45° left of starting position

Illustration 14: Non-Reflective Test Result 2



Non-Reflective Test 3	
Master Alignment	Slave Aligment
90° left of obstacle	90° right of obstacle

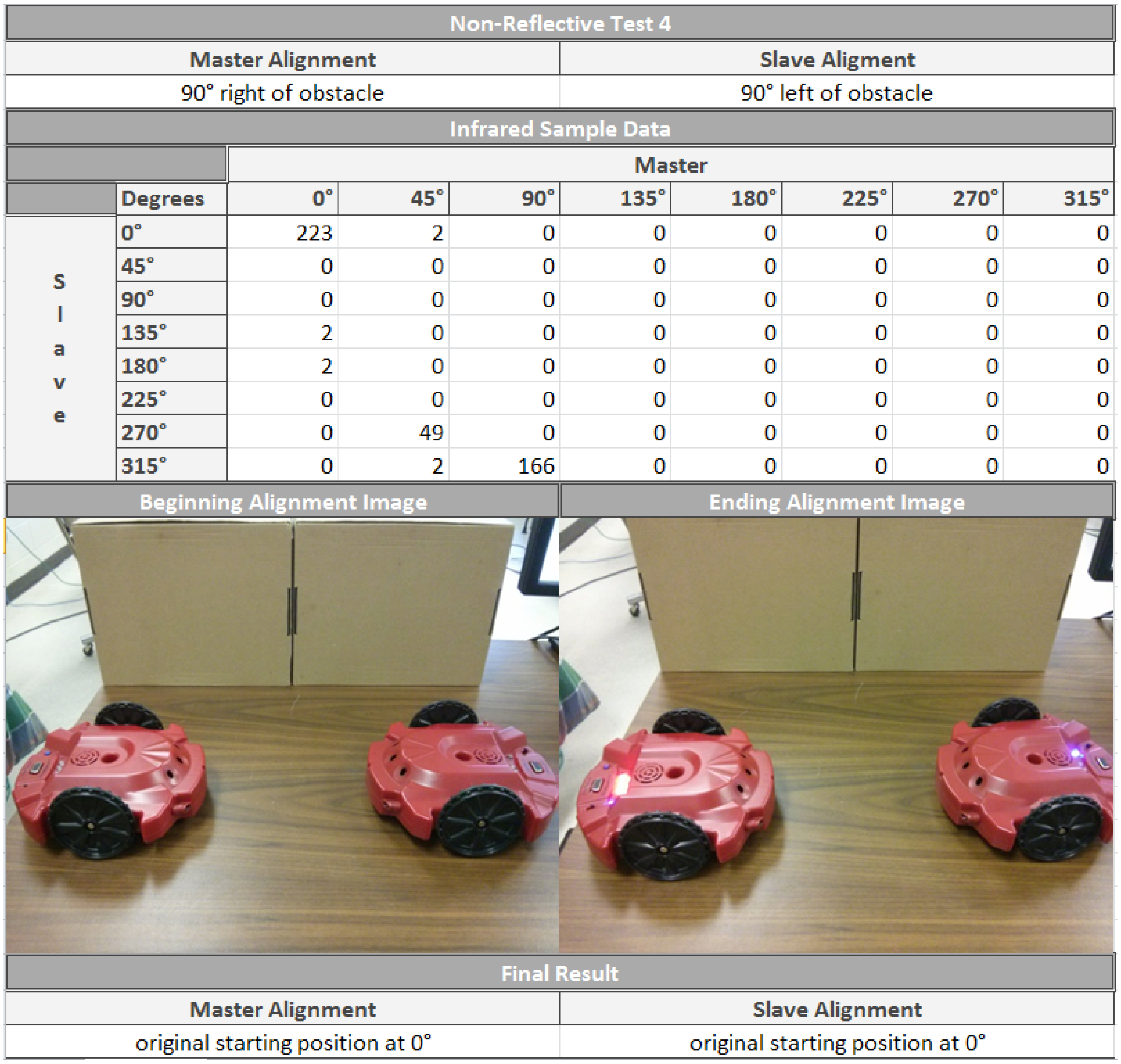
Infrared Sample Data

		Master							
	Degrees	0°	45°	90°	135°	180°	225°	270°	315°
Slave	0°	0	0	0	0	3	0	0	0
	45°	0	0	0	0	0	0	0	0
	90°	0	0	0	0	0	105	0	0
	135°	0	0	0	0	0	0	175	0
	180°	0	0	0	0	222	1	0	0
	225°	0	0	0	0	0	0	0	0
	270°	0	0	0	0	0	0	0	0
	315°	0	0	0	0	0	0	0	0

Beginning Alignment Image
Ending Alignment Image

Final Result	
Master Alignment	Slave Alignment
180° left of starting position	180° left of starting position

Illustration 15: Non-Reflective Test Result 3



Non-Reflective Test 4	
Master Alignment	**Slave Aligment**
90° right of obstacle	90° left of obstacle

Infrared Sample Data

		Master							
	Degrees	**0°**	**45°**	**90°**	**135°**	**180°**	**225°**	**270°**	**315°**
Slave	**0°**	223	2	0	0	0	0	0	0
	45°	0	0	0	0	0	0	0	0
	90°	0	0	0	0	0	0	0	0
	135°	2	0	0	0	0	0	0	0
	180°	2	0	0	0	0	0	0	0
	225°	0	0	0	0	0	0	0	0
	270°	0	49	0	0	0	0	0	0
	315°	0	2	166	0	0	0	0	0

Beginning Alignment Image	Ending Alignment Image

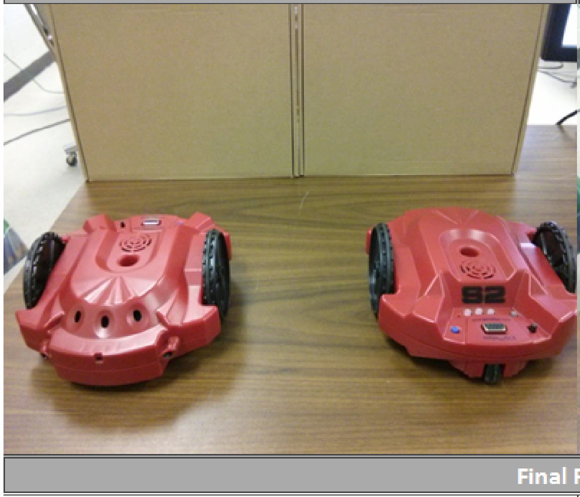
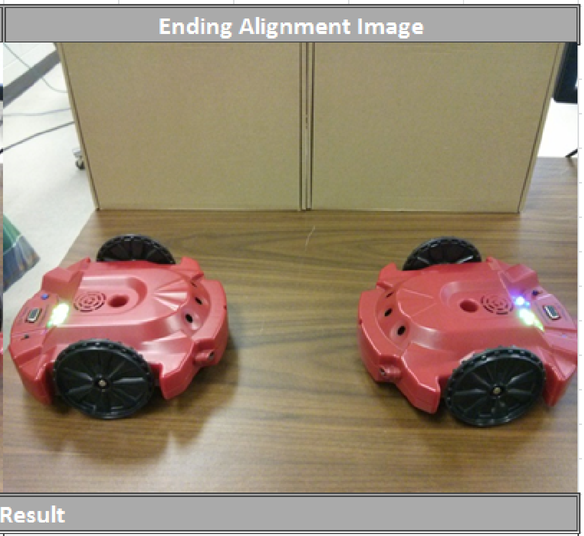
Final Result

Master Alignment	Slave Alignment
original starting position at 0°	original starting position at 0°

Illustration 16: Non-Reflective Test Result 4

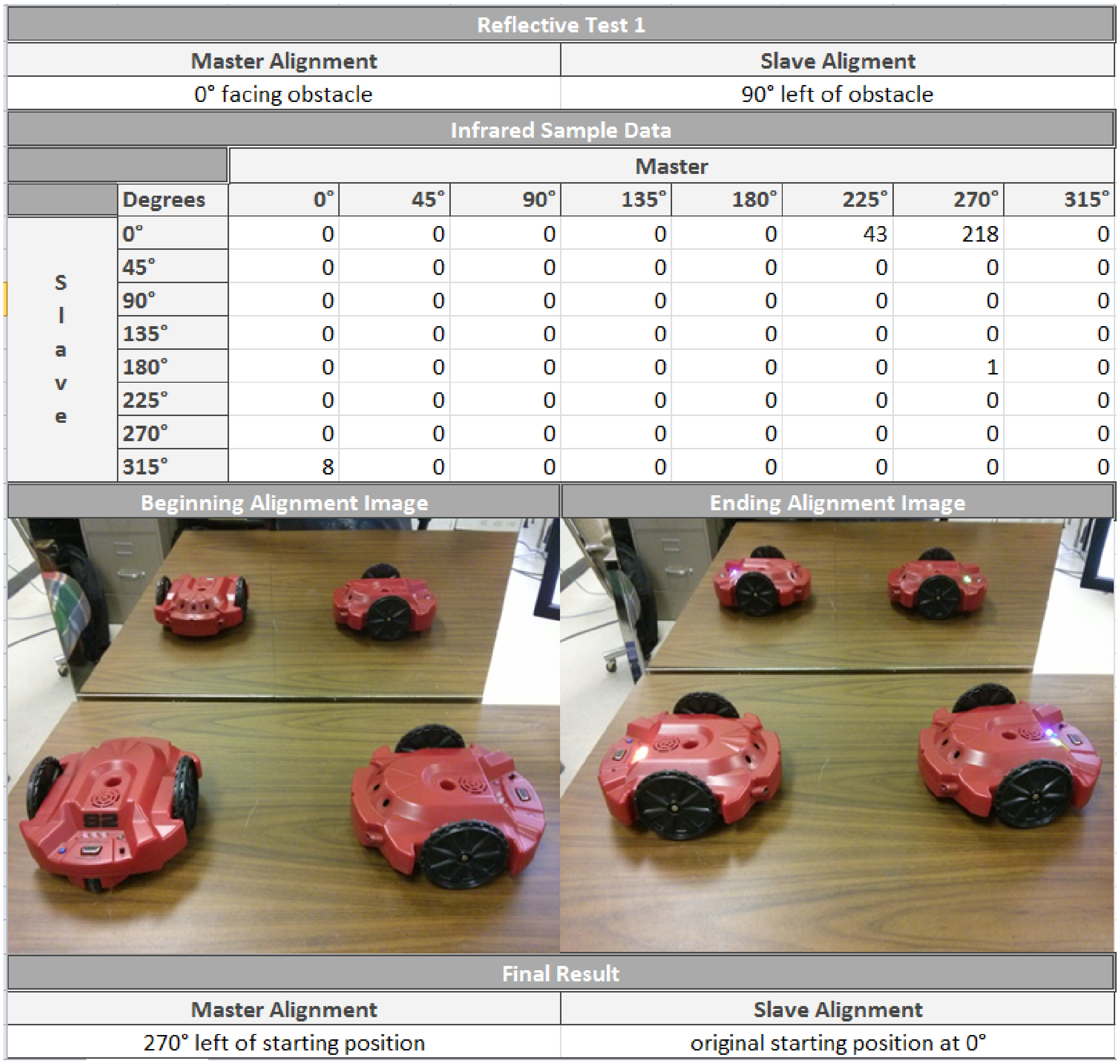
Non-Reflective Test 5	
Master Alignment	**Slave Aligment**
180° left of obstacle	0° facing obstacle

Infrared Sample Data									
		Master							
	Degrees	0°	45°	90°	135°	180°	225°	270°	315°
S l a v e	0°	0	0	0	121	0	0	0	0
	45°	0	0	0	0	187	0	0	0
	90°	0	71	220	0	0	0	0	0
	135°	12	0	0	0	0	0	0	0
	180°	0	0	0	0	0	0	0	0
	225°	0	0	18	0	0	0	0	0
	270°	0	0	0	0	0	0	0	0
	315°	0	0	0	0	0	0	0	0

Beginning Alignment Image	Ending Alignment Image
	

Final Result	
Master Alignment	**Slave Alignment**
90° left of starting position	90° left of starting position

Illustration 17: Non-Reflective Test Result 5



Reflective Test 1

Master Alignment	Slave Aligment
0° facing obstacle	90° left of obstacle

Infrared Sample Data

		Master							
	Degrees	0°	45°	90°	135°	180°	225°	270°	315°
Slave	0°	0	0	0	0	0	43	218	0
	45°	0	0	0	0	0	0	0	0
	90°	0	0	0	0	0	0	0	0
	135°	0	0	0	0	0	0	0	0
	180°	0	0	0	0	0	0	1	0
	225°	0	0	0	0	0	0	0	0
	270°	0	0	0	0	0	0	0	0
	315°	8	0	0	0	0	0	0	0

Beginning Alignment Image	Ending Alignment Image

Final Result

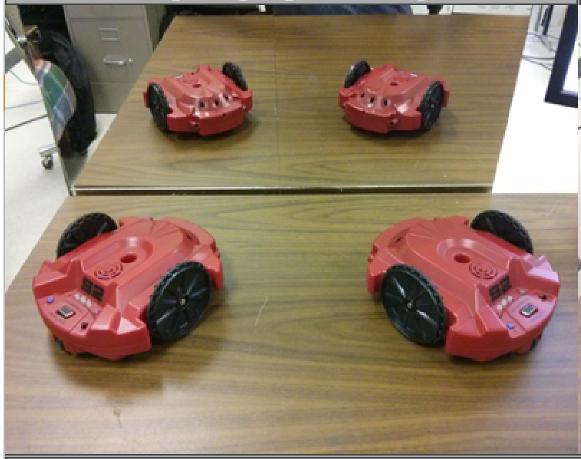
Master Alignment	Slave Alignment
270° left of starting position	original starting position at 0°

Illustration 18: Reflective Test Result 1

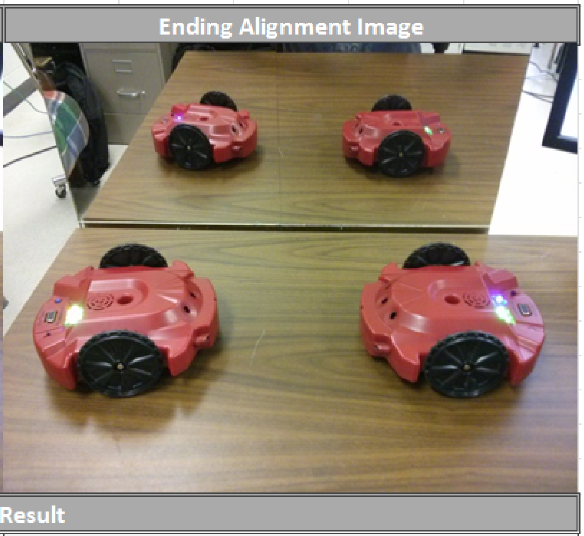
Reflective Test 2	
Master Alignment	Slave Aligment
45° right of obstacle	45° left of obstacle

Infrared Sample Data									
		Master							
	Degrees	0°	45°	90°	135°	180°	225°	270°	315°
Slave	0°	215	0	0	0	0	0	0	0
	45°	24	0	0	0	0	0	0	221
	90°	0	0	0	0	0	0	0	0
	135°	0	0	0	0	0	0	0	0
	180°	0	0	0	0	0	0	0	21
	225°	0	0	0	0	0	0	0	0
	270°	0	0	0	0	0	0	0	0
	315°	0	0	0	0	0	0	0	0

Beginning Alignment Image

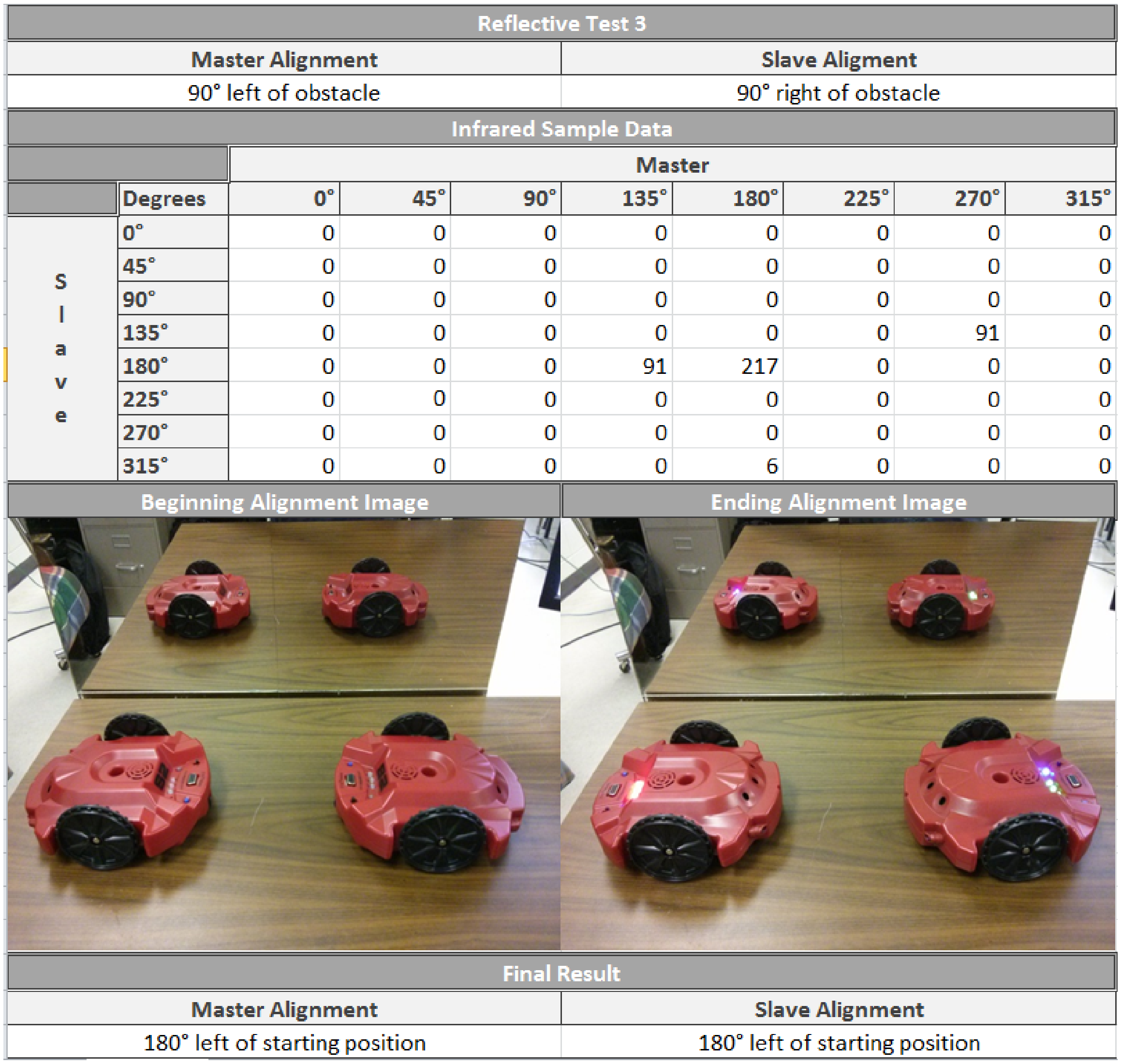


Ending Alignment Image



Final Result	
Master Alignment	Slave Alignment
315° left of starting position	45° left of starting position

Illustration 19: Reflective Test Result 2



Reflective Test 3

Master Alignment	Slave Aligment
90° left of obstacle	90° right of obstacle

Infrared Sample Data

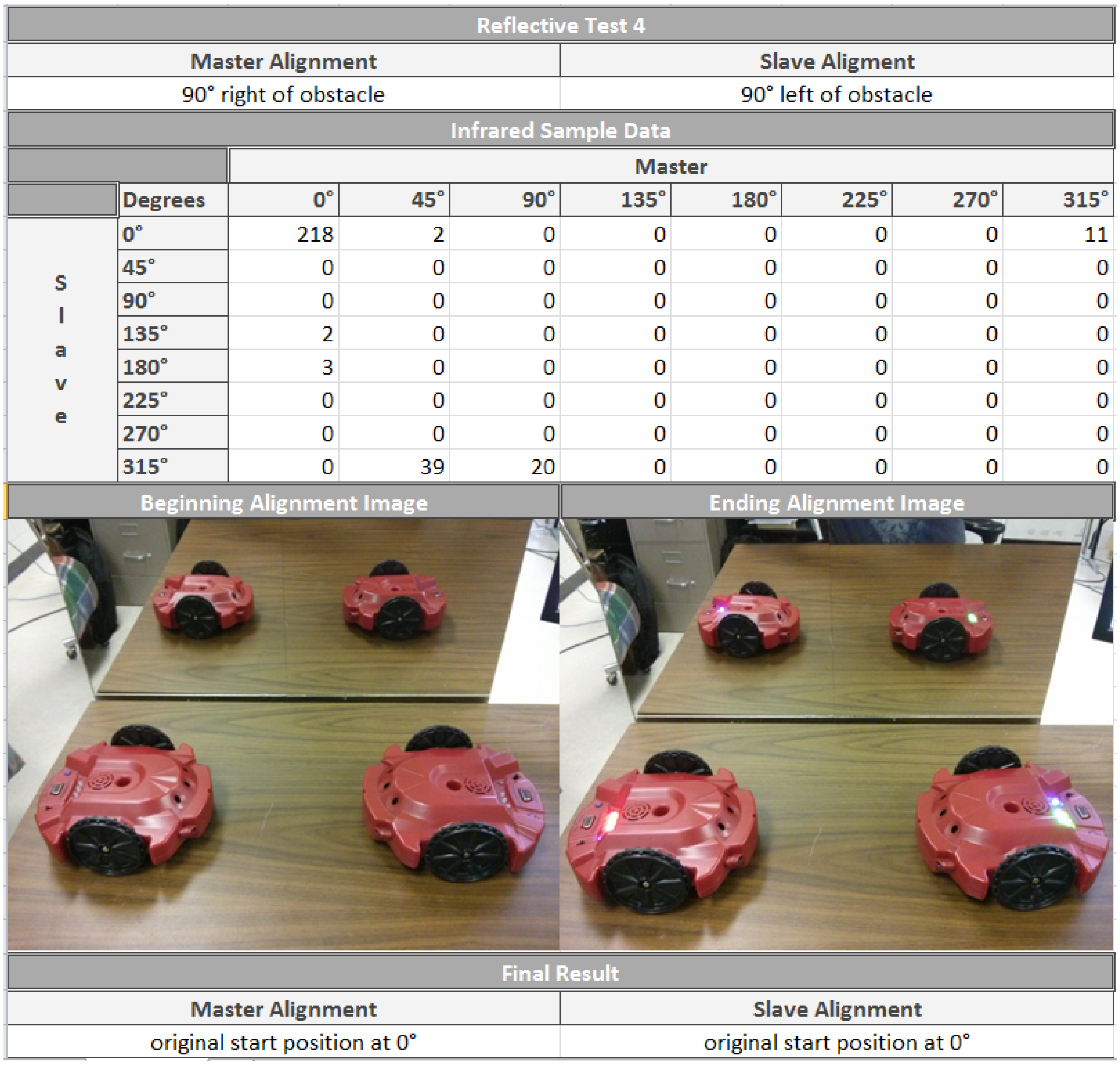
		Master							
	Degrees	0°	45°	90°	135°	180°	225°	270°	315°
Slave	0°	0	0	0	0	0	0	0	0
	45°	0	0	0	0	0	0	0	0
	90°	0	0	0	0	0	0	0	0
	135°	0	0	0	0	0	0	91	0
	180°	0	0	0	91	217	0	0	0
	225°	0	0	0	0	0	0	0	0
	270°	0	0	0	0	0	0	0	0
	315°	0	0	0	0	6	0	0	0

Beginning Alignment Image	Ending Alignment Image

Final Result

Master Alignment	Slave Alignment
180° left of starting position	180° left of starting position

Illustration 20: Reflective Test Result 3



Reflective Test 4	
Master Alignment	Slave Aligment
90° right of obstacle	90° left of obstacle

Infrared Sample Data

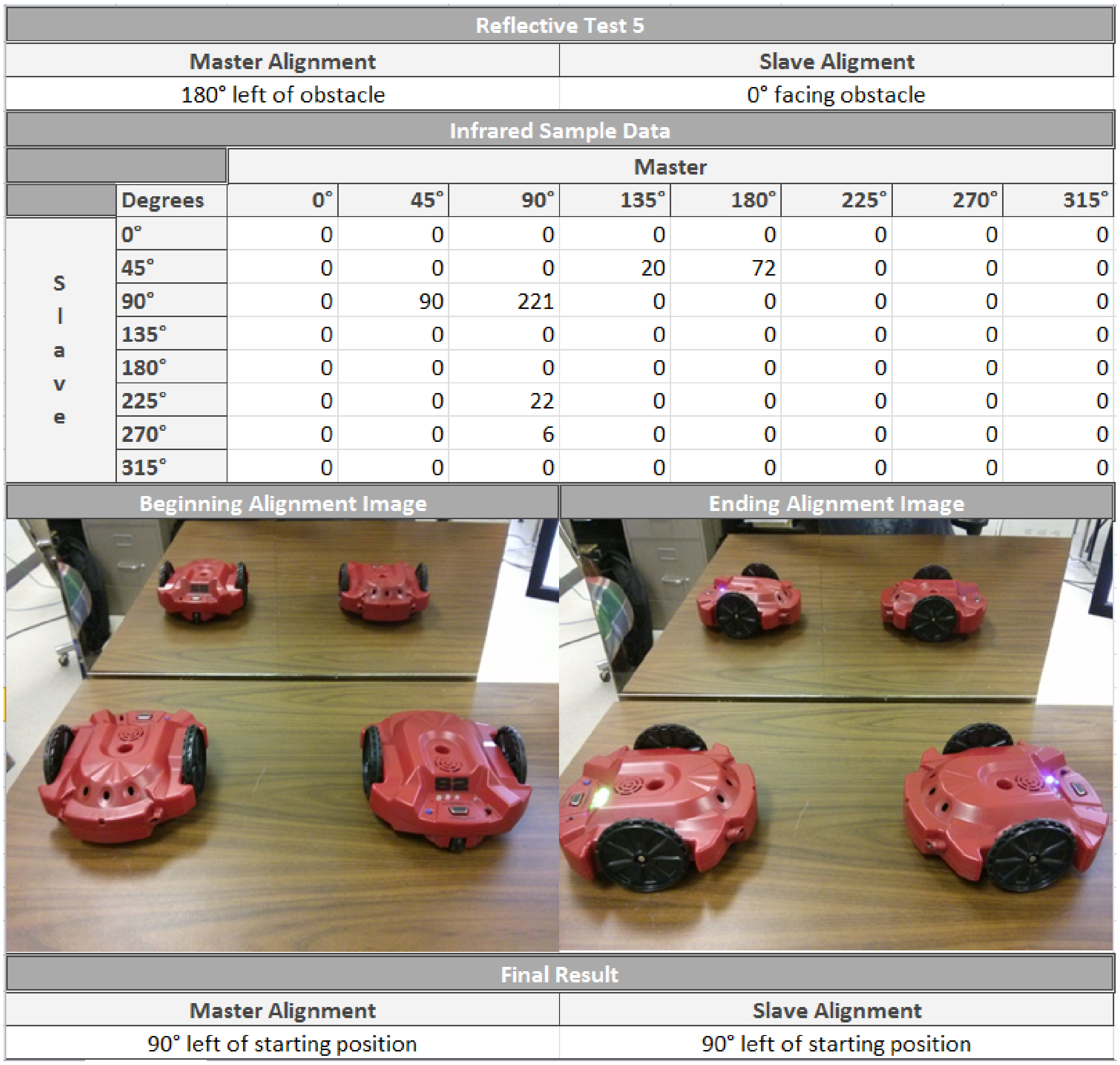
		Master							
	Degrees	0°	45°	90°	135°	180°	225°	270°	315°
Slave	0°	218	2	0	0	0	0	0	11
	45°	0	0	0	0	0	0	0	0
	90°	0	0	0	0	0	0	0	0
	135°	2	0	0	0	0	0	0	0
	180°	3	0	0	0	0	0	0	0
	225°	0	0	0	0	0	0	0	0
	270°	0	0	0	0	0	0	0	0
	315°	0	39	20	0	0	0	0	0

Beginning Alignment Image	Ending Alignment Image

Final Result

Master Alignment	Slave Alignment
original start position at 0°	original start position at 0°

Illustration 21: Reflective Test Result 4



Reflective Test 5	
Master Alignment	**Slave Aligment**
180° left of obstacle	0° facing obstacle

Infrared Sample Data

		Master							
	Degrees	0°	45°	90°	135°	180°	225°	270°	315°
Slave	0°	0	0	0	0	0	0	0	0
	45°	0	0	0	20	72	0	0	0
	90°	0	90	221	0	0	0	0	0
	135°	0	0	0	0	0	0	0	0
	180°	0	0	0	0	0	0	0	0
	225°	0	0	22	0	0	0	0	0
	270°	0	0	6	0	0	0	0	0
	315°	0	0	0	0	0	0	0	0

Beginning Alignment Image	Ending Alignment Image

Final Result	
Master Alignment	**Slave Alignment**
90° left of starting position	90° left of starting position

Illustration 22: Reflective Test Result 5

Chapter 7 Conclusion

The robots aligned successfully in all of the tests. Each robot started from its starting alignment shown in the picture from each test result. It then began it's sampling routine. There were several times throughout the test when the robots were unable to receive a message on the first try, but with the response confirmation system, they eventually received the message that was needed for them to continue the routine. Once the sampling routine was finished, the master robot correctly chose and communicated the slice that would put both robots into alignment. Both robots then executed a rotation to put them in alignment facing each other.

Even though the robots were able to align correctly in all the tests, one of the data results was only successful by a small margin. This was an instance where the successful alignment decision was separated by a difference in the single digits as demonstrated in the reflective test 2. In reflective test 2, the master robot detected a sample count of 215 at 0 degrees and a sample count of 221 at 315 degrees. The correct alignment was at 315 degrees, and the 0 degree count was from both robots being directly aligned with each others reflections in the mirror. With sample counts this close, I believe there would be the possibility of the sample count from a reflected object being higher than the sample count directly from the robot. It might be a rare occurrence, but a modified solution would still need to be considered.

References

[1] Roufas, K., Y. Zhang, D. Duff, M. Yim. “Six Degree of Freedom Sensing for Docking Using IR LED Emitters and Receivers.” in The 7th Intl. Symp. On Experimental Robotics. 2000. Hawaii.

[2] Quiňonez, Y., Baca, J., De Lope, J., Ferre, M., Aracil, R., “Self-Alignment Approach Based on Cooperative Behaviors for the Docking Process of Modular Mobile Robots.” in Electronics, Robotics, and Automotive Mechanics Conference (CERMA), 2010, vol., no., pp.445,450, Sept. 28 2010-Oct. 1 2010

[3] Shen, Wei-Min., Will, P., “Docking in Self-Reconfigurable Robots.” in Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on, vol. 2, no., pp.1049, 1054 vol. 2, 2001

[4] Delrobaei, M., McIsaac, K., “An Infrared Docking System for Modular Wheeled Mobile Robots,” in 5th Symposium on Advances in Science and Technology, 2011

[5] Martin, Jeff. *Propeller Manual Version 1.1*. : , . Print. "Propeller Object Exchange." *Propeller Object Exchange*. N.p., n.d. Web. . <http://obex.parallax.com/>.

[6] "Scribbler 2 (S2) Robot - USB." *Parallax Inc*. N.p., n.d. Web. . <http://www.parallax.com/product/28136>.