# CONTROLLING A VIRTUAL HELICOPTER WITH HAND GESTURES

by

Alper AKBAL (2005102467)

Ismail Ege AKPINAR (2006100175)

Submitted to the Department of Computer Engineering

in partial fulfilment of the requirements for the degree of

Bachelor of Science

in Computer Engineering

Boğaziçi University

Boğaziçi University

June 2011

# Table of Contents

# Abstract

**Project Name** : Controlling a virtual helicopter with hand gestures

**Project Team**: Ege Akpınar & Alper Akbal

**Term** :2010/11 Spring Semester

**Keywords** : Wearable Wireless Sensors, Hand Gesture Recognition

**Summary** :

As the capability and reliability of today's modern computer chips increase, sizes of these devices get smaller. This development enables widely use of programmable chips in daily life. Using a small sized microcontroller board, named Arduino LilyPad, and sensors connected to that board, clothes can be easily equipped with technology that can measure speed, acceleration, light, temperature, and density of some specific gases. LilyPad, owing to its compact and light design, can be used as wearable, which is one of the most significant characteristics of it. It also provides wireless connection via Zigbee or Bluetooth. Aim of the project is to develop a glove that can hand gestures and to control virtual helicopter according to hand gestures. Using C++, Arduino software is developed and loaded into LilyPad to hand gestures using an accelerometer. Hand gesture data is transferred to the computer via a Bluetooth adapter. Based on this hand gesture data, developed software identifies user's hand gestures and accordingly, control the virtual helicopter. A virtual helicopter is used to observe the movements and results of the glove. Successful results obtained from the project would prove that these microcontroller boards and sensors attached to them could be used for a wide variety of purposes in daily life. For instance, conventional interfaces such as touch panels or keyboards could be replaced with such wearable devices. Although this project is based on the idea of flying a helicopter, sensing motion and applying this method to control devices would be very helpful for issues that we encounter in life, like controlling household appliances, creating alternative interfaces etc. We believe that the applications involved in this project could further be extended to security or healthcare areas.

# Components/Tools Used

Definitions of the terms that is referred in this documents:

1. **Arduino:** Arduino is an open-source single-board microcontroller platform, designed to make the process of using electronics in multidisciplinary projects more accessible. It is based on flexible, easy-to-use hardware and software. The hardware consists of a simple open hardware design for the Arduino board with an Atmel AVR processor and on-board I/O support. The software consists of a standard programming language compiler and the boot loader that runs on the board. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments. [2]

1.1 **Arduino LilyPad:** The LilyPad Arduino is a microcontroller board designed for clothes and e-textiles. It can be sewn to fabric and similarly mounted power supplies, sensors and actuators with conductive thread. In this project, Arduino LilyPad hardware is used as a base electonic gadget [3]

1.2 **Accelerometer:** A devide to measure acceleration. Arduino LilyPad Accelerometer, by Sparkfun, is used in the project. It is a three axis accelerometer that is part of the LilyPad system. It is based on the ADXL330 accelerometer from Analog Devices. The LilyPad Accelerometer can detect joint movement as well as inclination and vibration. Readings from the gadget, after being converted to digital form can be used to calculate the true angle (tilt) of the wearer.

2. **Irrlicht:** The Irrlicht Engine is an open source high performance realtime 3D engine written and usable in C++ and also available for .NET languages. It is completely cross-platform, using D3D, OpenGL and its own software renderer, and has all of the state-of-the-art features which can be found in commercial 3D engines such as physics engine and 3d modelling. [1]

3. **Eclipse:** Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system.

# 1. Introduction

Creating devices that can be controlled without conventional interfaces but rather with natural interfaces such as voice, motion or eye is a futuristic phenomena. So much that many science fiction movies (Star Wars, Minority Report, James Bond to name a few) all fascinate their audience with such gadgets. With the recent improvements in the technology, this idea is getting much more reachable. Even, those revolutionary devices can be seen in daily life. Nintendo Wii, Kinect for Xbox 360 and PlayStation Move use motion detection for entertaining people. Smart Houses have voice controlled household gadgets.

Aim of the project is to create an interface that can sense the motion of the hand and to transfer this motion information into a system where it would be interpreted and used as a means for controlling that system. Hence, the project has three main steps:

- Hand gesture recognition using LilyPad Accelerometer
- Real time communication via Bluetooth using LilyPad
- Controlling a virtual helicopter with inputs in the form of gestures of a hand

First of all, LilyPad can be sewn to fabrics, therefore sensors, main board and battery is sewn to a glove, which covers the hand. Hence, every motion of the hand could be measured by the accelerometer on the glove. Then, results from accelerometer is recorded in the main board. These conclude feature requirements of the first component.

Second component includes real time communication which is crucial when controlling a flying object. Lilypad has a bluetooth modem, it can be connected to the main board. This step requires a reliable and frequent connection between Lilypad and computer or another platform. Then data from Lilypad could be transferred.

Third step is to process data from motion of the hand, so that it can be used to control helicopter. Different positions of the hand implies different helicopter moves. Data from the Lilypad are in the form of analog signals, they cannot be used until they are converted into digital signals. In this phase, we use a virtual helicopter to fly.

# 2. Hand Gesture Recognition

Our gesture recognition module is a result of many decisions taken along the way.

## Identifying hand motion classes

Our design is aimed for intuitive control so that it will be easier for users to learn how to control and have more fun. At the same time, the hand motions had to be distinguishable for successful recognition. Taking both of these criteria into consideration, we based our design on the following hand gestures:

| Hand motion | Helicopter action | Description |
|---|---|---|
|  |  | Ascend |
|  |  | Descend |
|  |  | Move backward |
|  |  | Move forward |
|  |  | Move left |

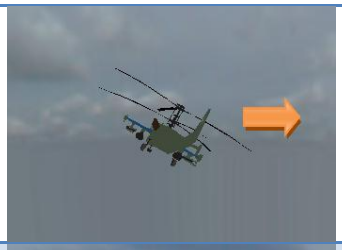
| | | |
|---|---|---|
|  |  | Move right |
|  |  | Idle position (No action) |

**Table 1** – Hand motion classes and their corresponding helicopter motion mappings

As depiced in Table 1, helicopter is oriented according to hand's orientation. We make it very intuitive and easy to control the helicopter. A typical run would consist of

- Stretch out your arm forward and make sure your palm is facing upwards.

- Raise your fingers (Helicopter starts to ascend)

- Flip your hand so that your palm is facing downwards

- Control the helicopter as if your hand itself it the helicopter

- Flip your hand so that your palm is facing backwards and lower your fingers to land the helicopter (Helicopter starts to descend)

Our trials showed that the most difficult hand motions are those for ascending and descending the helicopter and rotating it left. Since in a typical run, ascending and descending will occur less often than moving backward/forward, we assigned those difficult hand motions ascending and descending functions.

## Setting up and testing accelerometer

Initially we conducted test with accelerometer alone to determine its sensitivity and accuracy. We carried out several hours of data collection, with random users, to measure the accelerometer's sensitivity (how quick its output changes with sudden movements) and accuracy (how consistent its output is). Our accelerometer algorithm uses two parameters, as can be seen in the pseudocode below:

```
Begin function
        Record x, y, z values from accelerometer
```

```
        If total number of records == N
        Begin
                Output average of last N values recorded
                N ← 0
        End
        Sleep t milliseconds
End
```

After testing, we observed it is optimal (accurate and frequent enough) to use N = 100 and t = 1, which corresponds to outputting values every 200-300 milliseconds.

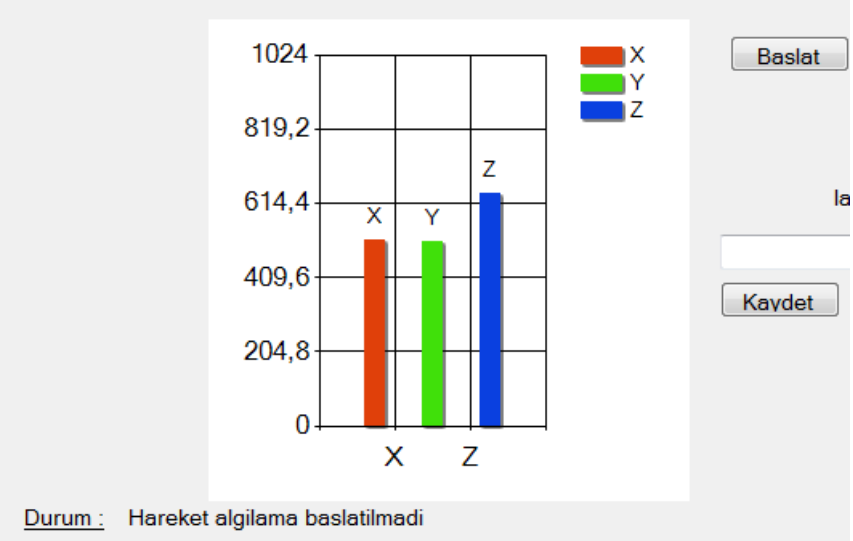



**Figure 1 –** A test application is developed to display, test and record accelerometer output values. Note that y=512 corresponds to 0g, 512-1024 spans +3g and 0-512 spans -3g linearly. Thus, in the given figure, x and y axes have almost zero gravitation pull whereas z axis has around +1g gravitational pull.

## Gesture recognition

Initially, we gathered training data for data analysis purposes (and calculation of parameters such as mean and variance). We tested the following main approaches (whose details will be explained later in this section).

**a.** Gaussian maximum likelihood estimator, assuming x-y-z **independent**, based on **absolute** values

**b.** Gaussian maximum likelihood estimator, assuming x-y-z **dependent**, based on **relative** values

**c.** Gaussian maximum likelihood estimator, assuming x-y-z **independent**, based on **relative** values

As mentioned earlier, we have identified 7 hand gesture classes. They will be referred as Position0, Position1, … ,Position6. Accelerometer data is time-based series of 3 dimensional numeric vectors (e.g. <400,450,500>) corresponding to acceleration values in x, y and z dimensions.

**a.** Gaussian maximum likelihood estimator, assuming x-y-z **independent**, based on **absolute** values

Initially, we used absolute training values (absolute values of acceleration for each hand motion) and treating x, y and z values independently, we tried a classifier that works as follows:

- Calculate Gaussian probability for each position's each dimension separately using mean and variance values (calculated from our training dataset).

| |
|---|
| P (Pos2 \| x) (where P: Gaussian probability , Pos2: 2nd position class) |

- For each dimension, normalize probabilities according to its dimension

| |
|---|
| P (Pos2 \| x) = P (Pos2 \| x) / [P (Pos1 \| x) + P (Pos2 \| x) + … + P (Pos6 \| x) ] |

- For each position, sum up its probabilities for each dimension

| |
|---|
| P (Pos2) = P (Pos2 \| x) + P (Pos2 \| y) + P (Pos2 \| z) |

- For each of an arbitrary number (W) of last iterations, find the positions with highest and second highest probabilities. Assign the position with highest probability a score equal to its probability minus the second highest probability.

| |
|---|
| Score = $P(Pos_{max}) - P(Pos_{second\_max})$ |

Algorithm can be summarized as follows:

| |
|---|
| Give each position a score of zero<br>for each last W iterations (where W is window size)<br>find max_probability<br>find second_max_probability<br>score ← max_probability – second_max_probability<br>add this score to position with max_probability |

```
end for each
if a position has score greater then a threshold
        select that position
else
        select no position
end
```

Although this worked better than what we had expected, it was not accurate enough. What's more, because it used absolute data, its performance varied significantly from one person to another. Thus, we decided to use relative data and for better precision, add covariance among x, y, z values into the equation.

**b.** Gaussian maximum likelihood estimator, assuming x-y-z **dependent**, based on **relative** values

Assuming dependence among x,y and z values, we used multidimensional Gaussian estimator in order to incorporate covariance into the model. Unlike part a, where we used absolute x, y, z acceleration values, we used relative values that measured deviations from the origin. In other words, we asked the user to stretch out his arm forward and hold his hand tight for calibration. We treated this position as origin and for the rest of the data, instead of using absolute x, y, z acceleration values; we measured differences in x, y, z and z acceleration values according to this origin.

The classifier worked as follows:

- Calculate Gaussian probability for each position using mean and covariance calculated from training dataset.
- Determine the highest and second highest probability values among all positions.
- If the highest probability exceeds the second highest probability by a specified threshold, select that position. Else, select no position.

For multidimensional Gaussian estimation, we tried the following several cases:

- Separate covariance matrices for each class, (so covariance among x,y and z are assumed different for each position class)

- A common covariance matrix for every class and (covariance among x,y and z are assumed same for every position class)

- A common covariance matrix with off diagonal entries removed. (no covariance is considered, only variance within x,y and z dimensions are considered)

Although this method was more complex than what we'd tried in part a, it performed very poorly so we decided not to use it.

**c.** Gaussian maximum likelihood estimator, assuming x-y-z **independent**, based on **relative** values

We applied Gaussian estimators to each dimension separately, using relative values (difference from calibrated origin). We made two major changes to the classifier that improved its performance:

1- We added idle position as a class. Idle position (arm is stretched out forward) is very close to most of the positions and thus, when user's arm is in idle position, it is easily misinterpreted as a different class. To overcome this problem, we added idle position as a class so when user's arm is in idle position, the classifier now identifies this class (although there is no action associated with it).
2- Some positions vary only in one dimension and thus, classifier may fail to distinguish between these two positions easily. For this reason, we added a mechanism that omits a position if any of its dimensions has very little probability associated. For instance, ascend and backward hand motions are very similar and they vary only in z dimension values. With this additional mechanism, although probabilities associated to its x, y, and z dimensions may be very high, either position is omitted because its probability based on z is incredibly low.

On top of these, we also applied thresholding at two phases (as explained below). Threshold values were calculated after testing with various values.The final algorithm works as follows:

- For each position's each dimension, calculate Gaussian probability based on mean and variance values from training dataset.

| P (Pos2 \| x) |
|---|

- If this calculated probability is lower than Threshold1, ignore this position regardless of its probability values on other dimensions

- If this calculated probability is lower than Threshold2, assume its probability to be zero (to avoid math of very small numbers)

- Normalize probabilities according to dimension

| P (Pos2 , x) = P (Pos2 \| x) / [ P (Pos1 \| x) + P (Pos2 \| x) + … + P (Pos6 \| x) ] |
|---|

- For each position, calculate its sum of probabilities for each dimension

| P (Pos2) = P (Pos2 \| x) + P (Pos2 \| y) + P (Pos2 \| z) |
|---|

Algorithm can be summarized as follows:

- For each of last W iterations (where W is window size)

    Calculate highest probability value

    Calculate second highest probability value

    If highest probability – second highest probability > Threshold3

        Select class (with highest probability) as winner of this window

- If for all windows, there is a single selected class, choose that class

- Else, return no selection

# 3. Simulation

We first used Panda3D graphics environment. It supports both Python and C++ programming languages; however, its online documentation is mainly targeted at Python. Thus, we found it difficult to use.

After Panda3D, we started using Irrlicht graphics environment. The Irrlicht Engine is an open source high performance real time 3D engine written and usable in C++ and also available for .NET languages. It is completely cross-platform, using D3D, OpenGL and its own software renderer, and has all of the state-of-the-art features which can be found in commercial 3D engines such as physical engine and 3d modelling.

During the initial steps we used keyboard controls to move object on the screen, and then we planned to connect motion detection to simulator. First we started moving and rotating an object. We placed a 3D hand model in the middle of the screen. Irrlicht allows us to behave models as nodes; hence we had a node called hand in the beginning. During the program execution we continuously check whether a key is pressed. If the key pressed is defined in our program, model on the screen moves or rotates in the direction defined.

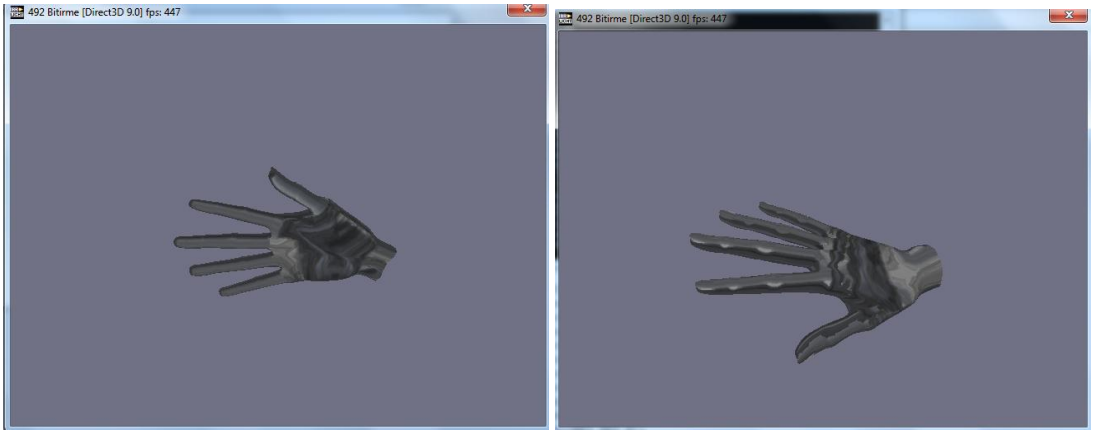



**Image 1 -** First 3D Hand Model

Second step includes placing a 3D helicopter on the screen in order to provide reality to the simulator. Criteria for a helicopter model was its visual beauty and format that is supported by Irrlicht. Then we included a terrain and a sky-dome (sky image for reality).

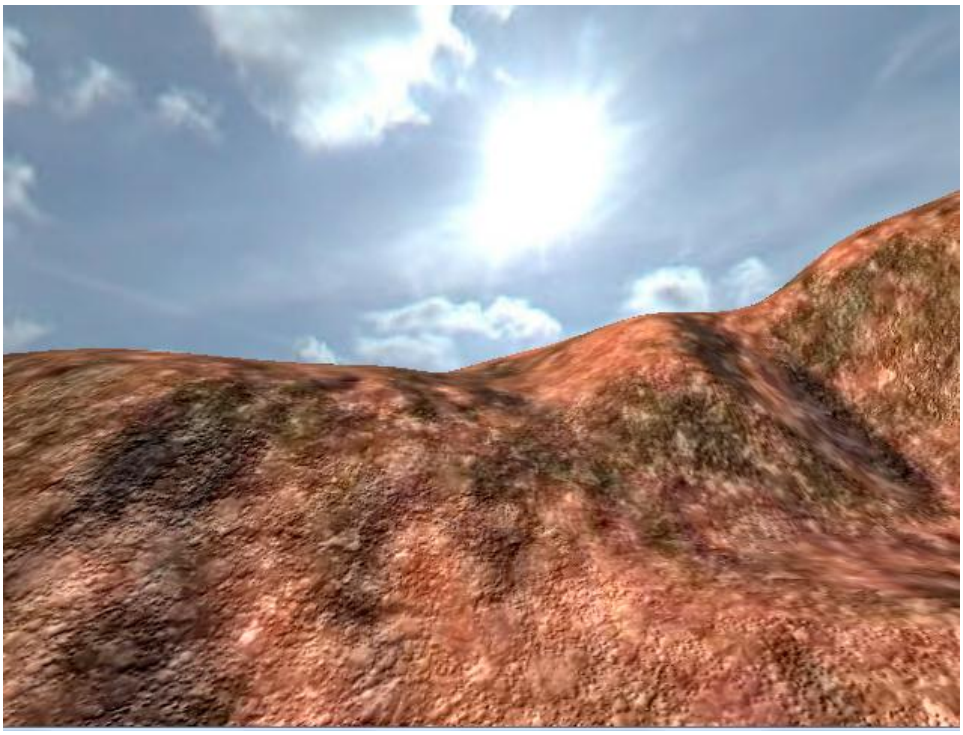


**Image 2 -** Terrain and sky-dome

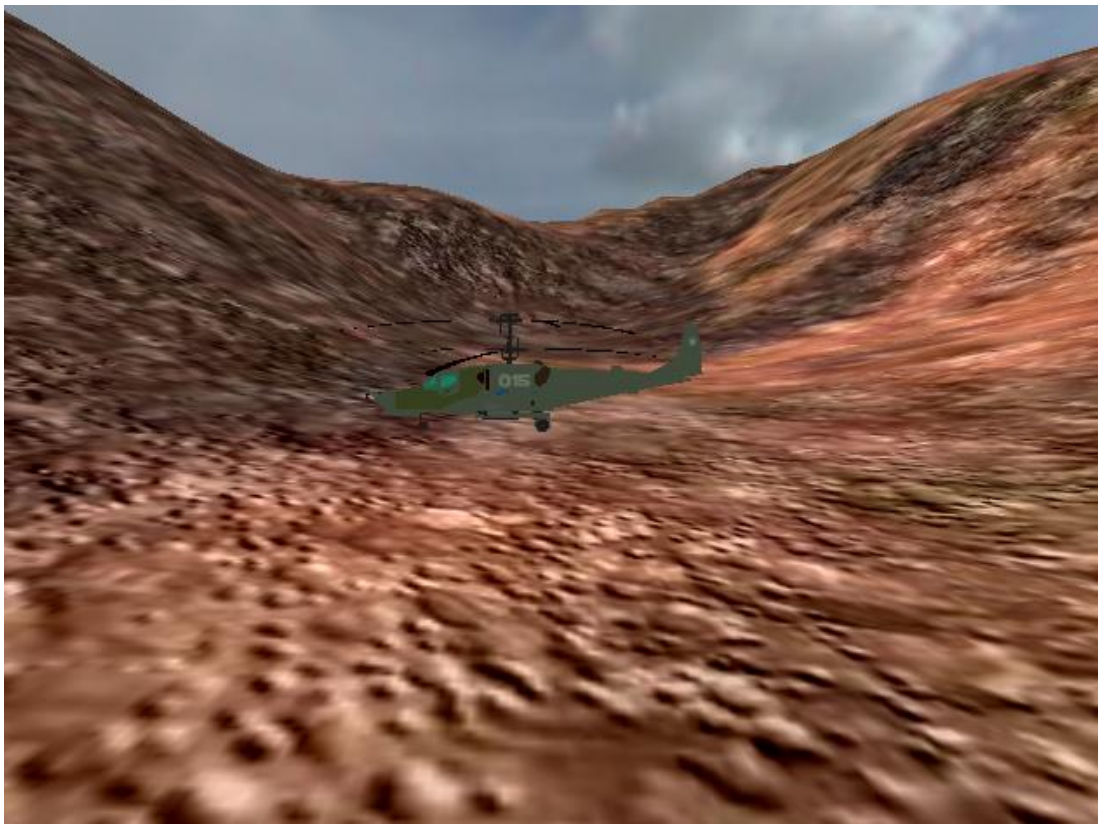


**Image 3 –** Helicopter on the ground

**Image 4 –** Helicopter flying in the air

**Image 5 –** 3D Helicopter Model

After placing all models and controlling helicopter with keyboard, it was time to connect simulator with motion detection. Motion detection program is written in C# and simulator is coded in C++. Therefore; we couldn't integrate codes. Instead we used a text file as a messenger between two programs.

# 4. Final Product

Our general approach was to create as many prototypes as possible along the way and to progress in steps, instead of trying to come up with a fully functional complete product at once. This way, we believe to have progress much more efficiently since we reduced number of uncertainties by introducing small number of changes and we were able to test each development by itself.

When both simulation and hand recognition modules were in working condition, we began work on communication among these two separate modules (written in two different programming languages, namely C# and C++). Instead of trying to merge these two modules into a single stand alone program, we decided to use an intermediate file for communication among the modules. After some testing, we came up with our final product that uses a single common file for communication (we tested parameters for communication such as frequency of writes and reads).

Design of the hardware plays an important role in the product's performance. Each of our prototypes that we have built along the way allowed us to identify bottlenecks and thus, with each prototype, we managed to improve our design. To be more specific, our first prototype (which was a sock) made us realize that our wearable product must wrap around user's hand as tight as possible so that the accelerometer on top would not relocate during execution. Our second prototype made us realize the importance of positioning of the accelerometer on the glove. It had the accelerometer placed on tip of the glove which lay right above the joint of the middle finger where it joined the hand. Since that joint moves during a typical run, it causes the accelerometer to change orientation and thus, creates a huge source of uncertainty for motion recognition. In our final prototype, we fixed the accelerometer and located it on top of the outer palm, to make sure that it doesn't get affected by fingers' movements.

# 5. Results and Conclusion

Our initial goal was to develop a wearable mobile device that allowed users to manage an application with motion. A glove that allows users to manage a helicopter (in simulator) with hand motions was our final decision among our project ideas. In this project, we reached our goal and successfully delivered a product that is a glove that allows its users to control a simulated helicopter via basic hand gestures.

During the whole progress, we tried to keep our product simple and robust as opposed to a complex product that failed occasionally. In other words, instead of trying to add more functionality and features, we tried to perfect the main features and make sure that the final outcome was robust and flexible.

Our final product achieves robustness and flexibility. It doesn't run into communication errors once initialized and it can be used by different people with ease, after a few seconds - typically 3-5 seconds - of calibration.

We initially assumed that connectivity, hand gesture detection and proper simulation would be our main software challenges. Surprisingly, connectivity (both Bluetooth and USB connections) worked without significant problems or delays. Hand gesture detection cause dtrouble but we managed to get it working and the final product delivers an enjoyable experience to any kind of user in this sense. Simulation and its inter communication with the hand gesture module also turned out to be less problematic than expected. As for hardware, we had the expectation that the limited processing power of LilyPad would be a bottleneck for hand motion recognition. However, with sufficient testing, we were able to optimize its communication with PC that did not exceed Lilypad's limited processing power and still provided accurate results.

On the other hand, some unexpected issues arose throughout the project. Firstly, we were not expecting that hand gestures would vary so greatly across different people. Due to different shapes of hands as well as different motion gestures, it was quite difficult to come up with a hand motion recognition module that worked for any typical user. Secondly, we didn't think designing the device would matter so much. When resistances of the wires used are taken into consideration, as well as the placement of the accelerometer on the device, design turned out to have a significant role in the outcome. Especially when the accelerometer lay

above a finger joint, it yielded inaccurate results due to movement of the joint. Our final design took all these issues into consideration.

In conclusion, we produced a wearable mobile device that is robust, flexible and simple. We delivered a product that achieves our goals and we hope to carry on working on it. Next section describes our intention for future work.

# 6. Future Work

The project has been developed step by step from the beginning. More work could be done in order to increase quality of the control and the graphics of the simulator.

Detection of the hand motions could be improved by collecting more samples from different people, although it already works with a high success rate. People have different hand shapes; hence glove may wrap differently from hand to hand. This is not a problem, because calibration is done at the beginning of the program execution.

On the simulator part, we may convert the program into a video game with a creative interface; glove. Now there is no collision detection implemented. So, helicopter may fly anywhere, even it can dig the ground. This version would be a training mode. With a collision detection and a story, an arcade mode can be coded easily. There are two ideas for the arcade mode:

1- Rings would be placed into air and user would try to fly into these rings. If the helicopter collides to ring, game is over. Number of rings would be the score.
2- Enemy flying objects would be put in the air, these object would fly to the helicopter, gamer should avoid these objects and have to finish a track to end game. When collision occurs, game is over.

Better 3D models may be obtained online and used. Helicopter model already used is not animated. An animated helicopter model may increase visual quality. Also better looking terrain and sky-dome models could be obtained or designed.

Different terrain and helicopter models could be used to increase variety of simulator. Each would be named as different level in the arcade mode.

Project could also be converted into a real radio controlled helicopter. To do this, output of the program that calculates hand motion should be sent to the helicopter with radio signals. This means a new component should be included in the system; a device attached to the computer that can establish wireless communication via radio waves.

We would also like to note that we didn’t have enough testing to determine battery usage of the device. Battery consumption is an important aspect of any mobile device and longer testing periods will allow us to evaluate battery requirements of the device.

# 7. References

1 - http://irrlicht.sourceforge.net/ - IRRLICHT 3D Engine website

2 - http://www.arduino.cc/ - Arduino mainboard website

3 - http://www.arduino.cc/en/Main/ArduinoBoardLilyPad - LilyPad Arduino overview

4 - http://web.media.mit.edu/~leah/LilyPad/ - LilyPad Arduino introduction by its designer Leah Buechley