

# Evaluation of Gesture-Based Controls for Robotic Systems

Lauren Frazier

May 3, 2012

# Contents

<b>Acknowledgements</b>	<b>iv</b>
<b>Abstract</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Related Work</b>	<b>3</b>
<b>3 Implementation of Control Systems</b>	<b>5</b>
3.1 XBOX 360 Controller . . . . .	5
3.2 Cellbots Application . . . . .	6
3.3 Gesture-Based Control Application . . . . .	7
<b>4 Human Factors Experiment Design</b>	<b>9</b>
4.1 Platform and Hardware . . . . .	9
4.2 Track Design . . . . .	10
4.3 Questionnaire Design . . . . .	10
4.4 Experiment Procedure . . . . .	11
<b>5 Results</b>	<b>12</b>
5.1 Time Trials . . . . .	12
5.2 Questionnaire . . . . .	13
5.3 Analysis and Hypotheses . . . . .	14
<b>6 Conclusion</b>	<b>19</b>

# List of Tables

5.1	Average time taken to complete the course, average number of Line Faults, and average number of Mistake Acknowledgements for each of the four controller types. . . . .	12
-----	---	----

# List of Figures

5.1	Numbers indicate rank given to the controller for using it to accomplish a critical task. The XBOX 360 controller was the most popular for accomplishing a critical task; the gesture-based controls were the least popular. . . . .	15
5.2	Numbers indicate rank given to the controller for using it for two hours continuously. The XBOX 360 controller was the most popular for using continuously for two hours; the gesture-based controls were the least popular. . . . .	16
5.3	The tilt controller was ranked the most intuitive, while the gesture-based controller was ranked the least intuitive. . . . .	17
5.4	The tilt controller was ranked the most fun to use, while the D-Pad and gesture-based controls tied for 3rd place. . . . .	18

# Acknowledgements

These are acknowledgements.

# Abstract

Robotic control systems are becoming more common, especially in the military, and there is a growing need for better control interface. Arm and hand gestures are typical human forms of communication, so applying that to a robotic control system can yield a more intuitive system. With military applications, there are lives at stake, so having the most efficient, intuitive control system can make a large difference in the success of a mission and the safety of the soldiers involved. The work in [11] describes a gesture based control system that uses the Nintendo Wiimote to determine arm/hand gestures and control a robot. I propose to create a gesture based system using a smartphone and conduct an experiment similar to the work in [11], collecting survey data from the participants on the the effectiveness and ease of use of each system. This thesis aims to test the effectiveness/ease of use of smartphone gesture-based robotic control systems vs. traditional control systems.

# Chapter 1

## Introduction

The need for human-robot interfaces is increasing rapidly. The military has already begun using unmanned vehicles in several different arenas (air, ground, water). In order to develop the most efficient human-robot interface, we turned to a traditional form of human-human communication, arm and hand gestures. Arm and hand gestures are typical human forms of communication, so applying that to a robotic control system can yield a more intuitive system. With military applications, there are lives at stake, so having the most efficient, intuitive control system can make a large difference in critical moments and improve the safety of those involved. A more intuitive system will also reduce the training time and expenses for the operators of the vehicle.

The work in [11] describes a gesture based control system that uses the Nintendo Wiimote to determine arm/hand gestures and control a robot. They then conducted a study where subjects used Wiimote gesture system and a more standard system and filled out a survey to indicate how effective the Wiimote system was as compared to the standard control system for the robot.

I propose a gesture based system using a smartphone and conducted a human factors experiment similar to [11]. The system uses a Samsung Galaxy S II phone for the gesture-based input, a tilt-based controller, and a touch screen D-Pad controller, and a Microsoft XBOX 360 controller for the more traditional input. Subjects used each of the four controls in a random order to guide an iRobot Roomba vacuum cleaner through a short time trial course. The raw data (video and observations during the time trials) was analyzed with metrics like the time required to complete the course, number of times the subject went outside the course boundaries, number of times the subject acknowledged making a mistake, etc. After the experiment is complete, subjects also filled out a survey, and both sets of data were used

to determine which control scheme is more effective and intuitive.

This thesis aims to test both the perceived and actual effectiveness and ease of use of smartphone gesture-based robotic control systems vs. traditional control systems.

This thesis introduces an inexpensive gesture based control system that uses commercial, off the shelf hardware (an Android smartphone and a Roomba vacuum cleaner). It also provides data showing both subjects' objective performance and their perceived experience with each controller.

The remainder of this thesis is organized as follows. In the next chapter, I present a taxonomy of related work. Chapter 3 provides implementation details for the different controls. Chapter 4 describes the design of the human factors experiment, including choice of hardware, track design, experimental procedure, and the questionnaire. In Chapter 5, I analyze and discuss the results of the experiment. Chapter 6 provides the conclusion and a brief summary.



## Chapter 2

### Related Work

There is a large body of work about robotic control systems that led up to this point. The work in [11] references three papers in human-robot interaction: [12], [10], and [2]. The work in [12] presents a computer vision based approach to human-robot communication where the computation is done on-board the robot. The work in [10] presents work on using both gestures and speech. The speech commands are used to augment and clarify the hand gesture commands. The work in [2] presents a system for using the Wiimote to control a robotic dog, even showing that the Wiimote outperformed the standard computer keyboard.

The work in [12] takes a computer vision based approach to gesture recognition. It explores several methods like template-based matching and neural networks. It is derived from several other vision-based works, like [7], which used optical flow to recognize up to 6 gestures, and [5], which recognizes pointing gestures using feature maps and the color of the user's clothing.

The work presented in [10] integrates speech recognition into the system, but it also uses a computer vision based gesture recognizer. It draws from [6], which recognizes static hand gestures from still photos, and CORA, a robotic assistant that uses speech and deictic gestures to execute commands (ex. "Turn and face me").

The work in [2] uses a Wiimote to control a Sony AIBO robotic dog. It describes a guide to designing effective human-robot interface [1]. It also describes [13], which classifies and details robotic control schemes and defines terminology related to HRI. The work in [13] defines the autonomy level of a robot as the percentage of the time that the robot carries out a task on its own (as opposed to intervention from the operator). Like [2], this paper deals with a robot with a low autonomy level.

## Comparison to Prior Work

While this paper has most of its origin in [11], there are a few differences. The work in [11] uses a Nintendo Wiimote to control the actions of the robot, rather than a smartphone or XBOX controller. In the case of the smartphone, computation of gestures is performed on the phone, rather than on an intermediate computer. Like the work presented in [11], this system uses the movement of the controller, rather than a computer vision system to observe the user. This gives the added benefit that the user can move while controlling the robot without getting out of range.

## Chapter 3

# Implementation of Control Systems

In this chapter I will describe the implementation of the four control systems used in the experiment. First, I describe the “standard” XBOX 360 controller setup, then the application used for the D-Pad and tilt controllers, and finally the gesture-based control application.

The robot in question is an iRobot Roomba 560 vacuum cleaner. None of the four control schemes require making modifications to the Roomba, and the vacuum/brushes are turned off during the experiment. It is controlled from a standard wireless Microsoft XBOX 360 controller and a Samsung Galaxy S II Android phone.

photo  
of  
roomba  
and  
phone

### 3.1 XBOX 360 Controller

The XBOX 360 controls for the Roomba are written in Python (see Appendix). The program uses a library called RoombaSCI to communicate with the Roomba via Bluetooth. RoombaSCI is a Python wrapper for iRobot’s Serial Command Interface Specification, which allows commands to be sent to the Roomba through its serial port. RoombaSCI abstracts tasks like setting up the connection and sending bytes to specific motors into commands like FORWARD, STOP, and SPIN LEFT [4]. It also allows the programmer to set the speed of the Roomba within the range allowed by the hardware (-500 - 500 mm/s). [3]

A RooTooth bluetooth-serial adapter was used to communicate with the computer wirelessly. The Pygame library is also used to properly capture the input from the controller.

The end result is a controller that can be used to move forward/backward

and rotate to the left or right in place. The right trigger sends the **FORWARD** command, the left trigger sends **BACKWARD**, and the left analog stick pressed to the left or right sends a **LEFT** or **RIGHT** command. The robot continues moving forward/backward/left/right as long as the trigger or analog stick is held, and will stop when it is released, or another button is pressed. When using the XBOX 360 controller, the robot can only move at the maximum speed, and it can only perform one of forward/backward or rotation at any given time (forward motion must stop in order to turn). The left and right controls are not gradual, meaning that leaning the analog stick partially to the side does nothing. In order to turn left/right, the analog stick must be pushed completely to one side or the other.

xbox  
dia-  
gram

## 3.2 Cellbots Application

The D-Pad and the tilt controls were downloaded to the phone as part of a single app called Cellbots. Cellbots is an open source Android application, available for free online or in the Android Market. The app comes with four control schemes, including the D-Pad controls and the tilt controls. It also included voice controls and an on-screen Atari-style joystick which were not used in the experiments but could be utilized in future research.

screenshots  
of cell-  
bots

The D-Pad controls require the user to press one of four on-screen buttons, arranged in a cross. Like the XBOX 360 controller, the robot moves forward/backward and rotates in place when the appropriate button is held down, stopping when the button is released. Also like the XBOX 360 controller, the D-Pad only allows one speed and one type of movement at a time.

The tilt controls use the phone's accelerometers to control the robot. The user holds the phone horizontally (screen facing up) and holds down an on-screen button. While the button is held down, tilting the phone forward (rotation around its x-axis) causes the robot to begin to drive forward. The phone can be rolled right or left (rotating around its y-axis) to steer. The tilt controls differ from the other controls in two ways: first, the speed can vary. The more the phone is tilted forward, the faster the robot moves, up to the maximum speed of 500 mm/s. The second difference is that the tilt controls allow forward/backward motion and turning at the same time. For example, if the phone is tilted forward and rolled left, the robot will perform a left turn, rather than rotating in place.

phone  
axis  
dia-  
gram

### 3.3 Gesture-Based Control Application

The gesture-based controls are implemented on top of the existing Cellbots application. Cellbots is an open source application, so the gesture controls were added as a fifth control type. To recognize the gestures using accelerometers only, I turned to another open source Android application called GestureTrainer. GestureTrainer is an implementation of the concepts outlined in [9] for gesture recognition using accelerometer data.

screenshot  
of ges-  
ture-  
trainer

Users can record and name their own gestures on the phone, and from then on when the gesture is performed and recognized, the name is sent to the Roomba. The phone recognizes gestures continuously, without the need to specify when the user is about to perform a gesture.

To record a gesture, the user presses a button to start “learning mode”. The phone receives a constant stream of events for every movement that the accelerometer picks up in the form of a vector of floats,  $(\Delta x, \Delta y, \Delta z)$ . If the norm of the vector is above a certain threshold, the movement is considered “significant” and the app begins recording the stream of vectors. When the phone receives 10 events in a row whose norms do not exceed the threshold, the gesture is determined to be complete. The list of xyz-vectors is stored as a Gesture object, and given the name specified by the user.

After exiting “learning mode”, the phone begins recognizing gestures immediately. When a gesture is detected, it is stored in a Gesture object, and sampled by one of several different feature extractors. The feature extractors serve to divide the gesture into a certain number of vectors, or normalize the values in the vectors. We then compute the distance between this sampled signal and each of the user’s recorded gestures.

The distance calculation from Gesture  $a$  to Gesture  $b$  is done through the Dynamic Time Warping algorithm (see Appendix for source code). A  $a.length \times b.length$  matrix of floats,  $dist$ , is constructed, where entry  $dist(i, j)$  is the norm of the  $i$ th vector in  $a$  minus the  $j$ th vector in  $b$ . A second matrix of the same size,  $cost$ , is constructed so that the first column of  $cost$  is the same as the first column of  $dist$ , and the rest of the entries are 0.

The  $cost$  matrix is filled in dynamically from the top left ( $cost(0, 0)$ ) to the bottom right ( $cost(a.length, b.length)$ ), iterating over columns first, then rows. To compute  $cost(i, j)$ , the current minimum cost is computed as

$$minCost = cost(i - 1, j - 1) + dist(i, j)$$

This is the value assigned to  $cost(i, j)$ . If  $cost(i - 1, j) + dist(i, j) < minCost$  or  $cost(i, j - 1) + dist(i, j) < minCost$ , an additional offset penalty of 0.5 is added to  $cost(i, j)$ .

After the whole *cost* matrix has been filled in, the entry  $cost(a.length - 1, b.length - 1)$  (the bottom right corner), is the minimum distance between the two Gestures *a* and *b*. The stored Gesture with the smallest distance from the performed Gesture is the one that is selected to be sent to the Roomba. The last step is to check that distance against a threshold. If the distance is less than the threshold, then the command would be sent, otherwise no command is sent. This is to ensure that all commands sent were close matches, since every gesture will return a match with some stored gesture, even if there are no similar gestures. If the closest match has a high distance, it is not close enough to be executed as a command so it is ignored.

This code was integrated into Cellbots as another control view, to ensure that the same speeds/timing would be used for each control scheme on the phone.

The gestures defined were FORWARD, BACKWARD, LEFT, and RIGHT. All the gestures are performed with the phone's screen facing the user. The STOP command was left as an on-screen button, rather than a gesture, to make it easier for the subject to maintain control (and for the sake of the hardware and testing environment).

screenshot  
photos  
of ges-  
tures

# Chapter 4

## Human Factors Experiment Design

In this chapter I will describe the setup of the human factors experiment, as well as the design choices. The human factors experiment was conducted with 15 subjects to gauge the effectiveness of each controller. The experiment involved empirical metrics (such as time taken to complete course and number of times the robot crossed the boundaries), as well as qualitative data obtained through a questionnaire.

### 4.1 Platform and Hardware

The Roomba was chosen because it is a commercial, off the shelf robot that is relatively easy to control via a store bought Bluetooth adapter. It has two motorized wheels in the center and a single caster in the front for smooth forward steering. The lack of a back caster meant that the Roomba's backwards movement could sometimes become slightly erratic at maximum speed, but for the most part, the movement was unaffected.

The Samsung Galaxy S II was chosen primarily for its gyroscopes. All Android phones come with accelerometers, but only some with gyroscopes, which help the phone to give more precise measurements. The Microsoft XBOX 360 controller was chosen because until recent games like "Mario Kart Wii", most driving/navigating games were made for the XBOX 360, making its controller a good "standard" controller.

## 4.2 Track Design

The track used in the experiments is based on a beginner's time trial circuit from Nintendo's "Mario Kart Wii" called Mario Circuit (sometimes Mario Raceway). The course used in the study is a scaled down version of Mario Circuit, which is a slight variation on a traditional figure-eight course. The course was chosen primarily because it involves two U-turns (one left, and one right), a left turn, a right turn, and two straightaways, allowing all the commands to be tested every lap. In the game, players race around the track for 3 laps, and the 3 lap requirement was kept for the experiment. Subjects completed 3 laps, then crossed the finish line a few yards away. A "Mario Kart Wii" course was also chosen because "Mario Kart Wii" is the best-selling racing game of all time, and Mario Circuit is an updated version of one of the classic courses, originally created by the game designers as the premier time trial course [8].

pictures

## 4.3 Questionnaire Design

The questionnaire opens with basic demographic questions (age, gender, field of study/profession, etc.), then asks about subjects' video game habits and computer experience. The questionnaire then asks the following specific questions about each of the four controllers (on a five-level Likert scale unless otherwise specified):

- How aware were you of the controls while you were using them?
- How awkward was holding the controller in your hand?
- Did you feel that you were fully in control of the robot at all times? (Yes/No)
- How difficult was it to maintain control while driving in a straight line?
- How difficult was it to maintain control while turning?
- How difficult was it to learn the controls?
- Did you perform as well as you wanted to in the obstacle course? (Yes/No)
- What was your favorite part about this control scheme? (Short Answer)
- What was your least favorite part about this control scheme? (Short Answer)



Finally, the questionnaire asks participants to rank controllers according to which they would use to accomplish a critical task, which they would use continuously for two hours, which was most intuitive, and which was most fun.

## 4.4 Experiment Procedure

When subjects arrived, they were first briefed on the purpose of the experiments. Every subject tried the controllers in a randomized order. A short period was allowed for the subject to get used to the controller before beginning the time trial course. The subject completed three timed laps around the course. Subjects were allowed to move between several locations outside the track during a run of the course, but were not allowed to touch the robot or follow it closely. The number of times the robot went outside the boundaries was recorded, along with the number of “mistake acknowledgements”. I defined “mistake acknowledgements” as a physical or verbal cue that the subject performed a command they did not intend. Examples of mistake acknowledgements include stamping one’s foot, shaking one’s head, statements like “No” and “Oops”, and swearing. After every run, the subject was informed of his/her time with that controller. Immediately after all four of the time trials were finished, the subject filled out the questionnaire.

# Chapter 5

## Results

In this chapter I will present the results of the experiments and questionnaire. First, a breakdown of the empirical data from the time trials is given, followed by the data from the questionnaire. Finally, I provide my analysis of both sets of data and present some hypotheses based on the combined data.

### 5.1 Time Trials

For the time trials, I recorded the time of each lap, the total time taken to complete the course, the number of “line faults” (robot crossing out of the boundaries), and “mistake acknowledgements” (defined in Section 4.4). The averages for each of these values are presented in Table 5.1. It should be noted that there were several subjects with times more than 2 standard deviations from the mean (3 in some cases). Because of the small sample size, their data was not excluded from the calculations. The full raw data can be found in the Appendix.

Controller	Avg. Time (sec)	Avg. Line Faults	Avg. Mistake Ack.
D-Pad	161.53	0.07	0.4
Tilt	164.58	2.13	2.6
Gesture	393.81	6.67	7.0
XBOX 360	124.74	0.93	0.5

Table 5.1: Average time taken to complete the course, average number of Line Faults, and average number of Mistake Acknowledgements for each of the four controller types.

The XBOX 360 controller had the fastest average time, with 124.74s, while the D-Pad and Tilt controllers were both approximately equal (161.53s

and 164.58s, respectively). The gesture-based controls performed the slowest, with an average time of 393.81s.

Though the XBOX 360 controller had the fastest average time, the fastest individual time belonged to the tilt controller: one subject completed the course in 83.3s, making the tilt controller the only controller with a total time under 100s.

The D-Pad averaged the fewest line faults by far, with 0.07 (there was a total of 1 line fault across all subjects), while the gesture-based controls again came in last with an average of 6.67 faults.

The D-Pad also averaged the fewest mistake acknowledgements, with 0.4 average mistake acknowledgements. The XBOX 360 controller was just behind the D-Pad with an average of 0.5 mistake acknowledgements. The gesture-based controls averaged 7.0 mistake acknowledgements, more than 17 times the average number of mistakes acknowledged with the D-Pad. The gesture-based system was also the only one where the minimum number of mistake acknowledgements was above 0; every single subject acknowledged at least one mistake.

## 5.2 Questionnaire

Presented here are answers to several relevant questions. More data can be found in the Appendix.

The D-Pad received mostly positive reviews, though 60% of respondents said that they were aware of the controls while using them. 80% said the controller was not awkward, and that they felt that they were fully in control of the robot at all times. 100% of respondents rated the controls as easy to learn. 80% said that they performed as well as they wanted to in the course. Many subjects enjoyed the “simplicity” and “familiarity” of the D-Pad, along with the fact that the directional arrows were visible. The most popular complaint about the D-Pad was that it could only handle one command at a time and could not turn and move simultaneously. It was also described as “mundane” and “less fun than [some other controls]”.

In a slight jump up from the D-Pad, 66% of subjects said they were aware of the tilt controls while using them. 53% rated the controls awkward, and only 60% said they felt in control of the robot at all times. 40% of respondents said they performed as well as they wanted to (a large deviation from the D-Pad). Subjects praised the tilt controls for being “intuitive”, “smooth”, and “fun”, while others criticized it for being “hard to control”.

The gesture-based controls did not fare as well. 80% of users said that they were aware of the controls while using them, and 66% described them

as awkward. 86.7% of subjects said they did not feel in control of the robot at all times, and 86.6% said it was difficult to maintain control while turning (46% rated it hard to control while driving in a straight line). The majority of subjects rated the gestures somewhat easy to learn. Subjects enjoyed the “novelty” of the gesture-based controls, but ultimately decided it was “unresponsive”, “inaccurate”, and “frustrating” at times.

The XBOX 360 controller got slightly better ratings than the gesture-based controls. Most respondents said that they were somewhat aware of the controls, while 86.6% rated it not awkward. 86.7% said they felt that they were fully in control of the robot at all times, which is the most of all the controllers. 100% rated the XBOX 360 controller easy to learn. 86.7% performed as well as that wanted to in the course. Subjects praised it for being “familiar”, “easy to learn”, and for having a physicality that allowed them to keep their eyes on the robot at all times. There were, however, many complaints about not being able to turn while moving forward, like the tilt controls, and having no way to gradually increase the amount the robot was turning.

The majority of subjects, 53.3% said that they would use the XBOX 360 controller as their first choice for accomplishing a critical task. 80% of subjects ranked the gesture-based controls last for accomplishing a critical task, with no users ranking it as their first choice (see Figure 5.1).

Similarly, 71% of subjects would use the XBOX 360 controller for two hours continuously, while 80% ranked the gesture-based controls in last place (see Figure 5.2).

The tilt controller was ranked the most intuitive (40%), followed by the D-Pad, then the XBOX 360 controller. No subjects rated the gesture-based controls the most intuitive (see Figure 5.3).

The tilt controller was also ranked the most fun to use, followed by the XBOX 360 controller, with D-Pad and gesture-based tying for third (see Figure 5.4).

### 5.3 Analysis and Hypotheses

Empirical data shows that the XBOX 360 controller was the fastest, and the second most “error free”, meaning a low number of line faults and mistake acknowledgement. The gesture-based controls performed poorly, with an average time, average number of faults, and average mistake acknowledgement more than twice all the others’.

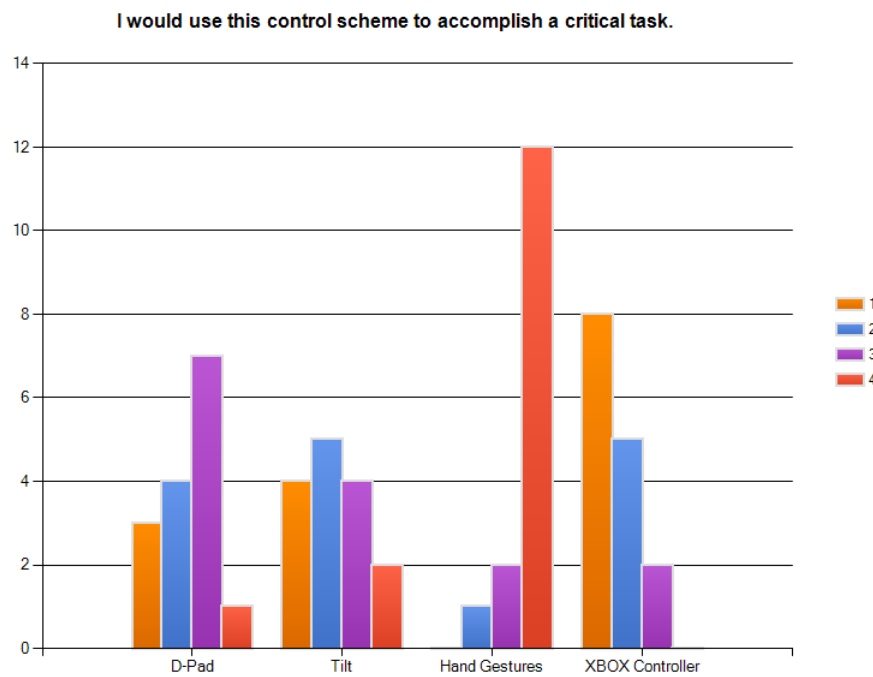


Figure 5.1: Numbers indicate rank given to the controller for using it to accomplish a critical task. The XBOX 360 controller was the most popular for accomplishing a critical task; the gesture-based controls were the least popular.

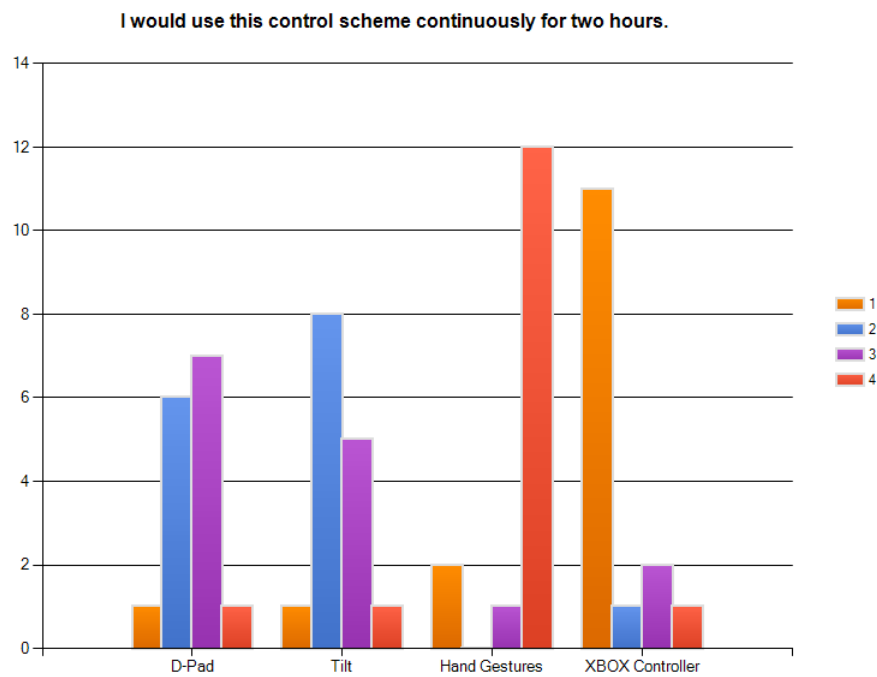


Figure 5.2: Numbers indicate rank given to the controller for using it for two hours continuously. The XBOX 360 controller was the most popular for using continuously for two hours; the gesture-based controls were the least popular.

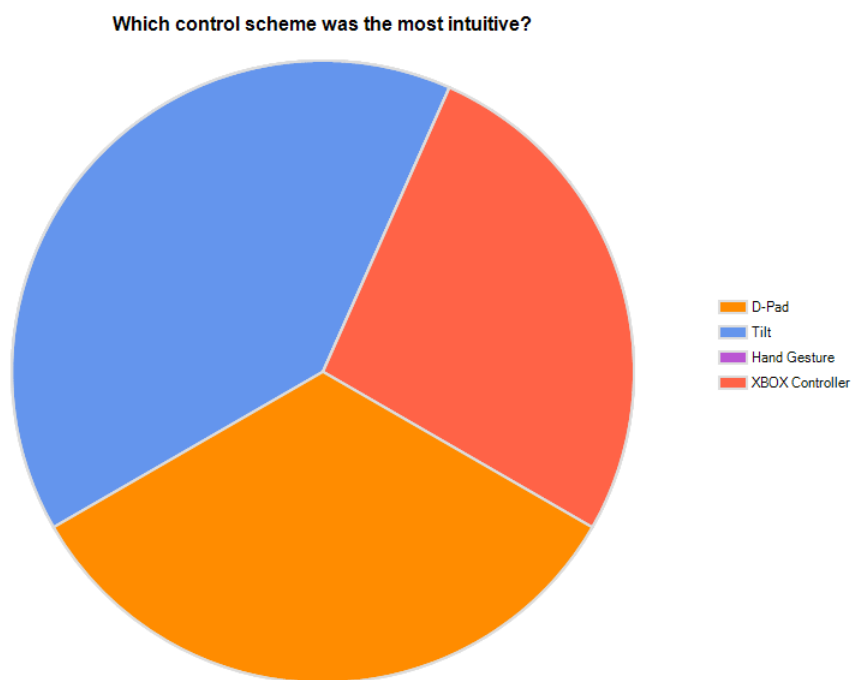


Figure 5.3: The tilt controller was ranked the most intuitive, while the gesture-based controller was ranked the least intuitive.

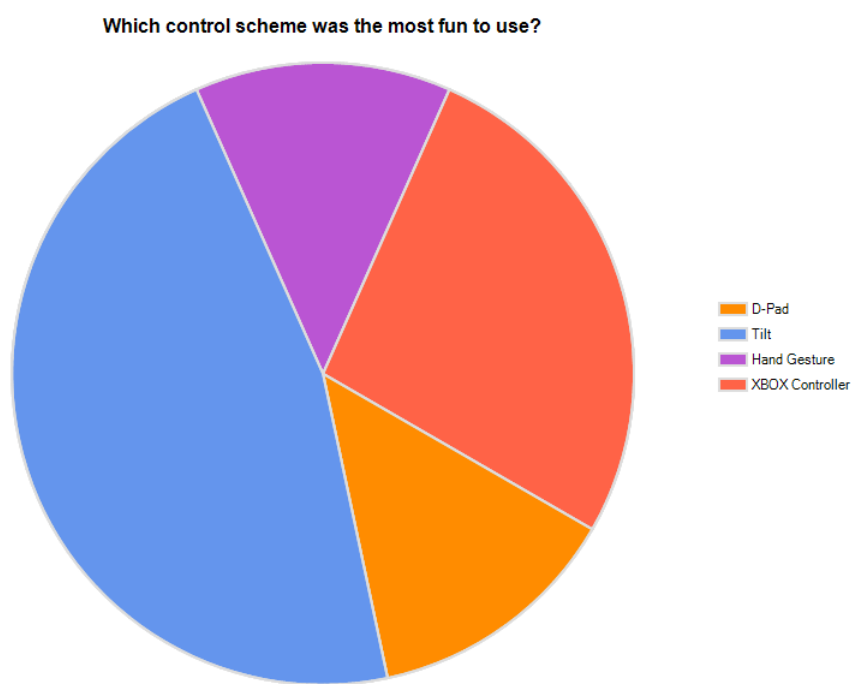


Figure 5.4: The tilt controller was ranked the most fun to use, while the D-Pad and gesture-based controls tied for 3rd place.



# Chapter 6

## Conclusion

This is a conclusion.

# Bibliography

- [1] M.A. Goodrich and Jr. Olsen, D.R. Seven principles of efficient human robot interaction. In *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, volume 4, pages 3942 – 3948 vol.4, oct. 2003.
- [2] Cheng Guo and Ehud Sharlin. Exploring the use of tangible user interfaces for human-robot interaction: a comparative study. In *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, pages 121–130, New York, NY, USA, 2008. ACM.
- [3] iRobot Corporation. irobot roomba serial command interface (sci) specification, 2005.
- [4] K. Jacobson. Python library for the roomba sci, 2008.
- [5] R.E. Kahn, M.J. Swain, P.N. Prokopowicz, and R.J. Firby. Gesture recognition using the perseus architecture. In *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR '96, 1996 IEEE Computer Society Conference on*, pages 734 –741, jun 1996.
- [6] H.A. Kestler, M. Borst, and H. Neumann. *Einfache Handgestikerken-nung mit einem zweistufigen Nearest-Neighbour-Klassifikator; SMART Ulm*. Report. Univ., SFB 527, 1999.
- [7] David Kortenkamp, Eric Huber, R. Peter Bonasso, and Metrica Inc. Recognizing and interpreting gestures on a mobile robot. In *In Proceedings of AAAI-96*, pages 915–921. AAAI Press/The MIT Press, 1996.
- [8] Nintendo Co. Ltd. Mario kart 64 instruction booklet, 1997.
- [9] Robert Nesselrath. Takg - ein toolkit zur automatischen klassifikation von gesten. Master's thesis, University of Saarland, 2008.

- [10] Rogalla, Ehrenmann, Zollner, Becher, and Dillman. Using gesture and speech control for commanding a robot assistant. 2002.
- [11] Paul Varcholik, Daniel Barber, and Denise Nicholson. *Interactions and Training with Unmanned Systems and the Nintendo Wiimote*, volume 2008, pages 1–9. NTSA.
- [12] Stefan Waldherr, Roseli Romero, and Sebastian Thrun. A gesture based interface for human-robot interaction. *Autonomous Robots*, 9:151–173, 2000. 10.1023/A:1008918401478.
- [13] H.A. Yanco and J. Drury. Classifying human-robot interaction: an updated taxonomy. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, volume 3, pages 2841 – 2846 vol.3, oct. 2004.

# Appendix

## Gesture Recognition Code

This is the portion of the code that calculates the distance between two Gestures, *a* and *b*.

```
/*
 * DTWAlgorithm.java
 *
 * Created: 18.08.2011
 *
 * Copyright (C) 2011 Robert Nesselrath
 *
 * This program is free software; you can redistribute
 * it and/or
 * modify it under the terms of the GNU General Public
 * License
 * as published by the Free Software Foundation; either
 * version 2
 * of the License, or (at your option) any later
 * version.
 *
 * This program is distributed in the hope that it will
 * be useful,
 * but WITHOUT ANY WARRANTY; without even the implied
 * warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License
```

```

    * along with this program; if not, write to the Free
      Software
    * Foundation, Inc., 59 Temple Place – Suite 330,
      Boston, MA 02111–1307, USA.
    */
package de.dfki.ccaal.gestures.classifier;

import java.util.ArrayList;

import de.dfki.ccaal.gestures.Gesture;

public class DTWAlgorithm {

    static float OFFSET_PENALTY = .5f;

    static private float pnorm(ArrayList<Double> vector,
        int p) {
        float result = 0, sum;
        for (double b : vector) {
            sum = 1;
            for (int i = 0; i < p; ++i) {
                sum *= b;
            }
            result += sum;
        }
        return (float) Math.pow(result, 1.0 / p);
    }

    static public float calcDistance(Gesture a, Gesture b
    ) {
        int signalDimensions = a.getValues().get(0).length;
        int signal1Length = a.length();
        int signal2Length = b.length();

        // initialize matrices
        float distMatrix [][];
        float costMatrix [][];

        distMatrix = new float [signal1Length] [];
        costMatrix = new float [signal1Length] [];
    }
}

```

```

for (int i = 0; i < signal1Length; ++i) {
    distMatrix[i] = new float[signal2Length];
    costMatrix[i] = new float[signal2Length];
}

ArrayList<Double> vec;

// calculate distances
for (int i = 0; i < signal1Length; ++i) {
    for (int j = 0; j < signal2Length; ++j) {
        vec = new ArrayList<Double>();
        for (int k = 0; k < signalDimensions; ++k) {
            vec.add((double) (a.getValue(i, k) - b.
                getValue(j, k)));
        }
        distMatrix[i][j] = pnorm(vec, 2);
    }
}

// genetischer Algorithmus um den  $g\tilde{A}_4^{\frac{1}{4}}$ nstigsten Pfad
// zu finden
for (int i = 0; i < signal1Length; ++i) {
    costMatrix[i][0] = distMatrix[i][0];
}

for (int j = 1; j < signal2Length; ++j) {
    for (int i = 0; i < signal1Length; ++i) {
        if (i == 0) {
            costMatrix[i][j] = costMatrix[i][j - 1] +
                distMatrix[i][j];
        } else {
            float minCost, cost;
            // i-1, j-1
            minCost = costMatrix[i - 1][j - 1] +
                distMatrix[i][j];
            // i-1, j
            if ((cost = costMatrix[i - 1][j] + distMatrix
                [i][j]) < minCost) {
                minCost = cost + OFFSET.PENALTY;
            }
        }
    }
}

```

```

        // i, j-1
        if ((cost = costMatrix[i][j - 1] + distMatrix
            [i][j]) < minCost) {
            minCost = cost + OFFSET_PENALTY;
        }
        costMatrix[i][j] = minCost;
    }
}
return costMatrix[signal1Length - 1][signal2Length
    - 1];
}
}

```

**Time Trials Raw Data**

**Questionnaire Data**



# Evaluation of Gesture-Based Robotic Control Systems



## 1. What is your subject number?

	Response Average	Response Total	Response Count
Subject Number	46.73	701	15
answered question			15
skipped question			0

## 2. Are you male or female?

		Response Percent	Response Count
Male		60.0%	9
Female		40.0%	6
answered question			15
skipped question			0