WiiRobot: Controlling Robots with Wii Gestures

Aysun Başçetinçelik
Department of Computer Science
Brown University
Providence, RI, 02906
aysun@cs.brown.edu

May 15, 2009

Abstract

In this project, we introduce WiiRobot, a simple and reproducable gesture based robot controller. Our proposed system uses the Nintendo Wii Remote Controller to interact with a robot. The user first trains the system using Wii Remote in order to recognize future gestures. For the recognition process, we implemented two simple algorithms; K-Nearest Neighbour and sum of square differences. Our results show that using only a few number of repetitions per gesture is enough for recognition.

1 Introduction

Finding a natural and effective way of communication with robots is important in order to enhance their contribution in our daily lives. For this purpose, there have been various developments in the area of Human-Robot Interaction. It is known that using a computer, a gamepad or a keyboard can be teleoperative but it is still unnatural and difficult to control. Hence, it is still one of the major research directions to design ease-of-use and intuitive modes of communication with robots.

Another form of communication is gestures. Currently, gestures are not just limited to face, hand and body gestures. Gesture controlled applications are becoming more common and we are able to create gestures with our fingers or with a mouse. We use gestures while we are doing work with our laptops and phones. Moreoever, after the introduction of Nintendo Wii Console, we were introduced to a new hardware, Wii Remote Controller, which has the capability of creating gestures and using it for a different intuitive control of video games.

Figure 1: User controlling the robot with Wii Remote

For this project, we aimed to create a simple gesture-

based controller for a mobile robot using Wii Remote. Our method holds promise for applicability as it only requires simple algorithms like K-Nearest Neighbour to interact with the mobile robot whereas the construction of such a robot is not expensive and even within the limits of personal use.

1.1 Related Work

Gesture recognition is an active research area in robotics, as well as in Human-Computer Interaction (HCI) community. Most of the work in HCI focuses on vision based gestures such as hand and face gestures [9], [4]. In another work in robotics, they focus on the whole body gestures recognized by a service robot [6]. Most of these gesture based recognizers are designed to work in indoors; however, research done in [7], which also uses body gesture and speech recognition, is able to work both indoors and outdoors.

Apart from the vision based gesture recognizers, there are a few applications made especially for Wii Remote controllers for gesture recognition. One of these applications is Wii Based Gesture Learning Environment (WiiGLE) [10], which is a Windows application that lets you choose the right features and classifiers for a better recognizer. There are also libraries such as WiiYourself [3] and WiiGee [11] both of which use Hidden Markov Models to recognize gestures. These libraries help the programmer to make his/her own application that uses Wii Remote controller.

2 Background and Platform

We first introduce some definitions and platforms used in this report.

2.1 Wiimote

As previously mentioned, in this project we used the Wii remote controller by Nintendo to get gestures. Wii remote controller (Wiimote), as shown in Figure 2, which has motion sensing capabilities, is the main controller for Nintendo's Wii console. Since its launch in November 2006, people have been exploring ways of using the Wiimote with different applications. Currently, there exist several third-party applications; moreover, open source libraries are provided on different platforms to use Wiimote. Although there are not any official specifications, much information about the hardware and usage can be found online [2], [1].

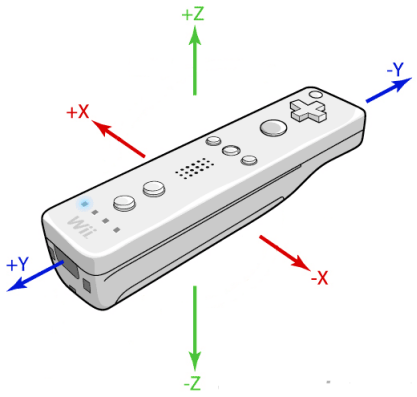


Figure 2: Wiimote controller with axes shown

As explained in [5], inside Wiimote, there are 3 accelerometers, an IR camera with a resolution of 1,024×768 pixels and a 100 Hz refresh rate, a 4 KHz speaker, a vibration motor, wireless Bluetooth connectivity, 5.5 Kbytes on board memory, 2 AA batteries and an expansion port. Also, the controller has 12 buttons each of which can be used to get input from the user. These buttons can be listed as 4 direction buttons (up, down, left, right), plus, minus, home, 1, 2, A and B buttons. Inside Wiimote, each accelerometer uses an unsigned byte to represent the acceleration and it has a +/-3G sensitivity range. Wiimote sends at most 100 points with the information of three accelerometers per second. The information sent from Wiimote also includes the status of buttons, battery, expansion port, IR camera, LEDs and the vibration motor.

2.2 Wii Gesture

A gesture is defined as a non-verbal communication using body motions. In our project, a Wii Gesture is a type of gesture done by using Wiimote. To be more specific, Wii

Gesture consists of accelerometer data points recorded during the time between a button is pressed and released on the Wiimote. These gestures are collected through the Bluetooth connectivity of Wiimote and does not depend on time. Only sequential unique data points are stored as the Wii Gesture. The reasons for this definition are to make gestures independent of the time value and it is hard to get the same data point values sequentially since the accelerometers are very sensitive.

2.3 Robot platform

Our robot is an *Asus eeePC* on an *iRobot Create* [8]. These robots are easy to construct and run. A cheap model of *Asus eeePC* costs around $250 and a simple *iRobot Create* base costs $130, so these robots can cost less than $500 depending on the additional hardware like webcams, cables, Bluetooth USB dongle.

Figure 3: Robot platform with Asus eeePC on iRobot Create Base

The eeePCs have Intel Celeron M CPUs with 1.2 GHz clock speed and 512MB/1GB memory. For the operating system, they come with a version of Xandros Linux. However, since they were incompatible with the robot control server *Player 2.1.1*, we reinstalled the operating system with *Ubuntu Eee* which is configured specially for *Asus eeePCs*. For the camera, we did not use the one on board and used a *Logitech* USB webcam compatible with *Video 4 Linux 2*. *Asus eeePC Surf* models have small hard drives so we had to remove most of the applications to install the necessary libraries. For Player to work, you can check the dependencies from the website. In addition to those libraries, to run WiiRobot we needed to install the *CWiid* library (*libcwiid1*) and Bluetooth (*bluez-gnome*) library. Since eeePCs do not have Bluetooth on board, we used a USB dongle. The robot platform we used during this project is shown in Figure 3 without the camera.

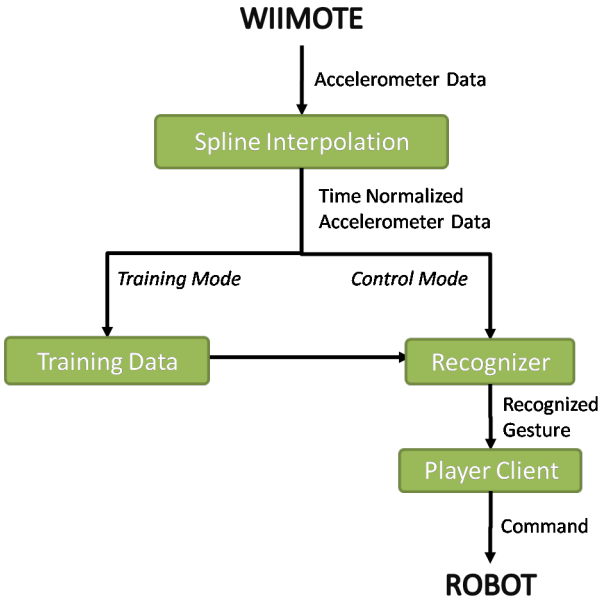


Figure 4: WiiRobot application process

3 Approach

As shown in Figure 4, the process begins with getting the accelerometer data from the Wiimote controller. After receiving this data, a cubic spline interpolation method converts it into the gesture data we used in our program. Then, depending on the current mode of the program, which can be either training or control, the data is stored in memory as the training data or sent to the recognizer to understand which gesture it represents. Recognizer classifies the gesture data using previous training with a classification algorithm. At any point, user can still switch to training if she/he thinks the training data may not be enough. As soon as the gesture is recognized, associated

command is found and sent to the robot for execution.

3.1 Accelerometer data

As we have already discussed above, we collect data from the Wiimote through Bluetooth. For this purpose, we are using a library called *CWiid* which helps us to connect to the Wiimote and get the status of the buttons and accelerometers. The accelerometers give us a data point consisting of three values. These values range from 0 to 255, each representing a different axis shown in Figure 2. Depending on the gesture, the number of data points we get from Wiimote can change dramatically. This data is collected during the time button A is pressed and stored as one continues data till it is released.

3.2 Normalized data

After we get the data from the Wiimote, we process them to add to our training set or recognize the gesture. Since each gesture can take different amount of time, we have to normalize them and have a fixed size independent of their duration. For this purpose, we used spline interpolation in *GNU Scientific Library*. We fixed the number of points per axis for this data to 250. So, time normalized data for a gesture consists of three 250 length arrays which store interpolated values of the accelerometer data.

3.3 Training

There are four available motions to train using the WiiRobot application. To start training, first the training option should be chosen by pressing button 1 on the Wiimote. Then, the move command should be chosen by using any four of the direction buttons. The latest pressed button will be the move command to train. To record the gesture in the training set, one should press the button A while making the gesture and then release it. This way, the data points sent to the program will be stored in memory as one gesture. In our training data, for the recognizer we grouped the data first, according to their time step value (0 to 249) and second, according to their direction.

3.4 Control

Getting the data for control is same as training. The only difference is button 2 should be pressed first to switch to the control mode. After the normalized data is received, it is sent to the recognizer to find the correct gesture.

3.4.1 Recognizer

For the recognizer, we implemented two basic classifiers: K-Nearest Neighbour (KNN) algorithm and sum of square differences method.

For the KNN, we group our data according to their time step value from 0 to 249. In each time step, data is split into four different groups associated with four buttons. For example, if we train Up button five times, we will have five Up button data points at each time step. Then, to recognize the gesture, for each time step, we apply KNN to the point we want to classify among those four groups. We do this process for all 250 points. At the end, we choose the most common group among these points as the recognized gesture.

$$v(t,g) = \begin{cases} 1, h(g) argmin_k [x'(t) - x_k(t)]^2 \\ 0, otherwise \end{cases}$$

$$y' = argmax_g \sum_{t:1}^{250} v(t,g)$$

$h(g)$ *: hash function*
$x'(t)$ *: gesture point at time step t that we are trying to recognize*
$x_k(t)$ *: k-nearest neighbour point at time step t*

For the sum of square differences, we iterate over all gestures that we train and calculate their sum of square differences point by point between the gesture and our trained gesture we want to recognize. As a result, we return the closest gesture in our training data.

$$y' = h(i) argmin \sum_{t:1}^{250} [x'(t) - x_i(t)]^2$$

$h(i)$ *: hash function*
x_i *: trained gesture*

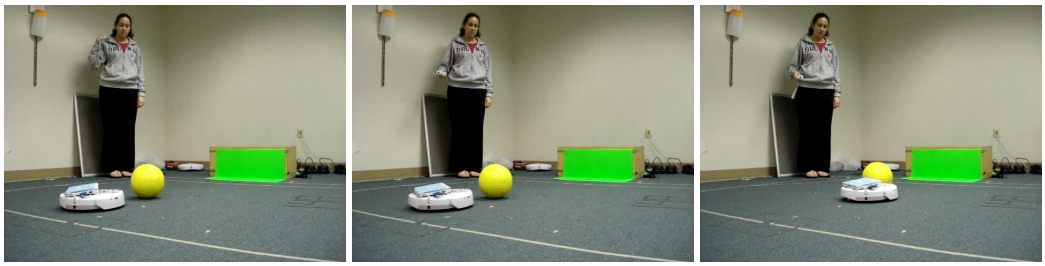
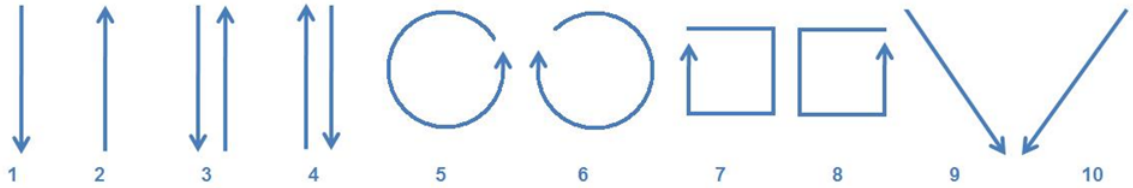


Figure 5: Photos from an experiment video, showing that after the gesture robot moves forward



	Test Gestures				Successful Recognition/Total							
					K-Nearest Neighbour				Sum of Squares Differences			
Buttons	Up	Down	Right	Left	Up	Down	Right	Left	Up	Down	Right	Left
Test1	1	2			10/10	10/10			10/10	10/10		
Test2	3	4			10/10	10/10			10/10	10/10		
Test3	1	2	3	4	8/10	7/10	9/10	9/10	10/10	6/10	10/10	10/10
Test4	1	2	5	6	9/10	10/10	9/10	8/10	10/10	10/10	9/10	10/10
Test5	7	8			10/10	9/10			10/10	10/10		
Test6	7	8	5	6	8/10	8/10	10/10	9/10	6/10	9/10	10/10	9/10
Test7	1	2	9	10	9/10	9/10	8/10	8/10	5/10	10/10	7/10	6/10

Figure 6: Gestures used in tests and the results

3.4.2 Robot Control

After the recognition phase, a command associated with the recognized gesture is sent to the robot. Robot controller is implemented using the *libplayerc++* library. Each of the four direction buttons are associated with a different speed for the robot and calling the *SetSpeed(straightSpeed, angularSpeed)* function. These two speeds can be set before starting the application through command prompt. For the current application, we associated these four motions as following:

- Up - go forward (angular speed is 0)
- $Down$ - go backward (angular speed is 0)
- $Right$ - turn right (straight speed is 0)
- $Left$ - turn left (straight speed is 0)

In the control mode of the application, each time the user makes a gesture, if this gesture is recognized, the robot does the associated movement. By pressing button B, user can stop the robot temporarily. To end the application and stop the robot, Home button should be pressed.

4 Experiments

Our experiments were done mainly in two parts. In the first part, we tested the recognizer separately from the robot and tested the success of recognition; then we tested the complete application with the robot.

For the first part, we used Athlon 64 Dual Core Processor 3800 with 2GB RAM computer to run the application for gesture recognition. Results of these experiments are shown in Figure 6. For each test, we trained the gestures associated with each button five times. The drawings over the table show which gestures are associated with which numbers used in tests. For example, for Test1, we trained gesture 1 for the Up direction button and gesture 2 for the Down direction button on the Wiimote. Out of 10 tries, both of them were easily recognizable. We ran same tests for each of the classifiers.

An interesting test is Test3; in this test, we trained gesture 1 for up, 2 for down, 3 for right and 4 for left buttons and tried to recognize each gesture 10 times. In both classifiers gesture 2 is the least recognized one. This shows that if a gesture is partly inside another gesture it is harder to be recognized. Also in Test6, the wrong recognitions were done mostly between 6 and 7; and in Test7, recognizer had hard time differentiating gestures 1, 9 and 10.

For the second part of experiments, we ran the application on the robot. First, we tried to score a goal and then we took it outside the lab and controlled it in the various floors of the department. For these experiments, we also used other users for controlling the robot. We have observed that people learned how to control the robot very easily and they were excited about using a Wiimote. We also observed an interesting thing that we did not think during our implementation. Gestures change depending on whether a user is right or left handed, and if a right handed user trains the controller, the gesture made by a left handed user may not be recognized. Although they are both doing the same movements, gestures are slightly different. Videos of some of the experiments with robots can be found at http://www.youtube.com/group/wiirobot. An example of controlling the robot with a gesture is shown in the photographs taken from a video in Figure 5.

5 Conclusion and Future Work

In this project, we showed that it is possible to create a gesture based robot controller with Wiimote using simple algorithms like KNN and sum of squares differences. Both of these methods had similar results in the recognizer. We showed that using cheap hardware and easy-to-implement algorithms, interaction between humans and robots can be improved. Our interactions with users showed that using gestures for controlling robots is very intuitive especially when users are able to define their specific gestures. We hope that this project will evolve and further increase the interaction between robots and humans.

For the future work, although all main controls for the application can be done through the Wiimote, a user interface can greatly increase the easiness of usage. Currently, the controller is limited to previously defined five moves, a user interface may be used to change the associated moves with these five buttons. Apart from the user interface, for the recognizer, implementing other classification algorithms may increase the chances of recognizing harder gestures better.

References

[1] Wii brew. http://wiibrew.org/.

[2] Wii li. http://www.wiili.org/index.php/.

[3] Wii yourself, 2008. http://wiiyourself.gl.tter.org/.

[4] M. Hasanuzzaman, T. Zhang, V. Ampornaramveth, H. Gotoda, Y. Shirai, and H. Ueno. Adaptive visual

gesture recognition for human-robot interaction using a knowledge-based software platform. *Robot. Auton. Syst.*, 55(8):643–657, 2007.

[5] J. C. Lee. Hacking the nintendo wii remote. *Pervasive Computing, IEEE*, 7(3):39–45, 2008.

[6] S.-W. Lee. Automatic gesture recognition for intelligent human-robot interaction. In *FGR '06: Proceedings of the 7th International Conference on Automatic Face and Gesture Recognition*, pages 645–650, Washington, DC, USA, 2006. IEEE Computer Society.

[7] M. M. Loper, N. P. Koenig, S. H. Chernova, C. V. Jones, and O. C. Jenkins. Mobile human-robot teaming with environmental tolerance. In *HRI '09: Proceedings of the 4th ACM/IEEE international conference on Human robot interaction*, pages 157–164, New York, NY, USA, 2009. ACM.

[8] L. Miller, A. Bascetincelik, and O. C. Jenkins. Setting up player 2.1.1: Asus eeepc and icreate robot, 2009. http://robotics.cs.brown.edu/projects/player_icreate/.

[9] H. S. Park, E. Y. Kim, S. S. Jang, S. H. Park, M. H. Park, and H. J. Kim. Hmm-based gesture recognition for robot control. *Pattern Recognition and Image Analysis*, pages 607–614, 2005.

[10] M. Rehm, N. Bee, and E. André. Wave like an egyptian accelerometer based gesture recognition for culture specific interactions. In *Procedings of HCI 2008 Culture, Creativity, Interaction*, 2008.

[11] T. Schlömer, B. Poppinga, N. Henze, and S. Boll. Gesture recognition with a wii controller. In *TEI '08: Proceedings of the 2nd international conference on Tangible and embedded interaction*, pages 11–14, New York, NY, USA, 2008. ACM.