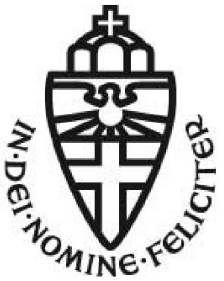


Radboud Universiteit Nijmegen

Bachelor Thesis

The effect of viewpoint and mode of control on performance and user experience while controlling a Sphero

Author:
Dennis Luiten

Supervisor:
Dr. Louis Vuurpijl

September 1, 2014

Contents

Chapter 1

Introduction

1.1 Intro

Teleoperation is defined by pioneer in robotics and remote control technology, Thomas B. Sheridan [1] as:

> "Teleoperation is the extension of a person's sensing and manipulation capability to a remote location."

It is used in a wide variety of situations and environments including: Unmanned Aerial Vehicles (UAV), Unmanned Ground Vehicles (UGV), nuclear reactors, Mars rovers and many more [2, 3, 4]. Much research is aimed at improving the robot's communication software/hardware, sensors, effectiveness of control and autonomy. However, considerably less research addresses the ease of use or user experience [5] of these systems. The focus points of this thesis will include what methods of teleoperating robots work best, is the easiest to learn, least exhausting or most fun?

1.2 Gestures in HCI

While interacting with computers or robots by means of gestures seems intuitive and more appealing, this is not always the case. One should test for, or at least think about, the benefits and drawbacks of abandoning traditional interaction methods and embracing novel ways of human computer interaction. [6] Furthermore, it is important to implement a good set of possible gestures the user can use. Not all gesture sets are neccesarily good

sets. Some may only work well in very specific settings. Others may be very effective, but difficult to remember or tiresome. All in all, designing a gesture interface requires thought and some knowledge of HCI. The usability of traditional interfaces is no less important, but one can rely on much more previous research that has been done in this area. [7, 8, 9]

The idea of gesture based interfaces has been around for quite some time [10]. A lot of research has been done regarding hand gestures and facial expressions [11], but this field of HCI has really developed with the introduction of the Kinect in 2010. Although this device was initially intended for gaming purposes, it is has quickly been adopted by researchers in robotics as a rather cheap and adequate option for computer vision.

While a lot of research regarding the Kinect has been done about its accuracy and speed of recognizing gestures, there are also a number of studies which investigate the usability of the device from the viewpoint of the user. A few examples:

- Rouanet et al.[12] compared three different interfaces to control the Aibo robot dog along an easy and a harder obstacle course. Their goal was to measure the most efficient interface and determine which one the users preferred. They determined the efficiency of an interface by the objective measures: total time until completion, time spent in motion and number of actions. The preferred interface was measured by a questionnaire at the end of the experiment.

  The first interface they used was a touchscreen gestures based one. The participant received information about the robot through a video stream of the onboard camera of the Aibo or by looking at the robot directly. They used a trajectory metaphor which lets the user draw a trajectory on the touchscreen of a smartphone, the robot then attempts to follow this path. A drawback of this approach is that you can only draw trajectories that are within the visual field of the onboard camera of the robot. However, you can tap the sides of the touchscreen to make the Aibo turn left or right to shift the visual field.

  Secondly, a tangible user interface (TUI) being a Nintendo Wiimote. With this Wiimote, the user performed hand gestures to control the Aibo. With this interface it is only possible to view the robot directly, there was no way to view the feed from the camera of the Aibo. Tilting the Wiimote forward or backward will cause the Aibo to move in the

corresponding direction. Twisting the device left or right will make the robot turn in this direction. Turning just the head instead of the entire robot could be done by pressing a button on the Wiimote while twisting.

Lastly, they used a virtual keyboard interface on a handheld device. Same as in the first interface, the robot could be watched directly and the video stream from the onboard camera could be seen on the screen of the smartphone. Four semitransparent arrow keys where displayed on the touchscreen of the device. These were used to move and turn the robot. It was possible to move just the head by moving either of the two slider bars on the edge of the screen. A problem with this interface was that the fingers often occluded the touchscreen, rendering the video stream less accessible.

They found no difference in efficiency between the interfaces on the easy course. However, on the hard course they found a slight advantage when using the Wiimote. Nonetheless, the Wiimote was clearly the least preferred interface of the users. While the touchscreen gestures based interface was distinctly the most preferred interface.

- Another study, by Sanna et al.[13], explored ways to fly a Parrot AR.Drone using various control methods. They had compared the use of multi touch gestures on an iPhone, control by means of a joystick and performing gesture and posture commands via the Kinect. The Kinect appeared to be the slowest, yet the most accurate way of controlling the Parrot AR.Drone.

- Farhadi-Niaki et al. [14] studied the use of three interaction methods on a 3D computer game. The methods they considered were: static gestures (without movement), dynamic gestures (with movement) and a haptic 3D mouse. They investigated objective measures precision (errors) and efficiency (time) as well as subjective measures such as ease-of-use, fun-to-use, naturalness, fatigue and mobility. It was found that the static gestures scored the best on all accounts.

1.3 Teleoperation versus Actual Presence

Actual Presence

As opposed to controlling a robot from a distance, one can command it while standing next to it. In this thesis we will refer to this as Actual Presence. Clearly, being in the same room as the robot one is controlling has it's benefits. For example, the user is not bound by fixed camera points in the surroundings or the robot itself. He is able to move around freely to look at the robot and the environment any way he wants to. (Although in our experiment, the user does not have complete freedom, his movements are somewhat limited. Nevertheless, the movement range is probably more than the users will want to use.)

Another benefit of Actual Presence is that the user can directly see and hear the robot's interaction with its environment. As opposed to watching a robot a (large) distance away on a computer screen. And of course, when directly interacting with a robot, there is no delay in information. This may be the case when teleoperating over large distances and/or with large amount of data.

Viewpoint

Depending on the point from which the user views a robot, his skill and experience can vary greatly. It is therefore important to think about where the user or a camera should be positioned with respect to the robot. Research has already been conducted in this area by Huber et al.[15] In one condition, they let the participant drive the vehicle from the viewpoint from the onboard camera. In the other condition, they generated a 3D model from the input of the camera, the participant could then choose from which viewpoint they would see the vehicle while performing the driving task. The tests showed that the 3D model condition resulted in faster as well as better (i.e. less collisions) performance. Also, from the many possible viewpoints, Huber et al. found that most participants preferred the so called "over the shoulder" viewpoint. From here, the user sees the entire vehicle from behind and slightly above. Although in our experiment the user is not always behind the robot, he is slightly above it. Thus, the user can view the robot itself, where it is currently going and an overview of the obstacles which the robot has to pass.

1.4 Our approach

For our research, we created a setting of navigating a robot through an obstacle course. Out of a wide range of possibilities we chose to control the robotic ball Sphero from Orbotix. We decided on this particular robot for a number of reasons. Firstly, it is very agile and thus ideal to maneuver through an obstacle course. Secondly, it is a novel robot and it moves distinctly different (yet still intuitive) from mainstream remote controlled robots with four wheels. Thus, we can assume that few of our participants have used this kind of robot before, thereby reducing the noise in the data from previous experience. Nevertheless, we checked for it by means of our questionnaire after the experiment. The Sphero is a 7.5 cm in diameter ball which can be used both as the controller and as the controlled robot. The Sphero has only been available since 2012. As such, there has not been done much research that made use of this novel device. Nonetheless, there are a few examples which use it as input device [16], as a remote controlled robot [17] or as a robot to interact with. [18]

The objective of our research was to investigate two variables: mode of control and viewpoint. For the first variable, we compared two different ways of controlling the Sphero. In our first condition, the user would move the Sphero around using hand/posture gestures. To record these gestures we used the Kinect, as this is a cheap and reliable hardware choice. For the second condition we used a different device as input: a smartphone. Orbotix has developed an application which can control the Sphero using the accelerometers of a smartphone. Simply tilt the device in a direction and the Sphero will move in the corresponding direction.

For the second variable, we varied the viewpoint from which the participant was looking at the Sphero. The first condition we named actual presence. The participant is standing in front of the robot. Looking directly at it and its surroundings. In the second condition, the user and the Sphero are separated. The robot is viewed by a camera and the participant can see this feed on a computer screen. This condition is called teleoperation.

The two dependent variables in our research were objective performance and subjective user experience. Our performance metrics were: total time until task completion (tttc) and number of calibrations needed. We measured the time taken for each lap in each condition. Tttc for each condition was calculated by adding all four laps in a condition together. Number of calibrations means the number of times the participant and the Sphero have gotten

out of sync and required recalibration. The calibration process is explained in more detail in section 2.1.

User experience (UX) is defined by Hassenzahl [5] as:

> "UX is a momentary, primarily evaluative feeling (good-bad) while interacting with a product or service."

According to Hassenzahl, UX is the result of two factors of the product: pragmatic (or ergonomic) quality and hedonic quality. Pragmatic quality is the degree to which the interface supports the user in achieving its current tasks. This is closely related to the traditional notion of usability in HCI. Hedonic quality on the other hand may have no direct relation to the current task. They are more abstract concepts which a user may desire from a product such as novelty or relatedness. Hassenzahl has conducted experiments which with questionnaires to assess these aspects of UX [19]. We decided not to use the same questionnaires as Hassenzahl used in his experiment, as these were rather lengthy. Instead we created a shorter questionnaire of our own. Thus, these questions are not guaranteed to the amount to the same UX as Hassenzahl, but we think it will resemble it nonetheless. For the rest of this thesis, UX will mean the value of UX as measured by our own questionnaire.

Now that we have discussed all of our dependent and independent variables, we can formulate our research questions:

1. (a) How does the mode of control affect the performance while controlling a Sphero?
   (b) How does the viewpoint affect the performance while controlling a Sphero?
2. (a) How does the mode of control affect the user experience while controlling a Sphero?
   (b) How does the viewpoint affect the user experience while controlling a Sphero?

As stated earlier, Sanna et al. [13] found that their participants preferred the use of gestures but they performed worse than when using traditional input methods. Therefore, we hypothesize that our participants will prefer the use of the Kinect over the use of the tilt application. We also predict that the participants will perform better when controlling the Sphero with the tilt application. Our third hypothesis is that our participants will prefer and perform better in the actual presence condition over the teleoperation condition.

Chapter 2

Gesture design and recognition

Before we decided on the gestures we used in the experiment, we considered a number of alternatives. In this section, firstly we will discuss the difference in controlling the Sphero and other remote controlled vehicles. Then we will elaborate on how this affected the gestures we decided on. Then we will describe a few alternative gestures we considered and compare their advantages and drawbacks to the final gestures. Finally, we will provide an overview of how the gestures are recognized by our system. This process will be described in detail later on in section 4.1.

2.1 Sphero versus remote controlled vehicles

One important aspect of designing gestures for controlling the Sphero was that unlike many remote controlled (RC) vehicles, the Sphero is not bound by a direction it is facing. To illustrate, while controlling a RC car, the objective is to move to the north when the initial position of the vehicle is facing west. To complete this task, one would then issue a command to make the vehicle turn left, so that it is facing north and then drive forward.

However, controlling a Sphero is quite different. The Sphero does not have a direction it is facing. Instead, you have to calibrate it such that its forward is aligned with the user. The user can then give commands to the Sphero from his own point of view, instead of having to mentally place himself in the robot to determine which direction to go. Thus, instead of making the robot turn left and move forward, you can simply tell it to move north from your perspective. The Sphero's internal mechanisms to keep track of its

calibration point are very robust. However, it may be required to recalibrate when bumping hard into other objects, falling or slipping.

In other words, due to the nature of the Sphero, the gestures to design are independent from the direction which the robot is facing.

2.2 Gesture design

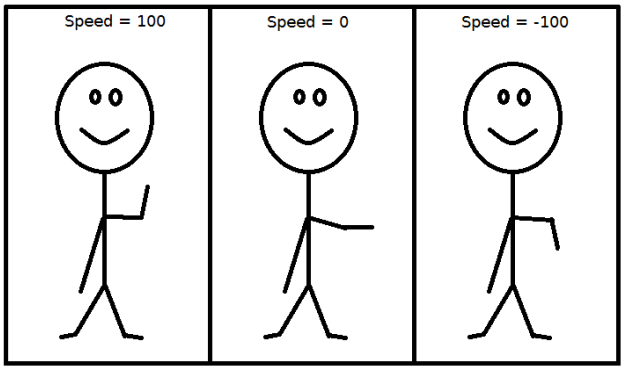
Thus, there were two important principles we had to keep in mind when designing our gesture repertoire. First, the direction of the Sphero was not dependent on a direction it is facing. Second, the gestures had to simultaneously control the direction and the velocity of the Sphero. With this in mind, we considered two main types of gestures. Either we can create a 'point to go here' type of design or we can make gestures for each direction de Sphero can move in. The 'point to go here' design means that the user can simply point to a place in space, an algorithm will calculate where the finger is pointing and send the Sphero in the corresponding direction. Although this approach seemed more intuitive, this gesture set comes with two main drawbacks. Firstly, it would take a lot of work to make the Sphero know where it was and determine exactly where the user is pointing. But the main problem was that it is impossible to execute the teleoperation condition while pointing. The user would then be supposed to point to the computer screen, which would not work very well and it would not be very intuitive. So, this leaves us with gestures for each direction the Sphero can move in.

Final gestures and recognition

The final version of the gestures uses the right arm for the direction, being left, front, right and six directions on equal distance between them. The left arm is used mainly for velocity. However, when the left arm signals the velocity to be negative. This affects the command from the right arm. For example, a command to the left-front would become left-rear.

In more detail, the velocity is determined by the vertical positions (y-position) of the left hand and elbow. If they are at the same height, the velocity will be zero. When the hand's y-position is a forearm's length above the elbow, the velocity will be the maximum which is 100. (The length of the forearm of the current user is measured upon calibration.) On the other hand, when the hand is a forearm's length below the elbow, the velocity is maximal

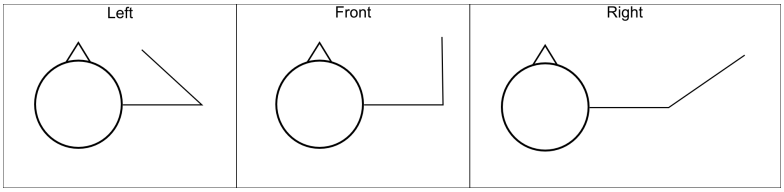
Figure 2.1: An illustration showing the left hand command for the speed of the Sphero.



in reverse, that is -100. Anywhere in between results in a value between -100 and 100. How this value is calculated from the input is described in section 4.1.

The direction is thus calculated by the positions of the joint in the right arm and the velocity. First, the Kinect measures the x-positions of the right hand and elbow. The possible directions to the front are divided in nine discrete options. Then it is computed in which section the hand is positioned in regard to the elbow. Finally, if the current velocity is negative, thus in reverse, the direction will be altered. For example, a command to the front-front-right reversed becomes a rear-rear-right command.

Figure 2.2: An illustration showing the right hand command for the direction of the Sphero.



Alternatives

We also considered a gesture set which used the left arm to determine the direction to the left and the right arm when going to the right. The other

arm would then determine the velocity. This was deemed unintuitive mainly because the switching between arms did not make sense.

Also, instead of commanding the velocity along the y-axis with the left hand, we first considered this to be along the z-axis. That is the user would have to extend his and for higher velocity. However, the drawbacks of this approach were that it was less intuitive and comfortable for the user, especially when going in reverse. Also, the z-axis had lower performance. Possibly because the hand of the user would obstruct his elbow and shoulder which are other joints which the system uses.

Another possibility was to include the shoulder joint in the measurements in addition to the elbow and the hand. This seemed slightly more intuitive. However it was possible for the user to clash his arms when steering left and increasing the velocity at the same time. Another issue with this approach was that it is more tiresome as the user has to make larger movements and has to keep more of his arm flexed. While in our final gesture set the right upper arm is used very minimally.

Chapter 3

Methods

3.1 Design

In this experiment, participants were tasked with completing an obstacle course with a Sphero. The main research questions were:

1. (a) How does the mode of control affect the performance while controlling a Sphero?
   (b) How does the viewpoint affect the performance while controlling a Sphero?
2. (a) How does the mode of control affect the user experience while controlling a Sphero?
   (b) How does the viewpoint affect the user experience while controlling a Sphero?

We used a 2x2 within subject design, resulting in 4 conditions for each participant. The independent variables were: firstly, mode of control. In one condition, the participant used gestures to control the Sphero, acquired via a Microsoft Kinect. In the second mode of control, the Sphero was commanded by using a tilt application on an iPhone. The second independent variable concerned the presence of the participant. The user can either be standing directly over the obstacle course, the actual presence condition. Or the participant is watching the course on a television screen, the remote presence or teleoperation condition. The order in which the participants performed each condition was randomized.

We wanted to measure the objective performance and the subjective user experience (UX) for each of the conditions. Our performance metrics included: total time taken and number of calibrations needed. To measure UX, we used a questionnaire on paper loosely based on the questionnaire used by Hassenzahl. A number of control questions were added to the questionnaire to assess a few other factors and to determine any possible flaws in our experiment. The entire questionnaire can be found in Appendix A.

3.2 Experiment

This section will specify the course of the experiment step by step. The actual instructions given by the experimenter can be found in Appendix B.

First, the participant was informed about the procedure of the experiment. Then the participant was explained how to control the Sphero using the tilt function of the handheld device. This was followed by an opportunity to practice with this mode of control for three minutes. After this practice session, the participant was informed about the gestures used to control the Sphero. Again, followed by an opportunity to practice with this mode of control for three minutes. Both of these practice sessions were done from the actual presence viewpoint. The participants were then instructed on the viewpoint teleoperation. Followed by a three-minute practice session with teleoperation as viewpoint and the tilt application as mode of control. After that, there was a final three-minute practice session for the teleoperation with Kinect control condition.

When the participant is familiarized with both ways of controlling the Sphero, the four conditions (these are: tilt control with actual presence, tilt control with teleoperation, gesture control with Actual Presence and gesture control with teleoperation) of the experiment were executed in a randomized order.

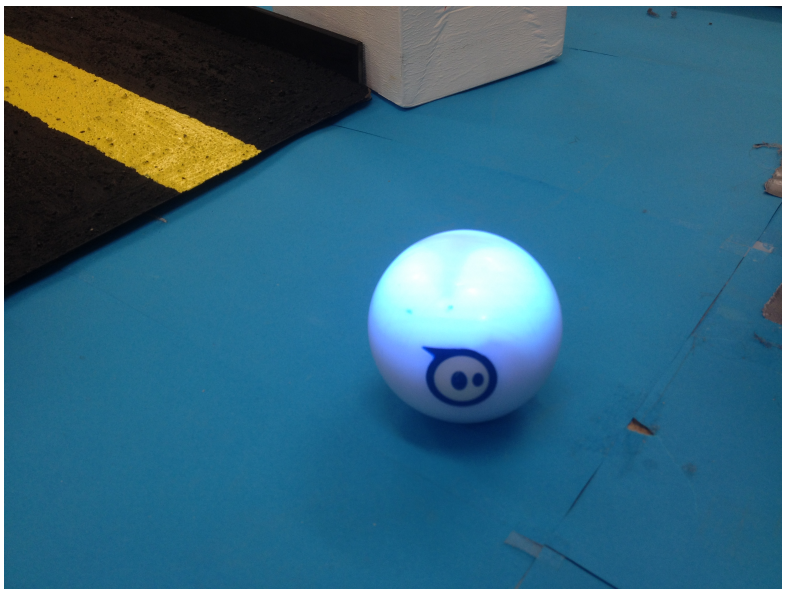
After the actual experiment had been conducted, the participant had to fill in a questionnaire on paper. The questionnaire consisted of some personal information, 34 questions about the experiment on a 5-point Likert scale and 1 open question. The full questionnaire can be found in Appendix A.

3.3 Participants

For this experiment, we tested 8 participants. All of them were affiliated with the Radboud University Nijmegen department of Artificial Intelligence and of Dutch nationality. Four bachelor students, two master students, one PhD student and one assistant professor. Five of them were males, the other three females. They aged from 22 to 31 (mean = 24.38, SD = 3.445). They were given cookies in exchange for their participation.

3.4 Hardware

Figure 3.1: The Sphero by Orbotix.



Here, we will provide the specifications of the numerous pieces of hardware that were used in this experiment.

For recognizing gestures, we used the Microsoft Kinect for Xbox connected to a HP Envy laptop with a 2.4 GHz Intel Core i7 processor and 8 GB RAM memory. For tilt control by application, we used a standard issue iPhone 4s.

To view the obstacle course from above for the teleoperation condition, we streamed the feed of a GoPro HERO3+ to a 60 inch LG tv. We mounted

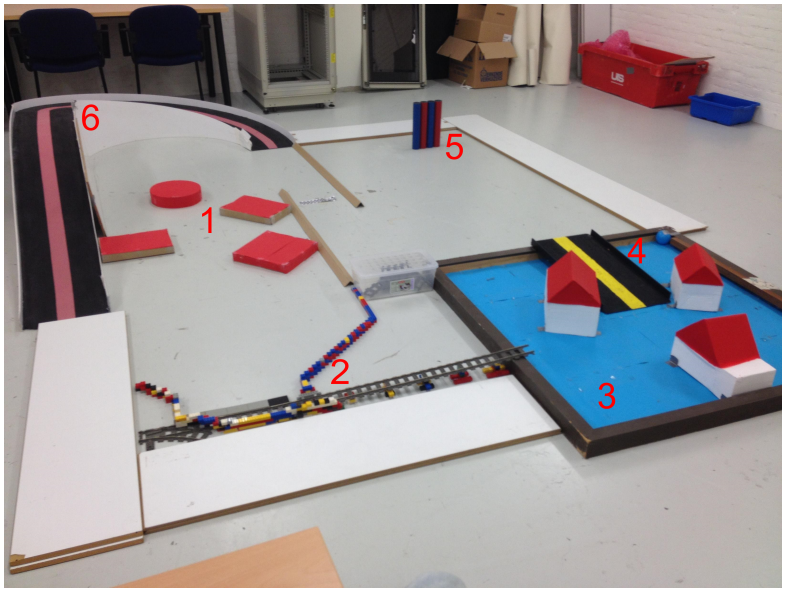
the GoPro onto a pipe on the ceiling. To get the entire obstacle course on camera we used the wide angle lens of the GoPro.

The robot we decided to control was the robotic ball named Sphero from Orbotix. A 7.5 cm sphere which can be used both as the controller and as the controlled robot. The Sphero can go up to 11 kilometers per hour. It has a 3-axis gyroscope as well as a 3-axis accelerometer. It communicates with other devices by means of Bluetooth 2.0. The Sphero is designed to be very robust such that hard collisions will not cause any damage.

3.5 Obstacle Course

This section specifies the components of the obstacle course in detail, as seen below in figure 3.2

Figure 3.2: An overview of the obstacle course



Maze (1): The obstacle course contained a small maze consisting of four cardboard squares and circles. They were arranged close to each other such that the user has to make small precise movements to cross the maze. Their dimensions are: the square is 28 cm wide and long. The circle has a diameter of 26 cm. The two rectangles are 20 cm wide and 31 cm long. These values

are somewhat arbitrary as the shapes are not custom made, but created with available material.

Railway track (2): We used a 85 cm long railway track built out of Lego rails to move the Sphero over ledge of the container. The length is determined by the height of the ledge. A shorter railway track is steeper and may cause the Sphero to fail to move over the track or cause the track to break or fall over due to the weight of the Sphero.

Container (3): The container has a ledge to prevent the Sphero from exiting at other points then the see-saw. There are also three house shaped obstacles present in the container. The container is entered trough the railway track and exited via the see-saw.

See-saw (4): We have used a see-saw with a rough surface for increased traction. Its dimensions are 30 cm wide, 56 cm long and 3 cm high from the floor to its turning point. Also, it has a 3.5 cm high ridge to prevent the Sphero from falling off from the side of the see-saw.

Four cylinders (5): The user was tasked to knock over these cylinders. They are 30 cm high and has a diameter of 5 cm. After every lap, the cylinders were placed back in their original positions, so that each lap was the same.

Bridge (6): On the left there is a bridge. However, due to the difficulty of this obstacle, it was not used in the experiment.

Chapter 4

Software

In this chapter, we will discuss all of the software components used in our experiment. The first section will cover supportive software such as the used iPhone application, imported libraries and the camera viewing program. The following section, processing pipeline, will elaborate on each step in the process of turning the input data from the Kinect and the Sphero into commands for controlling the Sphero. Last section lists the components of our graphical user interface.

All of the code written by us for this experiment was written in Java. Although, both the Sphero and the Kinect have their own SDK's, linking one to another would have been difficult and very time consuming. As Java has known interfaces to both devices, we deemed this to be an adequate solution. The code of a number of relevant classes can be found in Appendix C.

4.1 Supportive Software

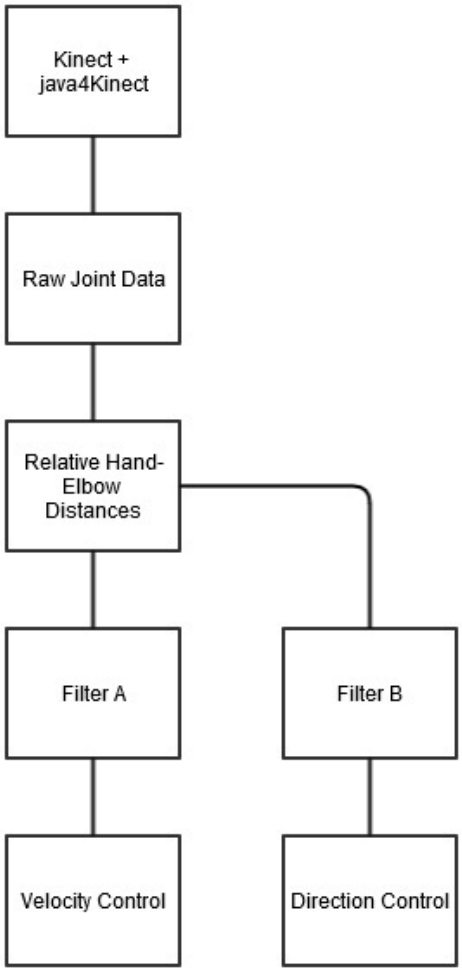
First off, a small summary of the supportive software which we did not write ourselves.

The application used to control the Sphero by tilting a smartphone was the app "Sphero Drive" 2.1. Free of cost downloadable in the Apple App Store and Google Play Store.

The HP laptop ran on Windows 8.1. The code for the experiment was fully written in Java. To interface with the Kinect, the public Java library Java4Kinect (J4K) was used. The communication with the Sphero was regulated with the public Java library Sphero Desktop API by Niklas Gavelin [20] .

4.2 Processing pipeline

Figure 4.1: The processing pipeline



In the following subsections, we will describe the various parts of the processing pipeline.

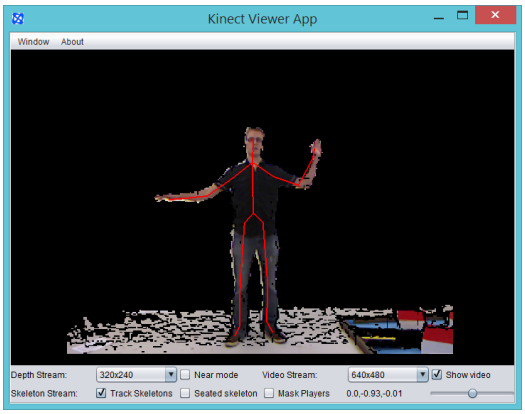
4.2.1 Kinect and Java4Kinect

As state earlier we used a Kinect to observe our participants and the gestures they were making. To read the data from the Kinect, Microsoft has provided an IDE for developers. However, we need to communicate the commands made by the user to the Sphero and this is not possible through this IDE. Instead, we used Java4Kinect, a Java library which reads the data from the Kinect and makes it accessible in Java. For more information as to how J4K works, see [21].

4.2.2 Joint positions

From Java4Kinect we get raw joint positions. These are three dimensional points for the following joints (if plural, both left and right are separate points): head, neck, shoulders, elbows, hands, torso, hips, knees and feet. The origin of the axes is in the center of the depth image.

Figure 4.2: Depth map by the Kinect with skeleton tracking.



In our experiment, only the elbows and hands were used to control the Sphero. The rest of the joints were only used to provide the user with feedback as depicted in figure 4.2. For velocity control, we calculated the vertical distance between the left elbow and hand. On the other hand for direction control, we calculated the horizontal distance between right hand and elbow.

4.2.3 Filtering

Filters are often used to reduce the amount of noise in sensory data. Although the level of noise in the relative hand-elbow distances was somewhat low, we considered a couple of filters to reduce the noise even further.

Median Filter

A median filter considers a number of previous input values. In the filter then produces as output the median of these input values.

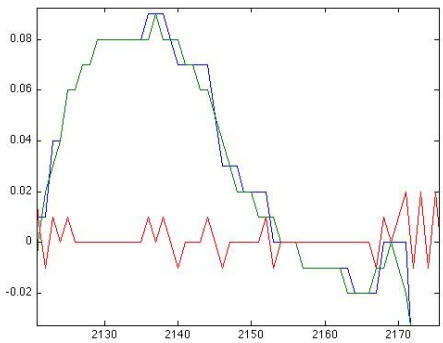
In our case, we determined the number of previous input values to be five. Less than five will reduce the effect of the filter. More than five will cause a larger delay for a different pose to pass through the filter, while not contributing any further to the noise reduction. It is important to keep the delay low as we are controlling a robot. Long delays will cause the robot to overshoot quite often.

Average Filter

An average filter also takes a number of previous input values. However, instead of calculating the median, it takes the average of them all.

Just like with the median filter, we determined the number of previous input values to be five. The arguments for this number are the same as for the median filter.

Figure 4.3: Average filter versus median filter (Blue: average, green: median, red: difference)



Comparison

We compared the two filters to see with would suit our needs better. We chose for the average filter because it is more smooth. When the user changes his desired velocity, the median filter tends to jump between values. While the average filter gradually changes it's output value. However as figure 4.3 shows, the difference between the two filters is minimal. In total the average difference between them was less than 1 per cent each frame.

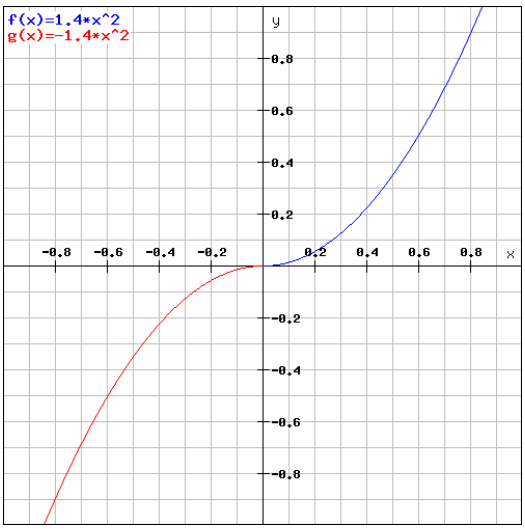
4.2.4 Velocity curve

As stated earlier, the user can influence the velocity of the Sphero by manipulating the distance between his left elbow and hand along the y-axis. In our calculations, we used the current distance between the two mentioned joints and the maximum distance between them. The speed of the Sphero can be adjusted by a numerical value between 0 and 1, 0 being motionless and 1 being full speed. Thus, the speed at which the Sphero goes is proportionate to the ratio of the distance and the maximum distance between the elbow and the hand joint.

At first, we varied this velocity linearly. Meaning that the velocity percentage of the Sphero is the same as the ratio of the joints. To check if our way of controlling the Sphero works well and to finetune certain aspects, we conducted a few pilot studies with ourselves and an occasional outside person. In these pilots we that this linear variation is not the optimal solution as the users could not fine tune the velocity at low speeds while less fine tuning is necessary at high speeds. More specifically, the user more often wanted to make the distinction between driving at say 20% and 30% speed than make a difference between 80% and 90%. As a solution, we changed to a quadratic function (formula 4.1). As seen in figure 4.4, The user gains in fine tuning power in the low velocity area at the cost of that power in the high speed area. We also implemented a multiplication factor after the squaring. This multiplication makes sure that very low speeds cannot be used. They are not necessary because at speeds below 10% the Sphero has not enough power to overcome its friction and move at all. Therefore low speeds are never used.

$$f(x) = \begin{cases} 1.4 * x^2 & : x \geq 0 \\ -1.4 * x^2 & : x < 0 \end{cases} \tag{4.1}$$

Figure 4.4: The velocity curve



4.3 GUI

The GUI consists of two windows. The left window as seen in figure 4.2, shows the depth image from the Kinect. The skeleton of the user is painted on top of this image. Figure 4.5 depicts the right window of the GUI. In this window, the user can see various information about what the system thinks the user currently intends:

1. Speed. The window shows which speed the system currently calculates from the user's pose. This is displayed as 'Speed: ' followed by its value.

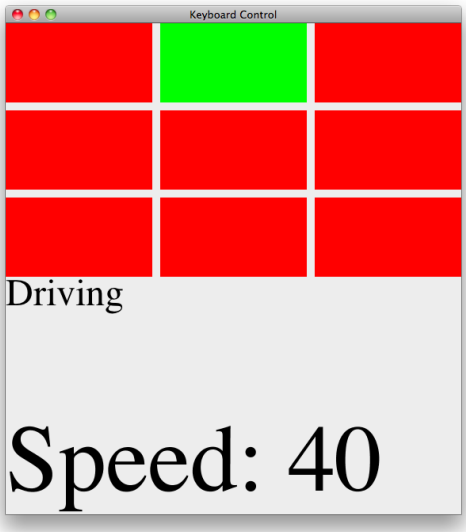
2. Direction. This part of the interface consists of nine colored blocks. One of them is colored green. This indicates in which of the eight cardinal directions the robot will be sent. The other blocks will be colored red to indicate that they are not currently in use. The block in the center is colored when the robot is ordered not to move. Also, when the user is calibrating, the green block indicating the direction, will be colored yellow instead.

3. Other feedback. Under the colored blocks, there is room for a line of textual feedback. This feedback can be one of the following lines:

driving, when all is well and the user is driving the Sphero. Calibrating, when the user is calibrating the Sphero. Not moving, when all is well and the user is not moving the Sphero. No user detected, when the Kinect does not recognize a user in it's vision. And finally: Do the starting pose, when the Kinect has detected a user, but needs him to stand in this particular pose to be able to start tracking his skeleton.

The GUI is refreshed at 10 frames per second and with no noticeable delay. So the user can always see up to date feedback on the screen.

Figure 4.5: The right feedback window



Chapter 5

Results

In this chapter we will discuss our results. First we will discuss the results of our research questions, after that a number of control questions are analyzed.

5.1 Research Questions

To answer our first pair of research questions, we measured in total time till task completion (tttc) per condition. We performed a 2x2 repeated measures ANOVA with mode of control and viewpoint as within subject variables. As for mode of control, we found that the use of the tilt application (mean = 3:56, SD = 1.37) outperformed the use of gestures with the Kinect (mean = 8:48, SD 3:13). The found effect was significant and the effect size was large ($F(1,7) = 19.679$, $p = .003$, $eta^2 = 0.738$). As for viewpoint, the results showed us a better performance for the actual presence condition (mean = 5:18, SD = 2:47) than the teleoperation condition (mean = 7:26, SD = 3:55). Again, we found that this effect was significant and large ($F(1, 7) = 11.680$, $p = .011$, $Eta^2 = 0.625$). There was no significant interaction found between the independent variables ($F(1,7) = 0.765$, $p = 0.411$, $Eta^2 = 0.099$).

Our second research question was regarding the self-reported user experience (UX) of participants. In our questionnaire we asked 5 questions for each of the levels of our independent variables, these are tilt application use, Kinect use, actual presence and teleoperation. This results in a total of 20 questions regarding UX, all on a five-point Likert scale. Of these five questions per variable level, two of them were asked negatively, so their values had to be mirrored. The answers were then summed to get a UX value for

each variable level. To answer our second pair of research questions we used 2 separate repeated measured ANOVA's. For the first analysis, we used the summed UX values of the Kinect(mean = 13.38, SD = 3.852) and the tilt application (mean = 21.00, SD = 1.195) conditions. It was found that users reported a significantly higher UX in the tilt application condition than for using the Kinect ($F(1,7) = 44.073$, $p = .000$, $eta^2 = .863$). This effect is large. The other ANOVA analyzed the difference in the reported UX values of viewpoint conditions: actual presence (mean = 20.25, SD = 2.315) and teleoperation (mean = 17.50, SD = 3.625). However, this effect was not significant ($F(1,7) = 3.221$, $p = 0.116$, $eta^2 = 0.315$).

5.2 Control Questions

In our questionnaire we asked our participants a number of control questions to determine possible factors that influence their performance. Other control questions are only important for exposing possible flaws in the experiment. We have listed the descriptive statistics of the control questions in the table below.

Table 5.1: Descriptive statistics of the control questions

	Minimum	Maximum	Mean	Std Deviation
Question 1	4	5	4.25	0.46
Question 2	1	3	2.25	0.70
Question 3	1	3	1.88	0.64
Question 4	4	4	4.00	0.00
Question 5	1	5	2.25	1.58
Question 6	2	3	2.50	0.53
Question 7	3	5	4.25	0.70
Question 8	2	5	3.63	1.40
Question 9	3	5	4.38	0.74
Question 10	2	4	2.87	0.64
Question 11	3	4	3.62	0.51
Question 12	1	5	3.38	1.73
Question 13	1	5	2.50	1.60
Question 14	1	5	3.37	1.50

The control questions which may be influence are analyzed here.

- Question 7 (After practice it was clear to me which gestures I had to use.) did not have a significant effect on the performance while using the Kinect. In both the teleoperation with Kinect ($F(2,8) = 0.057$, $p = .945$, $eta^2 = 0.022$) and the actual presence with Kinect conditions ($F(2,8) = 1.668$, $p = .279$, $eta^2 = .400$).

- Question 8 (I had no trouble remembering the gestures.) did not have a significant effect on the performance while using the Kinect. In both the teleoperation with Kinect ($F(2,8) = 0.041$, $p = .960$, $eta^2 = 0.016$) and the actual presence with Kinect conditions ($F(2,8) = 0.109$, $p = .899$, $eta^2 = .042$).

- Question 12 (I noticed a delay in the camera feed.) did not have a significant effect on the performance in the teleoperation conditions. In both the teleoperation with Kinect ($F(3,8) = 1.954$, $p = .263$, $eta^2 = 0.594$) and the teleoperation with the tilt application conditions ($F(3,8) = 0.483$, $p = .712$, $eta^2 = .266$).

- Question 13 (At the Kinect and actual presence condition I made use of the feedback from the laptop.) did not have a significant effect on the corresponding condition, actual presence with use of the Kinect ($F(3,8) = 0.094$, $p = .959$, $eta^2 = .066$).

- Question 14 (At the Kinect and teleoperation condition I made use of the feedback from the laptop.) did not have a significant effect on the corresponding condition, teleoperation with use of the Kinect ($F(3,8) = 1.489$, $p = .345$, $eta^2 = .528$).

Chapter 6

Discussion

In this chapter we will discuss our results, their possible causes and their implications. First the results regarding the differences between the gesture based interface with kinect and the tilt application with smartphone are discussed. Then the results from the comparison between actual presence and teleoperation are discussed. We will finish with a number of possibilities for future research regarding these topics.

6.1 Gesture based versus tilt app control

Our experiment showed that participants performed a lot better while using the tilt application than when using gestures with the Kinect. This confirms our hypothesis. A possible explanation is that tilt control is simply more suited for driving with a remote controlled robot. Similar results were also found by Sanna et al. [13]. As they reported that use of a gesture based interface resulted in the slowest performance. Both results together show a trend that various remote controlled robots may not be controlled in the fastest way by using gestures. However more research with other robots is required to confirm this. Another possible cause for the effect we found is that the gestures set we designed for this experiment was not good enough for this particular setting, despite that we eliminated a number of possible issues with our gesture set by means of our control questions in the questionnaire. We found that the participants did not find the gestures unclear after practicing and they had no significant trouble in remembering them throughout the experiment. Also, making use of the feedback in particular Kinect conditions did not have a significant effect on the performance in that particular condition. Thus, it is not the case that certain participants

performed worse because they forgot or simply did not want this feedback.

It is also possible that the better performance of the tilt application is due to increased user experience with this mode of control. As it was found that the participants preferred the use of the tilt application over the use of the Kinect, we hypothesized the opposite effect. Of course, it is also possible that this causal effect is the other way around. That the users preferred the tilt application and therefore performed better with this mode of control. In informal observations of the participants during the experiment, a number of them showed more frustration when using the gesture based interface. This may be correlated with the lower reported UX in the Kinect conditions.

6.2 Actual Presence versus Teleoperation

As we predicted in our hypotheses, the data show us that participants performed significantly better in the actual presence conditions, looking directly at the Sphero, as opposed to teleoperating it. An obvious explanation would be that the user can make better judgments about speed, time and distance of the Sphero and the obstacle course. Another possibility is that there was a delay between the camera and the television, which caused issues. However, by means of our control questions we found no significant effect of the experienced delay of the camera and teleoperation performance. But this does not mean that the delay did not impair performance. It just means that the extend to which the delay is noticed by the user does not affect his or her performance.

The effect of viewpoint on performance is not caused by increased or decreased UX experience in either of the conditions, as this effect was not proven to be significant. This is against our hypothesis. In our informal conversations we found a small number of participants who expected to find teleoperation easier as it more resembled computer games, with which they were more proficient. Thus they may have felt more at ease when controlling a robot in this setting, increasing their reported UX. This in combination with the fact that not all participants have gaming experience, may have caused the lack of effect found. Another explanation is that our number of participants was somewhat low; only 8 subjects were tested in our experiment. There was a trend found that actual presence is preferred by the participants. Thus, increasing the number of participants may turn this trend into a significant effect.

6.3 Future Research

After the experiment, we held an informal conversation with most participants. A couple of them suggested various other gesture sets. For future research, it might be worth comparing a number of different gesture sets to determine which one works best. Or perhaps let users come up with their own gestures and let them test their sets immediately. Another possibility is to make use of an entirely different gesture paradigm. In section 2.2 we discussed a "point to go here" gesture set. It may be worth investigating whether this paradigm has some viability.

It is interesting to note that in our informal conversations after the experiment, some participants indicated that they found the feedback helpful, but that they did not find it very accessible in all situations as the feedback and the Sphero were sometimes too far apart. It may be an interesting topic for future research to determine the best way of providing feedback about the robot while controlling it directly. For example, with the use of Google Glass the user can always have the feedback within their field of vision while still tracking the robot with their own eyes.

Furthermore, the use of a gesture based interface is not restricted to the Kinect. New devices are constantly being developed. For example, a novel gesture input device is the LEAP motion. It is worth investigating what the potential advantages and drawbacks are of this new interface.

For other research, it may be of interest to make more use of the extra possibilities of the actual presence near the controlled robot. In our study, the participants stayed in the same spot during each condition. But of course it may be desirable that a user is able to follow a robot around while controlling it with gestures. However, this should be combined with a different gesture interface than the Kinect as this device is in a fixed position and has limited abilities to track a moving user. Again, a possible example here is the LEAP motion.

Lastly, a user could be immersed in a virtual reality environment with a simulation of the remote controlled robot. In this setting a user effectively teleoperates a robot while the situation is comparable to actual presence. Research is required to determine the extent to which this immersion is effective and what its benefits and drawbacks are.

Bibliography

[1] T.B. Sheridan. Telerobotics. *Automatica*, 25(4):487 – 507, 1989.

[2] Terrence Fong and Charles Thorpe. Vehicle teleoperation interfaces. *Autonomous robots*, 11(1):9–18, 2001.

[3] Jong Kwang Lee, Hyo Jik Lee, Byung Suk Park, and Kiho Kim. Bridge-transported bilateral master–slave servo manipulator system for remote manipulation in spent nuclear fuel processing plant. *Journal of Field Robotics*, 29(1):138–160, 2012.

[4] Kiri L Wagstaff. Smart robots on mars: Deciding where to go and what to see. *Juniata Voices*, pages 5–13, 2008.

[5] Marc Hassenzahl. User experience (ux): towards an experiential perspective on product quality. In *Proceedings of the 20th International Conference of the Association Francophone d'Interaction Homme-Machine*, pages 11–15. ACM, 2008.

[6] Michael Nielsen, Moritz Störring, Thomas B Moeslund, and Erik Granum. A procedure for developing intuitive and ergonomic gesture interfaces for hci. In *Gesture-Based Communication in Human-Computer Interaction*, pages 409–420. Springer, 2004.

[7] Alan Dix. Human–computer interaction: A stable discipline, a nascent science, and the growth of the long tail. *Interacting with Computers*, 22(1):13–27, 2010.

[8] Melody Y Ivory and Marti A Hearst. The state of the art in automating usability evaluation of user interfaces. *ACM Computing Surveys (CSUR)*, 33(4):470–516, 2001.

[9] Jakob Nielsen. Usability inspection methods. In *Conference companion on Human factors in computing systems*, pages 413–414. ACM, 1994.

[10] Vladimir I Pavlovic, Rajeev Sharma, and Thomas S. Huang. Visual interpretation of hand gestures for human-computer interaction: A review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):677–695, 1997.

[11] Sushmita Mitra and Tinku Acharya. Gesture recognition: A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(3):311–324, 2007.

[12] Pierre Rouanet, Jérome Béchu, Pierre-Yves Oudeyer, et al. A survey of interfaces using handheld devices to intuitively drive and show objects to a social robot. In *IEEE International Conference on Robot and Human Interactive Communication*, 2009.

[13] Andrea Sanna, Fabrizio Lamberti, Gianluca Paravati, and Federico Manuri. A kinect-based natural interface for quadrotor control. *Entertainment Computing*, 2013.

[14] Farzin Farhadi-Niaki, Jesse Gerroir, Ali Arya, S Ali Etemad, Robert Laganière, Pierre Payeur, and Robert Biddle. Usability study of static/dynamic gestures and haptic input as interfaces to 3d games. In *ACHI 2013, The Sixth International Conference on Advances in Computer-Human Interactions*, pages 315–323, 2013.

[15] Daniel Huber, Herman Herman, Alonzo Kelly, Pete Rander, and Jason Ziglar. Real-time photo-realistic visualization of 3d environments for enhanced tele-operation of vehicles. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1518–1525. IEEE, 2009.

[16] Mary Lou Maher, Kazjon Grace, and Berto Gonzalez. A design studio course for gesture aware design of tangible interaction.

[17] J Pieter Marsman and Louis G Vuurpijl. Sharing control with a robotic ball. 2013.

[18] Jens Stokholm Høngaard, Jens Kristian Nielsen, Simon Højlind, Stine Eklund, Andreas Møgelmose, and Thomas Moeslund. Sphero labyrinth theme: Visual computing project period.

[19] Mare Hassenzahl, Axel Platz, Michael Burmester, and Katrin Lehner. Hedonic and ergonomic quality aspects determine a software's appeal. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 201–208. ACM, 2000.

[20] Niklas Gavelin. Sphero api. `https://github.com/nicklasgav/Sphero-Desktop-API`. Accessed: 2014-07-26.

[21] Angelos Barmpoutis. Tensor body: Real-time reconstruction of the human body and avatar synthesis from rgb-d. *Cybernetics, IEEE Transactions on*, 43(5):1347–1356, 2013.

Appendix A

Questionnaire

A.1 Personal Information

Participant number: _____ Date: _____
Age: _____ Sex: male/female

A.2 Experiment

Specify how much you agree with the following statements (1 means strongly disagree. 5 means strongly agree. 3 means neutral)
Teleoperation implies when you were controlling the Sphero while looking at the tv. Actual presence implies when you were controlling the SPhero while directly looking at it.

		1	2	3	4	5
1.	I liked participating in this experiment.					
2.	The obstacle course was too difficult.					
3.	The obstacle course was too hard.					
4.	The obstacle course was fun.					
5.	I had previous experience with the Sphero.					
6.	The Sphero moves intuitively.					
7.	After practice it was clear to me which gestures I had to use.					
8.	I had no trouble remembering the gestures.					
9.	I had enough time to practice.					
10.	After practice, I could handle the Sphero well.					
11.	After the experiment I could handle the Sphero well.					
12.	I noticed a delay in the camera feed.					
13.	At the Kinect and actual presence condition I made use of the feedback from the laptop.					
14.	At the Kinect and teleoperation condition I made use of the feedback from the laptop.					

		1	2	3	4	5
15.	I enjoyed using the Kinect.					
16.	Using the Kinect was tiring.					
17.	Using the Kinect was intuitive.					
18.	Using the Kinect was easy.					
19.	Using the Kinect was cumbersome.					
20.	I enjoyed using the iPhone.					
21.	Using the iPhone was tiring.					
22.	Using the iPhone was intuitive.					
23.	Using the iPhone was easy.					
24.	Using the iPhone was cumbersome.					
25.	I enjoyed teleoperating the Sphero.					
26.	Teleoperating was tiring.					
27.	Teleoperating was intuitive.					
28.	Teleoperating was easy.					
29.	Teleoperating was cumbersome.					
30.	I enjoyed controlling the Sphero with actual presence.					
31.	Actual presence was tiring.					
32.	Actual presence was intuitive.					
33.	Actual presence was easy.					
34.	Actual presence was cumbersome.					

A.3 Open Question

Do you have any remarks?

Thank you for participating.

A.4 Objective measurements

Condition 1: _____ tttc: _____ _____ _____ _____
Calibrations: _____
Concedes: _____

Condition 2: _____ tttc: _____ _____ _____ _____
Calibrations: _____
Concedes: _____

Condition 3: _____ tttc: _____ _____ _____ _____
Calibrations: _____
Concedes: _____

Condition 4: _____ tttc: _____ _____ _____ _____
Calibrations: _____
Concedes: _____

Start Time Experiment: _____
End Time Experiment: _____

Appendix B

Instructions

B.1 Instructies Sphero experiment

Introductie

Bedankt voor het meedoen aan mijn experiment met de Sphero. Eerst zal ik je uitleg geven over het verloop en het doel van het experiment.

Het doel van dit experiment is dat jij de Sphero over de hindernisbaan rijdt met behulp van verschillende bestuur- en kijkmethoden. Wat ik daadwerkelijk onderzoek is welke methode mensen het best kunnen en het leukst vinden.

Oefenen tilt

Met de tilt functie kantel je de tablet/telefoon om zo de Sphero een bepaalde kant op te laten gaan. Hoe verder je kantelt, hoe sneller de Sphero gaat. Je kunt ook de snelheid aanpassen door een plek op dit balkje aan te raken.

Soms kan het zijn dat als je de Sphero naar voren stuurt dat hij een afwijking vertoont. Dat kan komen doordat de Sphero niet goed gecalibreerd meer is. Dat wil zeggen, dan is voor en achter niet hetzelfde voor jou als voor de Sphero. Je kunt hem opnieuw calibreren met behulp van deze knop onderin beeld. Je moet dan zorgen dat het blauwe lichtje achter is. Dit hoef je normaal gesproken alleen te doen als je ergens hard tegenaan bent gereden of als de Sphero geslipt heeft.

Je mag nu even, ongeveer 3 minuten, rondrijden om te oefenen met deze manier van besturen.

Oefenen gestures

Je gaat de Sphero nu besturen door gebaren te maken naar de Kinect. Je gaat eerst voor de Kinect staan en houdt beide handen in de lucht. De Kinect tekent nu een blauw skelet op de plaats waar jij bent.

Met je linkerhand bestuur je de snelheid. In een horizontale houding staat de Sphero stil. Als je je hand omhoog doet, zal hij vooruit rijden en als je je hand omlaag doet, juist achteruit. Met je rechterhand bepaal je de richting. Als je je hand vooruit houdt, zal de Sphero ook vooruit gaan (of juist achteruit). Als je je hand naar links of rechts doet, zal de Sphero ook die kant op gaan. Als je op de laptop kijkt, zie je aangegeven wat de computer op dit moment denkt welke kant je op wilt en met welke snelheid.

Ook met de Kinect moet je misschien soms de Sphero opnieuw calibreren. Als je beide handen omhoog houdt, ga je in de calibreer modus. De Sphero veranderd dan van kleur net als het blokje op het scherm. Je kunt dan met je rechterhand de Sphero linksom of rechtsom laten draaien totdat het blauwe LED-lampje naar je toe staat. Dan doe je weer beide hand omhoog om verder te rijden.

Als je even wilt pauzeren of gewoon dat de Sphero even helemaal stopt, kun je ook beide handen langs je zij laten hangen. Maar als je dan weer verder wilt, kun je beter eerst je linkerhand horizontaal houden zodat de snelheid nul is en daarna pas met je rechterhand beginnen met sturen. Anders kan het zijn dat de Sphero de Kinect denkt dat je hard achteruit wilt met een bepaalde richting.

Je mag ook even met de Kinect oefenen om aan de besturing gewend te raken.

Teleoperation

Tot nu toe heb je steeds naast de Sphero gestaan en er direct naar gekeken. In de andere conditie, genaamd teleoperation, ga je via vier webcams naar de Sphero kijken in plaats van direct. Op deze manier ga je zometeen de Sphero ook één keer besturen met de app en één keer met de Kinect. Helaas kun je bij het caliberen van de Sphero op deze manier niet altijd even goed zien welke kant het LED lampje op staat. Ik zal dan even voor je kijken en zeggen wanneer hij goed staat. Je mag namelijk niet zelf naar de baan kijken. Je mag nu ook even één minuutje oefenen via de webcams met de tilt functie.

Nu ga je voor de echt een voor een de condities doen. Als laatste wil ik nog zeggen dat als je een obstakel echt niet overkomt mag je hem overslaan, maar probeer wel altijd je best te doen. Heb je nog vragen voordat we beginnen?

Questionnaire

Alvast bedankt voor je deelname. Wil je nu nog even deze vragenlijst invullen? Dat is het laatste deel van het experiment.

Appendix C

Code

C.1 Class Kinect

```
package j4kdemo.kinectviewerapp;

import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.Collection;
import javax.swing.JLabel;
import kinect.FeatureExtractor;
import se.nicklasgavelin.bluetooth.Bluetooth;
import se.nicklasgavelin.bluetooth.Bluetooth.EVENT;
import se.nicklasgavelin.bluetooth.BluetoothDevice;
import se.nicklasgavelin.bluetooth.BluetoothDiscoveryListener;
import se.nicklasgavelin.sphero.Robot;
import se.nicklasgavelin.sphero.command.RGBLEDCommand;
import se.nicklasgavelin.sphero.exception.InvalidRobotAddressException;
import se.nicklasgavelin.sphero.exception.RobotBluetoothException;
import shared.CommandSender;
import shared.MyFileWriter;
import shared.Prefs;
import shared.View;
import edu.ufl.digitalworlds.j4k.DepthMap;
import edu.ufl.digitalworlds.j4k.J4KSDK;
import edu.ufl.digitalworlds.j4k.Skeleton;
/*
 * Copyright 2011, Digital Worlds Institute, University of
 * Florida, Angelos Barmpoutis.
 * All rights reserved.
 *
 * When this program is used for academic or research purposes,
 * please cite the following article that introduced this Java library:
 *
 * A. Barmpoutis. "Tensor Body: Real-time Reconstruction of the Human Body
 * and Avatar Synthesis from RGB-D', IEEE Transactions on Cybernetics,
 * October 2013, Vol. 43(5), Pages: 1347-1356.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *     * Redistributions of source code must retain this copyright
 * notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce this
 * copyright notice, this list of conditions and the following disclaimer
 * in the documentation and/or other materials provided with the
 * distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
public class Kinect extends J4KSDK implements BluetoothDiscoveryListener, KeyListener{

      ViewerPanel3D viewer=null;
```

```
	JLabel label=null;
	boolean mask_players=false;
	public void maskPlayers(boolean flag){mask_players=flag;}

	private int skellyInArray = 0;
	double [][] joints = new double [6][3];
	private FeatureExtractor fe;
	private CommandSender cs;
	private Robot robot;
	private int counter = 0;
	private boolean stillCalibratingSpheroPose = false, skeletonEventTag = false;
	private MyFileWriter fw = new MyFileWriter ();

	public Kinect()
	{
		super();
		  if(!Prefs.TESTWITHOUTSPHERO){
			Bluetooth bt = new Bluetooth(this, Bluetooth.SERIAL_COM );
			String id = Prefs.BLUETOOTHADDRESS;
			BluetoothDevice btd = new BluetoothDevice( bt, "btspp://" + id +
":1;authenticate=true;encrypt=false;master=false" );
			try {
				robot = new Robot( btd );
			} catch (InvalidRobotAddressException e) {
				e.printStackTrace();
			} catch (RobotBluetoothException e) {
				e.printStackTrace();
			}
			if( robot.connect())
				  robot.sendCommand(new RGBLEDCommand(255, 255, 255));
			else{
				System.out.println("No Connection.");
				robot = null;
			}
		  }else{
			System.out.println("Testing without Sphero. To run with a Sphero,
set the boolean 'TESTWITHOUTSPHERO' to false in Prefs (Preferences).");
		  }
		  cs = new CommandSender(robot);
		  fe = new FeatureExtractor();
		  View view = new View(cs, this);
		  cs.addObserver(view);
		  cs.calibrate(0);
		  new Thread (cs).start();
	}

	public Kinect(int index)
	{
		super(index);
	}

	public void setViewer(ViewerPanel3D viewer){this.viewer=viewer;}

	public void setLabel(JLabel l){this.label=l;}

	@Override
	public void onDepthFrameEvent(short[] depth, int[] U, int V[]) {
		if(!skeletonEventTag)
			cs.setTrackingSkeleton(false);
		skeletonEventTag = false;
		if(viewer==null || label==null)return;
		float a[]=getAccelerometerReading();
```

```
	label.setText(((int)(a[0]*100)/100f)+","+((int)(a[1]*100)/100f)+","+((int)(a[2]*1
00)/100f));
		DepthMap map=new DepthMap(depthWidth(),depthHeight(),depth);
		if(U!=null && V!=null) map.setUV(U,V,videoWidth(),videoHeight());
		if(mask_players)map.maskPlayers();
		viewer.map=map;

	}

	@Override
	public void onSkeletonFrameEvent(float[] data, boolean[] flags) {
		skeletonEventTag = true;
		cs.setTrackingSkeleton(true);
		if(viewer==null || viewer.skeletons==null)return;
		for(int i=0;i<Kinect.NUI_SKELETON_COUNT;i++)
			viewer.skeletons[i]=Skeleton.getSkeleton(i, data, flags);
		Skeleton skelly;
		if (viewer.skeletons.length > 0) {

			skelly = viewer.skeletons[skellyInArray];
			while(!skelly.isTracked()){
				skellyInArray++;
				if(skellyInArray >= viewer.skeletons.length)
					skellyInArray = 0;
				skelly = viewer.skeletons[skellyInArray];
			}
			joints [0] = skelly.get3DJoint(Skeleton.ELBOW_LEFT);
			joints [1] = skelly.get3DJoint(Skeleton.HAND_LEFT);
			joints [2] = skelly.get3DJoint(Skeleton.SHOULDER_LEFT);
			joints [3] = skelly.get3DJoint(Skeleton.ELBOW_RIGHT);
			joints [4] = skelly.get3DJoint(Skeleton.HAND_RIGHT);
			joints [5] = skelly.get3DJoint(Skeleton.SHOULDER_RIGHT);

			if(Prefs.WRITEOUTPUT){
				fw.write(joints[3][0]);
				fw.write(joints[3][1]);
				fw.write(joints[4][0]);
				fw.write(joints[4][1]);
				fw.writeNewLine();
			}
			controlSphero();
		} else
			System.out.println("No skelly detected.");
	}

	private void controlSphero (){
		 if(counter < 1){
		   if(fe.calibratingSpheroPose(joints)){
				System.out.println("Calibrating Sphero pose = true");

			  if(!stillCalibratingSpheroPose){
				stillCalibratingSpheroPose = true;
				if(cs.getCalibrating()){
					cs.stop();
				} else {
					cs.calibrate(0);
				}
			  }
		   } else {
				stillCalibratingSpheroPose = false;
				if(cs.getCalibrating()){
```

```
                                    if(fe.idle(joints)){
                                          cs.calibrate(0);
                                    }else{

cs.calibrate(fe.extractCalibrationDirection(joints));
                                    }
                              }else {
                                    if(fe.idle(joints))
                                          cs.stop();
                                    else{
                                         int [] results = fe.extractDirAndVel(joints);
                                         cs.setCommand(results[Prefs.DIRECTION],
results[Prefs.VELOCITY], results[Prefs.INVERTED]);
                                    }
                              }
                        }
                        if(cs.getCalibrating())
                              counter = 20;
                        else
                              counter = 1;
                  }
                  counter--;
      }

      @Override
      public void onVideoFrameEvent(byte[] data) {
            if(viewer==null || viewer.videoTexture==null) return;
            viewer.videoTexture.update(videoWidth(), videoHeight(), data);
      }

      @Override
      public void deviceDiscovered(BluetoothDevice arg0) {
      }

      @Override
      public void deviceSearchCompleted(Collection<BluetoothDevice> arg0) {
      }

      @Override
      public void deviceSearchFailed(EVENT arg0) {
      }

      @Override
      public void deviceSearchStarted() {
      }

      @Override
      public void keyPressed(KeyEvent arg0) {
            if(Prefs.WRITEOUTPUT){
                  fw.close();
                    fe.fileWriter.close();
                    System.out.println("Output file Closed.");
                  }
      }

      @Override
      public void keyReleased(KeyEvent arg0) {
      }

      @Override
      public void keyTyped(KeyEvent arg0) {
      }
```

```
}
```

C.2 Class FeatureExtractor

```
package kinect;

import shared.*;

public class FeatureExtractor {


	private static final int LEFTELBOW = 0,
	                                     LEFTHAND = 1,
	                                     LEFTSHOULDER = 2,
	                                     RIGHTELBOW = 3,
	                                     RIGHTHAND = 4,
	                                     RIGHTSHOULDER = 5,
	                                     X = 0,
	                                     Y = 1;
	private double distElbowHand = 0;
	private Filter filterMedSpeed = new MedianFilter();
	private Filter filterAvSpeed = new AverageFilter();
	public MyFileWriter fileWriter;

	private double differenceFilteredCum = 0, differenceFinalCum = 0;
	private int nrOfMeasurements = 0;

	public FeatureExtractor(){
		if(Prefs.WRITEOUTPUT){
			fileWriter = new MyFileWriter("Output all");
		}
	}

	public void calibrateKinect(double [][] joints){
		double distY = joints[RIGHTHAND][Y] - joints [RIGHTELBOW][Y];
		double distX = joints[RIGHTHAND][X] - joints[RIGHTELBOW][X];
		distElbowHand = Math.sqrt(Math.pow(distY, 2) + Math.pow(distX, 2));
		System.out.println("Distance between Elbow and Hand = " + distElbowHand);
		System.out.println("Kinect calibrated");
	}

	public int [] extractDirAndVel(double [][] joints){
		int direction = 0;
		int velocity = 0;
		int inverted = 0;
		velocity = extractVelocity(joints);
		direction = extractDirection(joints);
		if(velocity < 0){
			velocity = -velocity;
			direction = invert(direction);
			inverted = 1;
		}
		int [] ints = {direction, velocity, inverted};
		return ints;
	}

	private int extractDirection(double [][] joints){
		double dist = joints [RIGHTELBOW][X] - joints[RIGHTHAND][X];
		double relativeDist = dist/distElbowHand;
		double relativeDistFiltered = relativeDist;
		if(relativeDistFiltered > 0.5){
			return Prefs.LEFT;
```

```
        } else if(relativeDistFiltered > 0.3){
                return Prefs.LEFTFRONTLEFT;
        }else if(relativeDistFiltered > 0.15){
                return Prefs.FRONTLEFT;
        }else if(relativeDistFiltered > 0){
                return Prefs.FRONTFRONTLEFT;
        }else if(relativeDistFiltered > -0.2){
                return Prefs.FRONT;
        }else if(relativeDistFiltered > -0.35){
                return Prefs.FRONTFRONTRIGHT;
        }else if(relativeDistFiltered > -0.5){
                return Prefs.FRONTRIGHT;
        }else if(relativeDistFiltered > -0.7){
                return Prefs.RIGHTFRONTRIGHT;
        }else{
                return Prefs.RIGHT;
        }
}

public int extractCalibrationDirection(double[][] joints){
        int dir = extractDirection(joints);
        if(dir == Prefs.LEFTFRONTLEFT || dir == Prefs.LEFT)
                return 360 - Prefs.CALIBRATIONSPEEDKINECT;
        if(dir == Prefs.FRONTLEFT || dir == Prefs.FRONTFRONTLEFT)
                return 360 - (Prefs.CALIBRATIONSPEEDKINECT/2);
        if(dir == Prefs.RIGHTFRONTRIGHT || dir == Prefs.RIGHT)
                return Prefs.CALIBRATIONSPEEDKINECT;
        if(dir == Prefs.FRONTRIGHT || dir == Prefs.FRONTFRONTRIGHT)
                return Prefs.CALIBRATIONSPEEDKINECT/2;
        return 0;
}

private int extractVelocity(double [][] joints){

        double dist = joints[LEFTHAND][Y] - joints[LEFTELBOW][Y];
        nrOfMeasurements++;
        double relativeDist = ((dist/distElbowHand));
        double filteredVelMed = filterMedSpeed.filter(relativeDist);
        double filteredVelAv = filterAvSpeed.filter(relativeDist);
        double differenceFiltered = filteredVelMed - filteredVelAv;
        differenceFilteredCum += Math.abs(differenceFiltered);
        double adjustedVelMed = speedCurve(filteredVelMed);
        double adjustedVelAv = speedCurve(filteredVelAv);
        int steppedVelMed = makeStepwise(adjustedVelMed);
        int steppedVelAv = makeStepwise(adjustedVelAv);
        int differenceFinal = steppedVelMed - steppedVelAv;
        differenceFinalCum += Math.abs(differenceFinal);

        if(Prefs.WRITEOUTPUT){
                fileWriter.write(joints[LEFTHAND][Y]);
                fileWriter.write(joints[LEFTELBOW][Y]);
                fileWriter.write(relativeDist);//
                fileWriter.write(filteredVelMed);
                fileWriter.write(filteredVelAv);//
                fileWriter.write(differenceFiltered);//
                fileWriter.write(adjustedVelMed);
                fileWriter.write(adjustedVelAv);//
                fileWriter.write(steppedVelMed/100.0);
                fileWriter.write(steppedVelAv/100.0);
                fileWriter.write(differenceFinal/100.0);
                fileWriter.write(differenceFilteredCum/(nrOfMeasurements));//
                fileWriter.write(differenceFinalCum/(nrOfMeasurements));
```

```
                fileWriter.write("\n");
        }
        return steppedVelMed;
}

private double speedCurve(double x){
        double y;
        if(x < 0)
                y = -Math.pow(x, 2);
        else
                y = Math.pow(x,  2);
        return y*1.4;
}

private int makeStepwise (double x){
        boolean negative = false;
        if(x < 0){
                negative = true;
                x = -x;
        }
        int y = (int) Math.round(100*x);
        int z  = y%10;
        if(z < 3){
                z = 0;
        }else if(z < 8){
                z = 5;
        }else
                z = 10;
        int w = z + (y/10)*10;
        if(w > 100)
                w = 100;
        if(negative)
                w = -w;
        return w;
}

private int invert (int dir){
        if(dir == Prefs.LEFTFRONTLEFT)
                return Prefs.LEFTLEFTREAR;
        if(dir == Prefs.FRONTLEFT)
                return Prefs.LEFTREAR;
        if(dir == Prefs.FRONTFRONTLEFT)
                return Prefs.REARLEFTREAR;
        if(dir == Prefs.FRONT)
                return Prefs.REAR;
        if(dir == Prefs.FRONTFRONTRIGHT)
                return Prefs.REARRIGHTREAR;
        if(dir == Prefs.FRONTRIGHT)
                return Prefs.RIGHTREAR;
        if(dir == Prefs.RIGHTFRONTRIGHT)
                return Prefs.RIGHTRIGHTREAR;
        return dir;
}

public boolean idle (double [][] joints){
        if(joints[RIGHTELBOW][Y] - joints[RIGHTHAND][Y] > distElbowHand/2){
                return true;
        }
        return false;
}

public boolean calibratingSpheroPose(double [][] joints){
```

```
            if(joints[LEFTELBOW][Y] > joints[LEFTSHOULDER][Y] &&
               joints[RIGHTELBOW][Y] > joints[RIGHTSHOULDER][Y] &&
               (joints[LEFTHAND][Y] - joints[LEFTELBOW][Y]) >
distElbowHand*Prefs.CALIBRATINGPOSESENSITIVITY &&
               (joints[RIGHTHAND][Y] - joints[RIGHTELBOW][Y]) >
distElbowHand*Prefs.CALIBRATINGPOSESENSITIVITY ){
                   calibrateKinect(joints);
                   return true;
            }
            return false;
      }
}
```

C.3 Class CommandSender

```
package shared;

import java.util.Observable;
import se.nicklasgavelin.sphero.Robot;
import se.nicklasgavelin.sphero.RobotListener;
import se.nicklasgavelin.sphero.command.CalibrateCommand;
import se.nicklasgavelin.sphero.command.CommandMessage;
import se.nicklasgavelin.sphero.command.FrontLEDCommand;
import se.nicklasgavelin.sphero.command.RGBLEDCommand;
import se.nicklasgavelin.sphero.command.RollCommand;
import se.nicklasgavelin.sphero.response.InformationResponseMessage;
import se.nicklasgavelin.sphero.response.ResponseMessage;

public class CommandSender extends Observable implements Runnable, RobotListener{

	private int velocity = Prefs.STARTINGSPEED, previousVelocity = 0, direction = 0,
timesCalibrated = 0;
	private Robot robot;
	private boolean calibrating = false, stopFlag = true, userDetected = false,
trackingSkeleton = false, inverted = false, actuallyMoving = false;

	public CommandSender(Robot r){
		robot = r;
		if(robot == null)
			System.out.println("No Sphero initialized. Running anyway.");
		else{
			robot.sendCommand( new RGBLEDCommand( 255, 0, 0 ) );
			robot.sendCommand( new FrontLEDCommand(1));
			robot.addListener(this);
		}
	}

	@Override
	public synchronized void run() {
		while(true){
			sendCommand();
			try {
				wait(Prefs.COMMANDINTERVAL);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
	}

	public void setDirection (int dir){
		direction = dir;
		continueMoving();
	}

	public int getDirection(){
		return direction;
	}

	public void setVelocity (int vel){
		velocity = vel;
		continueMoving();
	}
```

```
	public void addVelocity (){
		velocity += Prefs.SPEEDINCREMENT;
		if(velocity > 100)
			velocity = 100;
		setChanged();
		notifyObservers();
	}

	public void substractVelocity(){
		velocity -= Prefs.SPEEDINCREMENT;
		if(velocity < 0)
			velocity = 0;
		setChanged();
		notifyObservers();
	}

	public int getRoundVelocity(){
		if(inverted)
			return -velocity;
		return velocity;
	}

	public void setCommand(int dir, int vel, int inverted){
		if(inverted == 1){
			this.inverted = true;
		}else{
			this.inverted = false;
		}
		direction = dir;
		velocity = vel;
		continueMoving();
	}

	public synchronized void calibrate(int heading){
		if(!stopFlag){
			stopFlag = true;
			timesCalibrated++;
			System.out.println("Number of times calibrated: " +
timesCalibrated);
			sendCommand(); // To make sure it is not moving while calibrating.
		}
		calibrating = true;
		direction = heading;
		setChanged();
		notifyObservers();
	}

	public void stop(){
		stopFlag = true;
		calibrating = false;
		setChanged();
		notifyObservers();
	}

	public boolean getStop(){
		return stopFlag;
	}

	private void continueMoving(){
		stopFlag = false;
		calibrating = false;
		setChanged();
```

```
			notifyObservers();
		}

		public boolean getCalibrating (){
			return calibrating;
		}

		private synchronized void sendCommand() {
			if(robot != null){
				if(calibrating){
					robot.sendCommand( new FrontLEDCommand(1));
				    robot.sendCommand( new RGBLEDCommand( 155, 155, 0 ) );
					robot.sendCommand(new CalibrateCommand(direction));
					direction = 0;
					previousVelocity = 0;
				}else if (stopFlag){
				    robot.sendCommand( new RGBLEDCommand( 255, 0, 0 ) );
					robot.sendCommand( new RollCommand(direction, 0, false));
					previousVelocity = 0;
				}else if (previousVelocity == 0){
				    robot.sendCommand( new RGBLEDCommand( 255, 0, 0 ) );
					robot.sendCommand( new RollCommand(direction, 0, false));
					previousVelocity = velocity;
				}else{
				    robot.sendCommand( new RGBLEDCommand( 255, 0, 0 ) );
					robot.sendCommand( new RollCommand(direction,
(float)((velocity*1.0)/100), false));
					previousVelocity = velocity;
				}
			}
		}


		public boolean getUserDetected() {
			return userDetected;
		}

		public void setUserDetected(boolean userDetected) {
			this.userDetected = userDetected;
			setChanged();
			notifyObservers();
		}

		public boolean getTrackingSkeleton() {
			return trackingSkeleton;
		}

		public void setTrackingSkeleton(boolean trackingSkeleton) {
			this.trackingSkeleton = trackingSkeleton;
			userDetected = trackingSkeleton;
			setChanged();
			notifyObservers();
		}

		@Override
		public void event(Robot arg0, EVENT_CODE arg1) {
		}

		@Override
		public void informationResponseReceived(Robot r, InformationResponseMessage
response){
		}
```

```
	@Override
	public void responseReceived(Robot arg0, ResponseMessage arg1, CommandMessage
arg2) {
	}

	public boolean getActuallyMoving(){
		return actuallyMoving;
	}
}
```