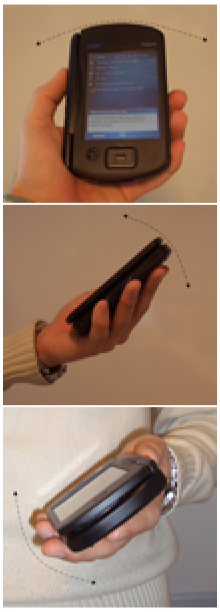
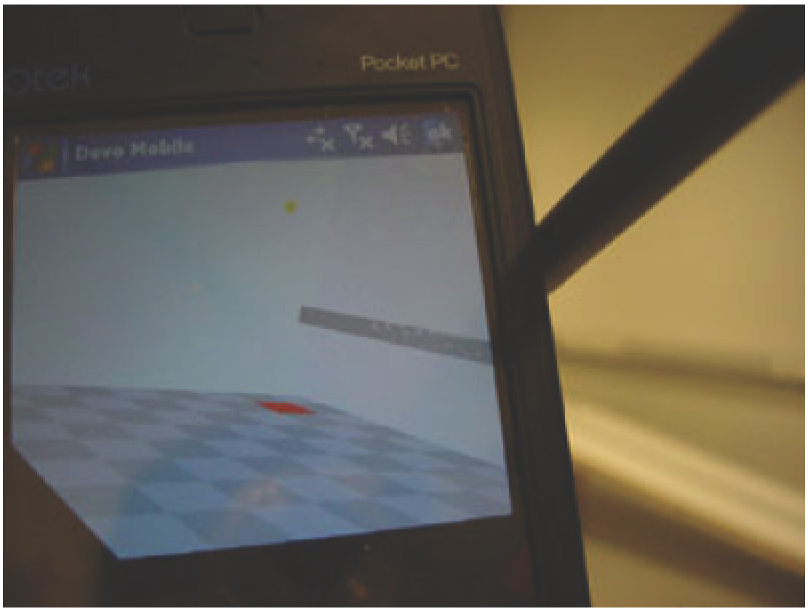
Device Orientation

Project Report

Location Aware Systems *(ITI45206)*

Forskning, skriving og publisering *(ITI40906)*





Sizarta Sarshar

sizarta.sarshar@hiof.no

June 2, 2006

Halden, Norway

Østfold University College

Mobile Applications Group

Abstract

Keywords: Mobile Applications, Device Orientation, Handheld Computers, Virtual Reality

In this project, device orientation and positioning is used to display a 3D virtual model on a handheld computer from the users point of view. This report describes how a prototype for the system has been developed and used for proof of concept. It describes the method and technical devices used.

The user experience of the system was intutitive and worked great, and the prototype did proof the concept.

Preface

This report has been written to meet the needs of the courses *Location Aware Systems* and *Forskning, skriving og publisering*. The project is, however, developed in the first cource. The courses are a part of the Master Education at Ostfold University College, Halden. I would like to thank Assistant Professor Glenn Ole Hellekjær and Associate Professor Rune Winther for their help during the writing process, and Associate Professor Gunnar Misund for lecture and tutoring. The supervisor for this project has been Associate Professor Rune Winther who also has provided technical support.

Prerequisites

In writing this report I have assumed that the reader know basic computer science and have common knowledge on handheld computer and virtual reality.

Chapter Breakdown

Chapter 1 (Introduction) provides a brief introduction to the project's research statement. Chapter 2 (Background) includes motivation and provides neccessary background information. It also illustrates different scenarios where the product might be used. Chapter 3 (Methodology) provides a brief introduction to the methodology of prototyping and how the method will be used in this project. The prototype's essential functionality with success criterias for the system will be explained. Chapter 4 (Design and Implementation) describes the design of the system. First, the technical device and hardware used are introduced. Secondly, the overall communication data flow, followed by the software applications for the server and the mobile device. Chapter 5 (Evaluation) will evaluate the prototype. The system will be tested for the success criteria described in Chapter 3. Possible solutions to occuring problems are discussed in Chapter 6 (Discussion, Conclusion and Future Work) with overall discussion of the project followed by a short conclusion and future work.

Similiar Project

There are other ongoing projects with use of handheld computers and device orientation. Glasgow University, Department of Computing Science, works on similar projects. The device orientation is measured using gyros and is used in several applications. Check out their Dynamics and Interaction workshop at:

`www.dcs.gla.ac.uk/~rod/DynamicsWorkshop2005.htm`

Table of Contents

Chapter 1

Introduction

Handheld devices are a part of everyday life and include more and more functionality. New devices offer services such as web browsing, email management, and document creation. In addition they feature multimedia functionality such as music players and image and video capture and editing. However, these devices often have limited input and output capabilities. Limited screen space means displays can easily become cluttered. Input is also limited; slow and cumbersome methods such as small keyboards or inaccurate handwriting recognition are common.

The innovative aspect of this project is to explore a new paradigm for interacting with mobile computers, based on three dimensional device orientation and positioning. I will use a three axis gyro attached to a handheld computer and a positioning system, using the computer's screen as output and hand and device gestures for input. This will allow me to investigate possibilities for using mobile devices in a range of new ways.

The purpose of this project is to develop a prototype for viewing 3D virtual models on a handheld device, using an attached gyro and positioning system. With a gyro attached to the handheld device, the view angle can be controlled by the user's hand gestures. Likewise, the position of the point of view in the virtual model is moved around using a positioning system attached to the handheld device. These features attached to the handheld device will give the user the opportunity to control the view angle and the position displayed on the device screen by moving around and pointing the device in desired direction. I will develop a prototype for evaluating this system. Due to time limitations, the prototype in this project will be developed for proof of concept mainly, and not for design or usability testing. This approach will focus on the problems and their possible solutions with the prototype and its concept.

The following chapters will provide motivation and background information, methodology and how the prototype will be developed. The prototype will then be evaluated and discussed.

Chapter 2

Background

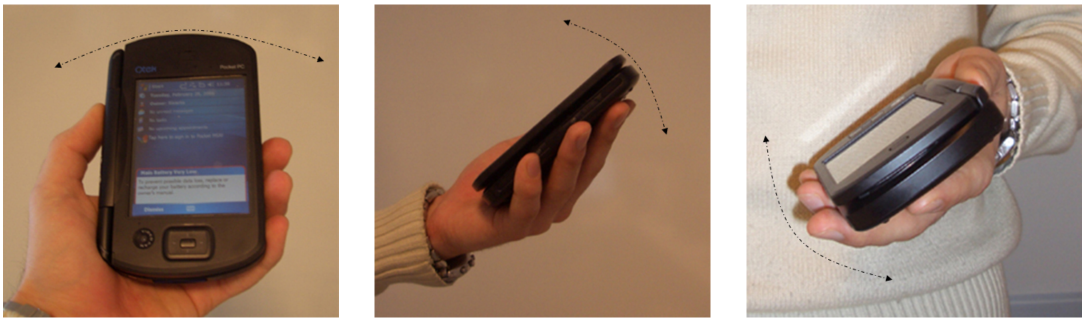
This chapter includes motivation and provides neccessary background information. It also illustrates different scenarios where the product might be used.

2.1 Motivation

The motivation for this project is based on the work of Institute for Energy Technology (IFE), which is the research institute for energy and nuclear technology in Norway [1]. Their research on the Visualization Technologies Supporting Design, Planning, Operation and Training area is built upon virtual and wearable technologies [2]. One area is to develop virtual reality techniques to support early human factors design input in control room design projects. Another topic is to improve the projects radiation visualization methods to visualize differebt varying radiation levels in 3D space. My project is motivated by their new interesting topic; the visualization of risk information in a 3D environment.

2.2 Scenario

The scenario I envision is walking in a real room, but where a virtual model of the room is displayed on the handheld computer. Many applications running on stationary computers provide 3D virtual scenery where the user can move and view different objects using the computer mouse and keyboard to navigate. First person games are played from the first person's point of view, and the user uses the mouse to control view angle and the keyboard to move camera position around the person. While handheld computers have small or even no keyboard and no mouse, they have the benefit of being handheld. Therefor, connecting a three axis gyro to the device will give the device orientation, i.e. yaw, tilt and roll (Figure 2.1). Combined with the position, the user can walk in a room with a handheld computer, while the virtual model of the room will be displayed on the device screen and the view angle can be controlled by the user's hand and device gesture. This approach gives the user valuable information of the present room from a virtual model.



(a) Yaw (b) Tilt (c) Roll

Figure 2.1: Device orientation

For instance, a room full of smoke can be quite difficult to navigate through. With a handheld computer a virtual representation of the room can be shown for the user. The user can simply point the device in the desired direction to see the virtual representation of that area. If no obstacles are in the way the user may easily navigate through the smoke.

Another example is in nuclear areas where radiation is a major hazard, and where different sensors placed in the area measure radiation. This information is valuable for a worker who works where radiation is present. The radiation level measured by the sensors can be plotted in a virtual model of the area, and can help the worker to avoid areas with high radiation. Thus, pointing the device in a desired direction simply gives the worker an indication of the radiation levels and a warning to leave if the level is too high.

Both scenarios can be expanded with new features as guiding the fireman through the smoke in scenario one and guiding the worker in scenario two along the path with least radiation.

2.3 Chapter Summary

In sum, this chapter includes the motivation for the project and describes two different scenarios where the system might be used. The motivation is based on the work of IFE and their topic; the visualization of risk information in a 3D environment. The scenarios described illustrate different usage of the system; in rooms with no or little view and in different areas where additional information can be provided on a model.

Chapter 3

Methodology

This chapter provides a brief introduction to the methodology of prototyping and how the method will be used in this project. The prototype's essential functionality with success criterias for the system will be explained.

3.1 Prototyping

A prototype is an initial version of a system which is used to demonstrate concepts, try out designs and, generally, to find out more about the problem and its possible solutions [3]. It is not a final product, it is a step on the iterative way to a final product. In this project, I will use prototyping as method and develop one for proof of concept. It is in my interest to clarify whether it is possible to use a gyro and indoor positioning to achieve a usable prototype based on the project description, and whether the response time and accuracy for the system will be good enough. These, and other criterias which must be fulfilled for the system to be useful, are defined in Section 3.3. The design aspects of the project is not addressed in this first prototype of the system. One of the reasons is that use of a smaller gyro with another interface can reduce the size and hence change and affect the design.

Besides hardware connectivity, building this prototype also includes software to connect and use the required hardware. In addition, the prototype will give me an opportunity to learn more about the system, to find problems and difficulties, test and evaluate the system, find possible solutions and get ideas for future work. The prototype makes use of a handheld device, an orientation device and a positioning system. It is required that the handheld device runs Windows Mobile 5 and support Bluetooth technology. The orientation device can be a gyro and the positioning system must give position in three dimensions and have centimeter accuracy.

3.2 An Iterative Process

Prototyping is a method in iterative processes and there are several varieties of prototypes: paper, evolutionary and executable prototype. Choosing the proper one depends on the type of risk that is likely to exist in the system. The paper prototype (throw-away) is used to develop a mock-up of the system and to do some system experiments. The evolutionary is used when a person simulates the response of the system based on a user's input. The automated prototype (operational) involves the use of a rapid development environment to develop an executable prototype [4]. The prototype used in this project is an operational one were the coding is done *quick and dirty* to realize computer execution without to much time or effort taken.

3.3 Required System Functionality

The system functionality describes the main features and functionality for the system. It will also set the guideline for how the prototype should be designed and developed. The handheld device must display a 3D model according to its position and orientation. To do so, the device has to:

- download a 3D model for display, from a server
- establish connection with the gyro and retrieve orientation data
- establish connection with a positioning system and retrieve its current position
- display the model according to the device position and orientation
- allow user to interact with the loaded 3D model by offering services related to the selected object's properties

To be able to download models, a server is needed. A server can also be used to communicate with the handheld computer. The software prototype for the server's main functionality is to:

- establish connection with the handheld device
- upload a 3D model

- display the uploaded 3D model from a point of view set by the position and orientation of the handheld device
- allow user to change point of view to a predefined position and orientation giving an overview of the model
- allow user to interact with the loaded 3D model by offering services related to the selected object's properties

3.4 Success Criteria

The success criteria for the system are set to determine if the achived concept is as required. The prototype must fulfill these success criterias before further work is worth while. Having mentioned the importance of these criteria, let us now describe them. I will devide the criterias in five groups: positioning, orientation, virtual modeling, interaction and usability.

- **Positioning:** The estimated position must be within 20 cm of the actual position and must be updated at least once per second.
- **Orientation:** The orientation accuracy must be within 10 degrees without drift and be updated several times per second.
- **Virtual Modeling:** The virtual model must be exactly the same on both the mobile device and the server. The room size must be in actual size and the room must maintain it's characteristics in the virtual model.
- **Interaction:** Success criterias in this case is that change in properties for an object must occur to the correct object.
- **Usability:** The user experience of the system has to be good and intuitive. The user must not experience significant time delays with the system.

3.5 Chapter Summary

This chapter has provided information about the methodology for prototyping and described how it is used in this project. The system functionality and system criteria for the prototype are also described. These system functionality and criteria are used in the Design and Implementation (Chapter 4) and Evaluation (Chapter 5) chapter later on.

Chapter 4

Design and Implementation

The scenario with radiation level in a nuclear facility and the scenario from a room full of smoke (from Section 2.2) are examples that are difficult for me to implement and develop. Instead, I will create a virtual version of the robot laboratory room at Ostfold University College [5], in which the indoor positioning system is installed.

In the following, hardware and software design will be described.

4.1 Hardware

This section describes the hardware equipment used to develop the prototype.

4.1.1 Technical Devices

The handheld computer used in this project is a Qtek9000 (Figure 4.1), combined with a gyro to get the orientation. Bluetooth[1] technology provides wireless communication between the gyro and the handheld computer. The orientation is however not enough; I will need to know the device position as well. Since Global Positioning System[2] has 2-100 meters accuracy, which is far too high for this project, it would be almost impossible to get a correct height. I will therefore use an Indoor Positioning System[3] (IPS) developed by Sonitor Technologies [6] which is already installed in the robot laboratory at Ostfold University College. The IPS has an accuracy of 3-5 centimeters, but

[1]Short-range radio links in the 2.4GHz ISM
[2]Satellite based navigation system on a worldwide basis
[3]The Sonitor indoor positioning system uses ultrasound to locate the source.

covers only an area of 5x5x5 meters. The benefit of good accuracy is however more important than the limitation of space.

Figure 4.1: Handheld computer Qtek9000

4.1.2 Gyro and Device Orientation

The three axis gyro used in this project is a 3DM-GX1 developed by MicroStrain [7], and is used to get the device orientation. It has a serial communication port interface, and is connected to the handheld computer using Handyport HPS-120 [8], which is a wireless Bluetooth RS-232[4] transceiver (Figure 4.2). The connections are illustrated in Figure 4.3.

The Bluetooth tranceiver must be configured to communicate with the handheld computer before usage. The configuration is described in Appendix A.

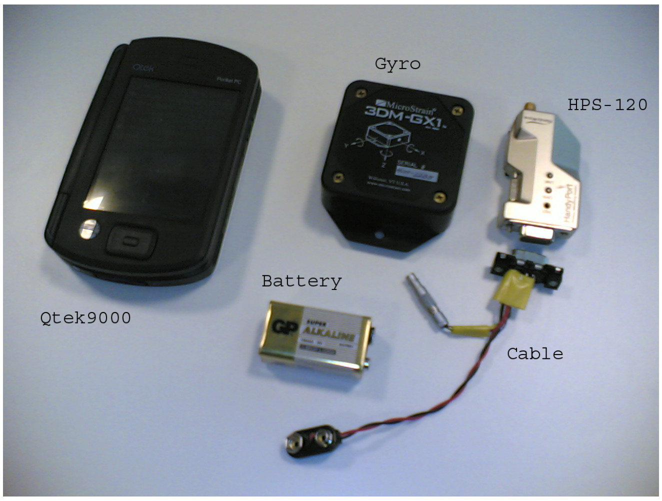
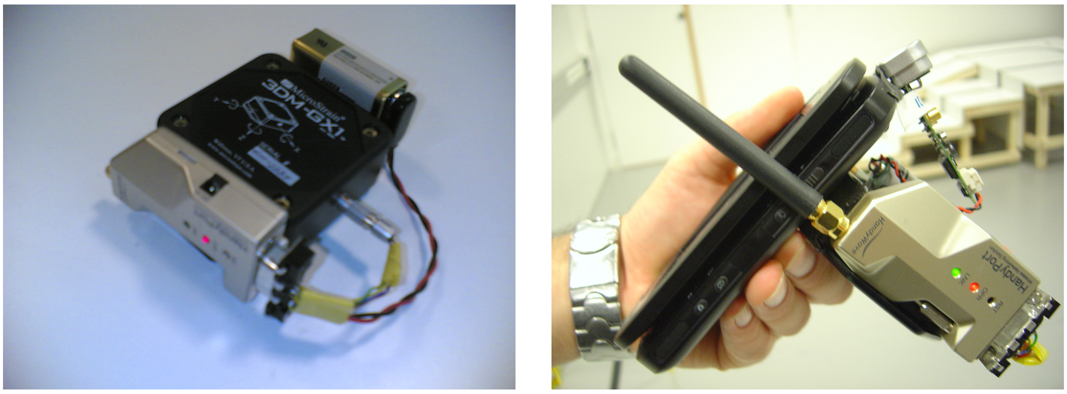


Figure 4.2: Required equipment for gyro connection to handheld computer

[4]An asynchronous serial data interchange standard.



(a) Gyro with battery and Bluetooth (b) Gyro connected to the handheld computer

Figure 4.3: Physical connections to the handheld computer

The gyro incorporates three accelerometer sensors to measure earths gravity, three magnetometer sensors to measure magnetic fields and three rate gyroscope sensors to measure the rate of rotation about their sensitive axis. It is a self-contained sensor system that measures the three degrees of its orientation in space with respect to earth[5]. It defines a coordinate system that is *fixed* to the Earth with the Z-axis pointing down through the center of the earth, the X-axis pointing North and the Y-axis pointing East as illsutrated in Figure 4.4. By *fixed* means that this coordinate system is stationary and provides a reference to measure against [9].

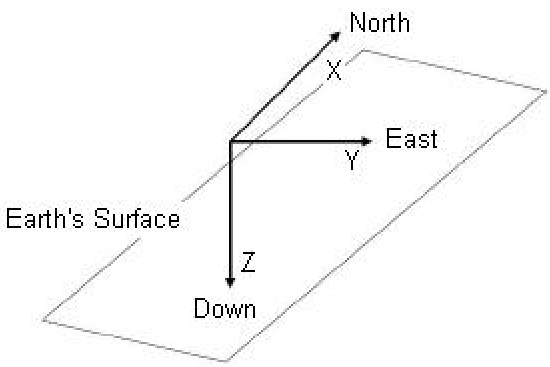


Figure 4.4: Earth's coordinate system

[5]When we say Earth, we are referring to the coordinate system established by the cardinal axes of our planet earth itself

4.1.3 Indoor Positioning System

The indoor positioning system installed in the robot laboratory of Ostfold University College, uses ultrasound. The system use an ultrasound transceiver tag (Figure 4.5), and eight sensors in the room to locate the tag. The tag sends short messages with time stamp three to four times per second. The sensors are placed strategically to cover the entire room area (Figure 4.6). They detect the message sent by the tag and use the time from the message was sent until it was received by each sensor to calculate the tag's position. The position is then sent to a server computer which forwards it to the handheld computer. The tag with battery will be attached to the handheld computer.

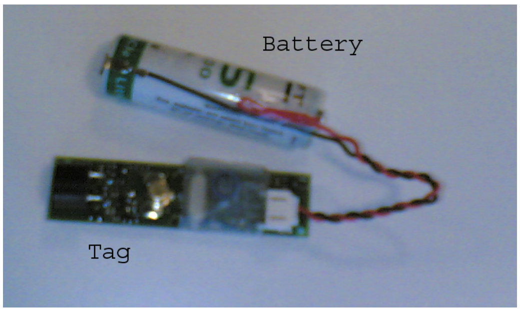


Figure 4.5: Ultrasound tranceiver tag

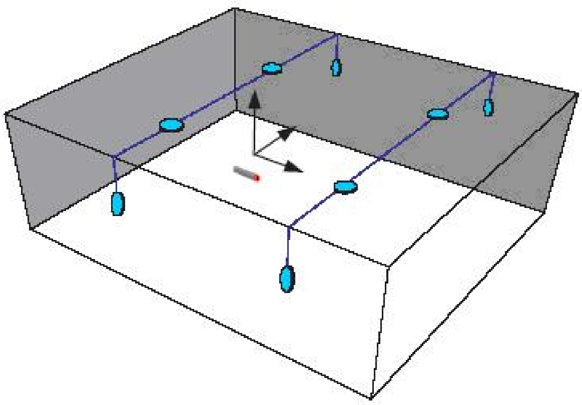


Figure 4.6: IPS sensors in the robot laboratory, with the tag in the center

4.1.4 Overall Device Communication

The overall device communication for the system is shown in Figure 4.7. The gyro and the server communicate with the handheld computer using Bluetooth technology. The server does not have

Bluetooth suppurt, and I will therefor use a Handyport HPS-120 which is a Bluetooth tranceiver. The Handyport is connected to the serial port of the server. Bluetooth is a common communication technology on mobile devices, so the handheld device can easily be replaced by newer or better ones on newer prototypes.

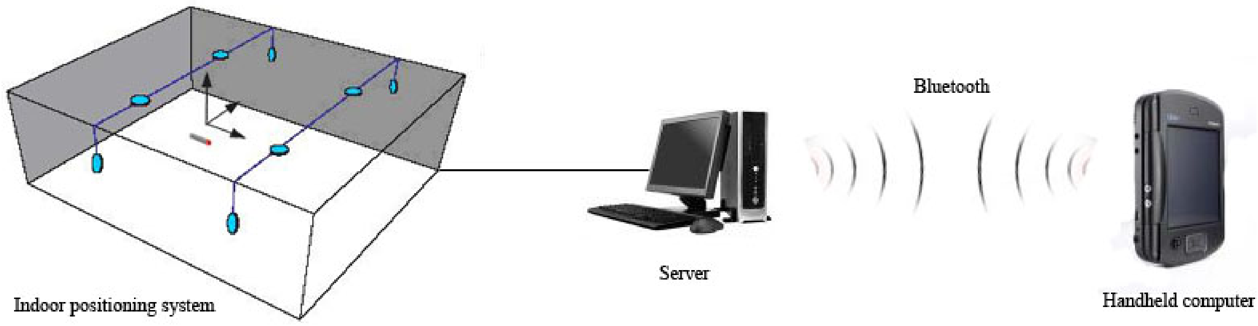


Figure 4.7: Overall device communication for the system

4.2 Software

This section describes how the software is designed and created for both the handheld device and the server. The applications are written in Microsoft Visual Studio .Net [10] using the C-Sharp programming language. The generated code for the server application is included in Appendix B and the code for the mobile device applications is included in Appendix C.

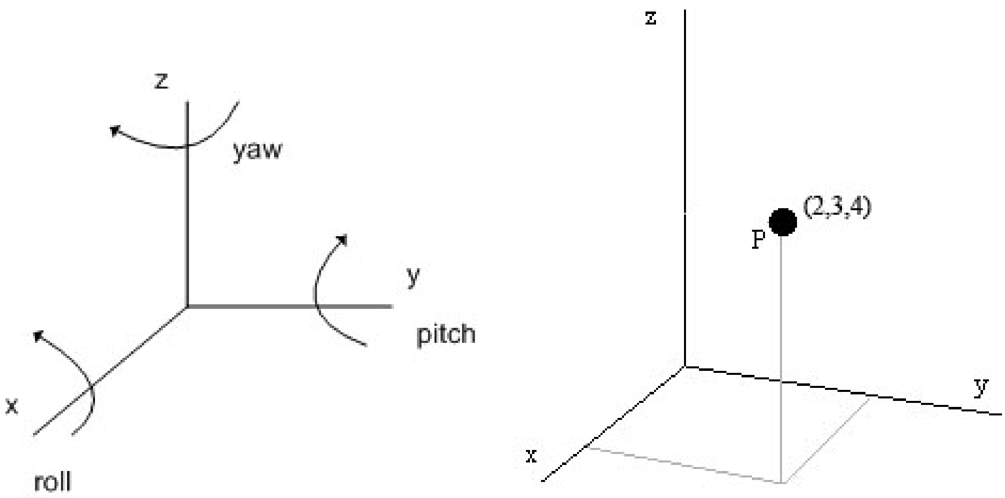
4.2.1 Visualizing in Three Dimensions

The handheld computer Qtek9000 runs Windows Mobile 5 (WM5) operative system. The WM5 SDK[6] features Direct 3D Mobile for visualization of three dimensional models. To reduce time and effort, the virtual model of the room is hard coded into the application in the first prototype.

When the virtual model of the room is rendered, the point of view angle and position must be set before the model is displayed on the mobile device. The position is by default set to be in the center of the room and the view angle is by default set to view straight ahead at the wall on the XZ-plane. When the gyro pitch, tilt and yaw are updated from the gyro, the view angle is transformed using axis rotation (Figure 4.8a). The first rotation must be a rotation around the Z-axis followed by rotations around the X- and Y-axis . Changing the order of the axis rotations will result

[6]Software Development Kit

in an inappropriate behavior of the point of view orientation angle. The position of point of view is transformed by matrix multiplication with the X, Y and Z coordinates given by the positioning system. Figure 4.8b illustrates a translation to the point P with X, Y and Z equal to 2, 3 and 4. The translation is done ahead of the axis rotation. The result is that the gyro controls the point of view angle and that the positioning system moves the point of view in the virtual model.



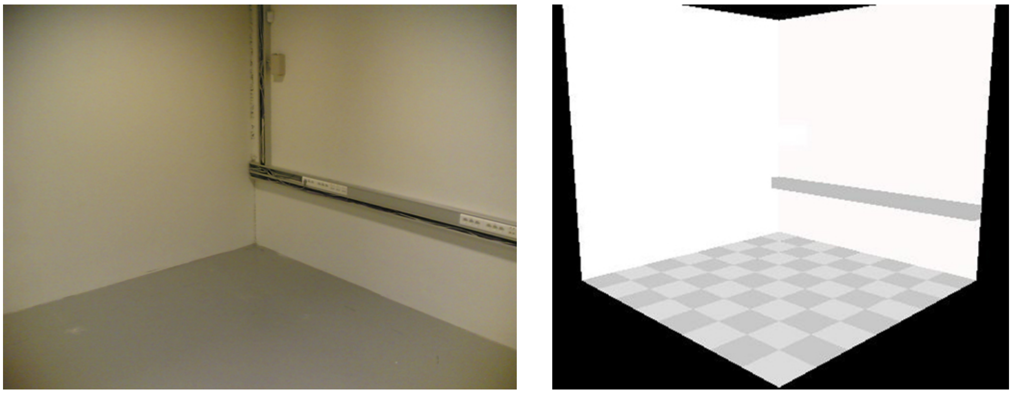
(a) Axis rotation (b) Translation

Figure 4.8: Point of view translation and rotation

The indoor positioning system has an accuracy of five centimeters, but it is unstable. Sudden change in coordinates with more than 50 cm, when we are actually standing on the spot, can cause problems in the visualization. The rendered image on the screen may jump and be difficult to follow for the user. A simple way to stabilize the positions is to use the last five known positions in each direction and calculate the mean position. We can also add in a feature to ignore position changes beyond what is natural (50 cm per second). This is done by the server application before the calculated position is used and sent forward to the handheld device.

4.2.2 The Virtual Model

A virtual model represents the physical world. The model used in this prototype represents only a small area of the robot laboratory. A small model is chosen since the positioning system only covers this area of the room. The model contains a floor area of 4x4 meters and two walls which make a corner. One of the walls have cabinets for electric wireing. Besides this, the model is low on details. The colors of the virtual model are almost identical to the real colors of the room. Figure 4.9a shows the area of IPS coverage and Figure 4.9b illustrates the virtual model of the same area. A minor change has been made to the virtual model though; the floor is devided into squares of 50x50 centimeters. The squares differ in shade of gray so they can be seperated from each other, like a chess board. This feature makes it easier for the user to follow the model when only the floor is displayed. The squares are also separated objects. Beeing separated objects makes them available to the user for interaction and selection.



(a) The robot laboratory (b) The virtual model

Figure 4.9: The robot laboratory with the virtual model of the same area

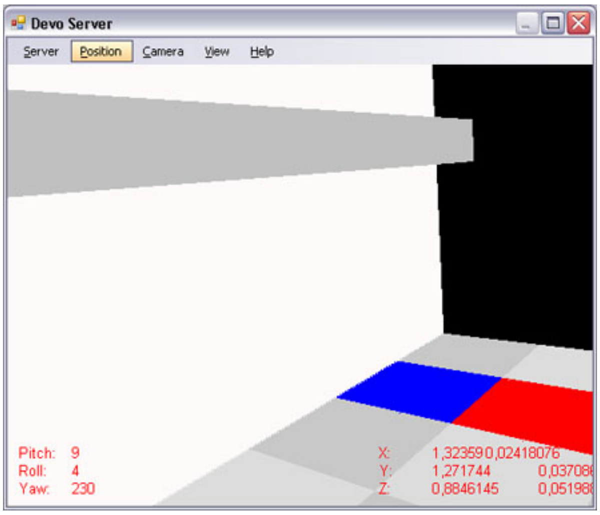
4.2.3 Interaction

The interaction is implemented so the user on the server can select an object in the virtual room which will be shown to the client on the handheld device. This interaction with the model is also available from the clients handheld device. The client can at all time select an object by simply touching the screen on the handheld device. When an object is selected, it changes its color. If it is selected by the server, it turns blue and if it is selected by the client, it turns red. The users can only select one item each at once. An illustration is given in Figure 4.10.

4.2.4 The Server Application

The server application has mainly three functions. The most important one is to retrieve tag position from the indoor positioning system (IPS) and forward it to the handheld device. Secondly, the server displays, in the virtual model, what the user is looking at. To do so, the server retrieves gyro data

(a) Interaction with the handheld device



(b) Interaction from server

Figure 4.10: Interaction

from the clients handheld device. Thirdly, the server can interact with the client. The application consist of mainly one cycle. The cycle handles positioning and rendering and include the following:

1. Retrieve position from the positioning system
2. Store the position in an array if the uncertainty is beneath 10 percent
3. Calculate the mean position from the array containing the last five coordinates
4. Send the position to the handheld device
5. Perform translation of point of view in the virtual model
6. Perform axis rotation of point of view
7. Apply change of color on selected objects
8. Render the image for display

In addition, the server consists of two event handlers. One event occures when data is retrieved from the handheld device and includes:

1. Retrieve the data from the handheld device

2. Update pitch, yaw and roll angles if the retrieved data contains gyro data

3. Update selected object ID for the client if retrieved data contains object ID

The other event occures when the user selects an object with a mouse click. It forwards the selected object ID to the handheld device.

It is expected that the positioning system is operative and is connected to the server, at all time. The server connection to the mobile device must also be operative during runtime.

The main functions for the server application is to send updated position, retrieve gyro data and display the gathered information in a virtual model.

4.2.5 The Mobile Application

The application running on the mobile device has mainly four functions. One is to retrieve gyro data, update local variables and forward the data to the server. The second is to retrieve the tag coordinates from the server. Third, it has to render and draw the virtual model on the device screen. And finally to interact with the server with the feature object selections.

The application consist of one main cycle and two event handlers. The main cycle includes the following:

1. Poll gyro data from the gyro

2. Send gyro data to server

3. Perform translation of point of view in the virtual model

4. Perform axis rotation of point of view

5. Apply change of color on selected objects

6. Render the image for display

The first event retrieves position from the server and updates the point of view coordinates for the virtual model. The second event handles user interaction. It occures when the user touch the device screen and select an object in the model. The object ID is then sent to the server.

4.2.6 Dataflow

The dataflow for the system is illustrated in Figure 4.11.The server and the mobile device communicate with each other several times per second. The dataflow contains gyro data, position and object selection. The gyro data is updated constantly in the mobile application, and is forwarded to the server on the run. The server retrieves position from the indoor positiong system which is filteret before it is sent to the mobile device. Besides sending and retrieving gyro and position data, both the server and the mobile application send and retrieve information about which object is selected.

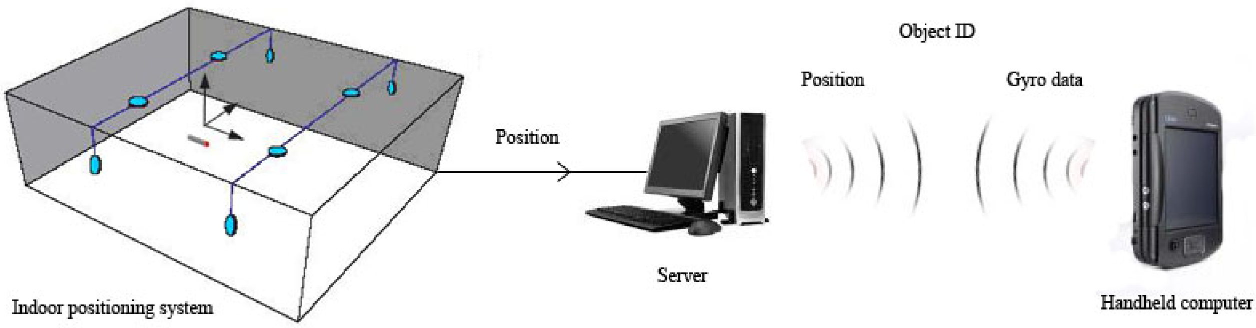


Figure 4.11: Overall dataflow for the system

4.3 Chapter Summary

The hardwares required to develop the prototype is a handheld computer, a gyro and a positioning system. They need to communicate with each other which is done by using Bluetooth technology. I have also developed software for both the server and the handheld computer. Next, I will test and evaluate the system.

Chapter 5

Evaluation

This chapter will evaluate the prototype. The system will be tested for the success criteria described in Section 3.3.

5.1 Testing for the Success Criterias

The system has been tested by simply using the system in the robot laboratory. It was set up and the handheld device was moved around and orientated in different directions. The test results for the success criteria are described in Table 5.1.

Some of the observed problems were a result of the algorithms used in the applications and some were caused by the technical device. The time delays are examples of a combination of both poor software algorithm and unstable hardware. Possible solutions to the problems are discussed in Chapter 6. The prototype limitiations of the hardware and software are described in the following sections.

5.2 Prototype Limitations

The positioning system has its limitations. Using the fastest tag, it updates the position three to four times per second with an accuracy within 10 centimeters. The accuracy gets better if the tag is hold steady and is pointing straight up. However, when it is connected to the handheld computer, it will not be pointing straight up all the time and it is unlikely that the user holds the device steady over a longer period. During movement, the accuracy is worse. The positioning system is also vulnerable for any metallic objects in the area. Metallic items reflect the ultrasound and interfere with the

GROUP	SUCCESS CRITERIA	TEST RESULTS
Positioning	The estimated position must be within 20 cm of the actual position and must be updated at least once per second	The position in the virtual environment did not refresh according to our real position, it was delayed with a few seconds and the accuracy was within 20-100 cm depending on where in the room we stood
Orientation	The orientation accuracy must be within 10 degrees without drift and be updated several times per second	The orientation worked great, but had some drift problems
Virtual Modeling	The virtual model must be exactly the same on both the mobile device and the server. The room size must be in actual size and the room must maintain it's characteristics in the virtual model	The model was simple, but it maintained some of the characteristics from the real room. The size of the room and the object on one of the walls seemed to be correct
Interaction	Change in properties for an object must occur to the correct object	The interaction with the model was a good idea and a useful feature, but the model was too simple to test it in a realistic setting
Usability	The user experience of the system has to be good and intuitive. The user must not experience significant time delays with the system	Usability problems occurred as result of poor accuracy in the positioning system. The position was also updated slowly

Table 5.1: Test results for the success criteria

system.

Having mentioned some of the limitations for the positioning system, let us take a look at the gyro and the device orientation. The gyro has an on-board processor which continuously executes a calculation cycle. The steps in this cycle include the following [11]:

1. Convert raw sensor outputs into digital form
2. Scale sensor outputs into physical units (including temperature, alignment, and G-sensitivity compensation). This provides the Instantaneous Vector quantities.
3. Compute the Gyro-Stabilized Vector quantities using the complementary filtering algorithm.
4. If host has issued a command byte, compute appropriate response data and transmit.

Step 4 in this cycle is only executed if the gyro has received a command from the host, this is done ten times per second by the application running on the handheld device before the virtual model is rendered. The calculation cycle continuously repeats itself. The time required to complete a calculation cycle determines the fundamental limit on the maximum data output rate. Tuning this and other settings can result in faster respons and more stabilized angle values from the gyro.

The gyro can output different types of data. The one used in this project is the set of gyro-stabilized Euler angles (pitch, roll, and yaw) which describe the orientation of the gyro with respect to the fixed earth. These angles are calculated according to the *ZYX* or *Aircraft* coordinate system. The quantities are gyro-stabilized and provide an accurate estimate of orientation even if the gyro is exposed to transient linear accelerations, or magnetic interference. The yaw angle do however drift.

An alternative output from the gyro is the set of Euler angles which do not incorporate any gyroscopic stabilization. In this mode artifacts will be present if the gyro is exposed to linear accelerations, or magnetic interference. None of the angles drift using this mode. The stabilization done by the gyro using the gyro-stabilized Euler data is not working properly since it drift. Instead the unstabilized angles can be stabilized in software. This will remove the drift problem, but will complicate the software a little bit. Due to time limitations, the method has not been implemented in this prototype.

5.3 The Virtual Environment

The virtual model and environment in this prototype is quite simple and were made for demonstration purposes mainly. It is therefor not good enough for evaluating the model. It is difficult to say anything about how small items or objects may appear on the device screen and how close we have to be to select one. Nor can it be said anything about how advanced models the device can render. Can the device handle a complete model of the room and how detailed can it be? These questions can be answered by reviewing the device specification and build a more detailed model for testing.

5.4 Usability

The user experience of the prototype was great. Every one who tried the prototype were surprised of how well it actually worked. Though the design was clumsy and not all success criteria were fulfilled, the prototype functioned well. You could actually move around in the room and point the device in desired direction to view a virtual visualization on the device screen. The interaction with the model did also work, both on the mobile device and on the server. All in all, the prototype proof the concept and the user enjoyed using it.

5.5 Chapter Summary

This chapter has provided the prototype evalutation. The prototype worked, but it has to be modified for further usage. Other software algorithms can be considered, and time delays must be avoided if they are caused in software. The technical devices also have their limitations. Usability problems occurred as a result of poor accuracy in the positioning system.

Not all success criteria were fulfilled. The orientation had drift problems, the update time for position was long and the virtual model was too small to be fully evaluated. However, these problems can be solved in a new and better prototype. In total, the system works and proof the concept.

The proof of concept is based on the user experience. Though there were time delays on position update, this was accepted. The user will not mind waiting a few seconds for update after movement. Despite some problems, the user experience of the mobile device was good and the user enjoyed using the prototype.

Chapter 6

Discussion, Conclusion and Future Work

This chapter will discuss both the project and the prototype. The solution chosen for the prototype will be discussed with other possible solutions. After the discussion part, I will summarize the work done before suggesting future work.

6.1 Discussion

This section will discuss the different decisions made in this project and solutions applied to the prototype.

6.1.1 The Virtual Model

The user experience of the system has to be good and intuitive. It is therefor important to render an image on the mobile screen that is easy to follow and easy to understand. There must be a correct representation of the real room in the virtual model, so users can easily recognize where they are and what they are looking at. Cohesion between the real world and the virtual world is essential.

A virtual model can give additional information to the real model. With the interaction between the user with the handheld device and the user on the server, it is possible to select objects and to change their properties. This feature allows the user in the field to ask for support or additional information on selected objects. The server application will immidiatly know which object the client has selected and can then upload information and requested data to the client.

The model used in this prototype is very simple and low on details. It only consists of a small floor area and two walls. Use of a larger and more detailed model of the room and the objects in it, would give a more realistic setting and a better user experience.

To reduce time and effort, the virtual model of the room was hard coded into the application in this first prototype. In later versions, it is recommended to download the 3D models and objects to the handheld device from the server. This way, the mobile device will have the same and the latest version of the models as the one running on the server. However, including and modifying virtual models, with Direct 3D Mobile, during runtime has not been tested in this project, but it is likely that it can be done, since it is possible in Direct 3D.

6.1.2 Mixed Reality

Augmented or mixed reality (AR) allow to mix or overlap computer generated 2D or 3D virtual objects on images of the real world. Unlike virtual reality that replaces the physical world, AR enhances the physical reality by integrating images ogf the physical world in to the virtual model which become in a sense an equal part of our natural environment [12]. This feature requires a camera to capture the real world. Due to time limits and difficulties with capturing live stream from the handheld device's camera, augmented or mixed reality was not an option in this prototype.

6.1.3 The Positioning System

For indoor use, the system requires an indoor positioning system. Sonitor's system based on ultrasound is unstable and is far from accurate. It must either be stabilized by a better algorithme than the ones used in this prototype, or alternative positioning systems can be used instead, providing a better accuracy.

6.1.4 The Orientation

The orientation measured by the gyro has drift problems. The problem occures occasionally and only affect the yaw angle. This happens when retrieving the gyro-stabilized Euler angles from the gyro. The Euler angles which do not incorporate any gyroscopic stabilization do not drift. It seems as if the drift is caused by the gyro while stabilizing the angles. By avoiding the gyro-stabilized Euler angles and instead do the stabilization in software, the drift problem can be removed.

6.2 Conclusion

The purpose of this project was to develop a prototype for viewing 3D virtual models on a handheld device, using an attached gyro and positioning system to control the point of view in the model. The prototype was developed for proof of concept.

A working prototype has been developed. It makes use of the indoor positioning system installed in the robot laboratory, at Østfold University College, a gyro and a handheld computer.

Success criteria were defined for the prototype and used in the evaluation of the system. Not all were fulfilled because the system had usability problems due to poor accuracy from the positioning system, drift problems and because the virtual model was too small to be fully evaluated. These problems have possible solutions which can be implemented in a new prototype. Despite the problems, the prototype does work and proof the concept. The user experience of the prototype was intuitive and good. The occured problems was accepted. The user will not mind waiting a few seconds for update after movement.

The prototype has reached its goal to proof the concept.

6.3 Future Work

This prototype can be developed further in different areas. It can be improved for indoor use, but it can also be used outside. The next two sub-sections will discuss each of these possibilities, followed by other usages indepent of location.

6.3.1 Indoor

Depending on which area or what type of room we are in, the system can provide additional information in the virtual model. For instance, the model can provide temperature or radiation information in different areas.

The model viewed on the device screen does not have to be a correct representation of the real world. The model can differ in color, include other objects or even display a different room. This feature can be used to:

- try different colors on an object before it is painted in reality

- walk through and evaluate a designed room before it is build

- preview how new inventory will look in a room

6.3.2 Outdoor

The prototype can be modified and developed further for outdoor use. To do so, the indoor positioning system must be replaced by an outdoor positioning system. As mentioned in Section 4.1.1, the GPS has typically an accuracy of a few meters. This can, however, be improved with a variety of different techniqes. One method is the use of DGPS[1] which has a reliable acuracy of 1-3 meters [13].

Let us now assume that the technical device works properly and describe some scenarios for outdoor use. Imagine walking by either a construction area or a planned build area for, for instance, the new opera in town. Would it not be nice to simply download a virtual model of the finnished construction to your handheld computer so you could walk around and look at a visualization of the finnished building from your own point of view. This system might be handy for landscape architects and designers as well as for the public.

Another scenario is to use the system for guiding. By simply viewing a building, you could for instance get information about its history and its current use.

6.3.3 Device Orientation

Another approach of how to use the system, would be to focus on the device orientation. The hand and device gesture can be used as application input. For instance, tilting the device could scroll a document or accelerate or break a car in a driving game.

[1] Differential Global Positioning System

References

[1] Institute for Energy Technology, “Institute for Energy Technology,” http://www.ife.no/index_html-en?set_language=en&cl=en, January 2006.

[2] ——, “Visualisation Technologies Supporting Design, Planning, Operation and Training,” http://www.ife.no/main_subjects_new/mto/visualisering, September 2005.

[3] I. sommerville, *Software Engineering*, 6th ed. England: Addison-Wesley, 2001.

[4] P. K. Pierre N. Robillard and P. d’Astous, *Software Enginering Process with the UPEDU*. Boston: Addison-Wesley, 2003.

[5] Ostfold University College, “Ostfold University College,” http://www.hiof.no, May 2006.

[6] Sonitor Technologies, “Sonitor Ultrasound IPS,” http://www.sonitor.com, February 2006.

[7] MicroStrain, “3DM-GX1,” http://www.microstrain.com/3dm-gx1.aspx, 2005.

[8] HandyWave, “HandyPort Bluetooth Adapter,” http://www.handywave.com/index3.htm, February 2006.

[9] MicroStrain Inc, *3DM-G User Manual*, 2003.

[10] Microsoft, “Microsoft Visual Studio .Net C-Sharp,” http://msdn.microsoft.com/msdnmag/issues/0900/csharp/default.aspx, Desember 2005.

[11] MicroStrain Inc, *3DM-GX1 Data Communications Protocol*, March 2006.

[12] I. Popyrev, “Shared space,” http://www.mic.atr.co.jp/~poup/research/index.html, February 2001.

[13] Kystverket, “Radionavigasjon (DGPS),” http://www.kystverket.no/?aid=9030966, 2006.

List of Figures

List of Tables

Appendix A

Bluetooth Configuration

A HPS-120 device can only connect to one and one base only.

Hyper Terminal Settings

Baudrate: 9600 bps

Data Bit: 8

Parity Bit: None

Stop Bit: 1

Flow Control: None

Emulation: VT100

Configuration

1. Plug HPS-120 into COM port of PC and power it on.
2. Open the Hyper Terminal and set it up.
3. Press the RST button of HPS-120. If you enter the configuration mode successfully, LNK LED will be flashing every second.
4. Hit the <Enter>key, 5 seconds later.

5. Change the configuration of HPS-120 with the following commands:

    - <N>Set the name of the device
    - <M>Set the connection mode to REGISTER and CONNECT mode
    - <C>Set the serial port of the device, each device must have a unique serial port number
    - <A>Set the remote BD_ ADDR to the baud address of the handheld computer
    - <B>Set the baudrate to 57600 for the server and 38600 for the gyro
    - <X>Save and exit the configuration

Appendix B

Server Application Code

```
using System;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;
using Sonitor;

namespace DevoServer2
{
    public partial class DevoMobile : Form
    {
        private Microsoft.DirectX.Direct3D.Font font;

        private bool showmessage = false;
        private string textMessage = "";

        const int numberOfMeshes = 68;
        Mesh[] meshes;

        Vector3[] meshLocations;
        Vector3[] meshBoundingBoxMinValues;
        Vector3[] meshBoundingBoxMaxValues;

        Mesh activeMesh;
        Mesh activeMesh2;
```

```

        Device device;

        public DevoMobile()
        {
            InitializeComponent();

            PresentParameters present = new PresentParameters();

            this.Text = "Devo Server";

            // Enable the form to be closed.
            // Required so that Hwnd of Form changes.
            this.MinimizeBox = false;

            present.Windowed = true;
            present.AutoDepthStencilFormat = DepthFormat.D16;
            present.EnableAutoDepthStencil = true;
            present.SwapEffect = SwapEffect.Discard;

            device = new Device(0, DeviceType.Hardware , this, CreateFlags.
            SoftwareVertexProcessing, present);
            device.DeviceReset += new EventHandler(OnDeviceReset);
            OnDeviceReset(null, EventArgs.Empty);

            font = new Microsoft.DirectX.Direct3D.Font(device,
                new System.Drawing.Font("Arial", 10.0f));

            if (!comportServer.IsOpen)
                comportServer.Open();
            comportServer.DataReceived += ComReadHandler;
        }

        private void OnDeviceReset(object sender, EventArgs e)
        {
            // Meshes must be recreated whenever the device
            // is reset, no matter which pool they are created in.
            meshes = new Mesh[numberOfMeshes];
            meshLocations = new Vector3[numberOfMeshes];
            meshBoundingBoxMinValues = new Vector3[numberOfMeshes];
            meshBoundingBoxMaxValues = new Vector3[numberOfMeshes];
```

```
            activeMesh = null;

            // Create several meshes and associated data.
            for (int i = 0; i < numberOfMeshes; i++)
            {
                GraphicsStream vertexData;

                if (i < 64)
                {
                    meshes[i] = Mesh.Box(device, 1f, 1f, 0.01f);
                    meshLocations[i] = new Vector3((float)(((i % 8) * 1) +
                    0.5f), (float)(((i / 8) * 1) + 0.5f), 0f);
                }
                else if (i == 64)// Draw walls X-axis
                {
                    meshes[i] = Mesh.Box(device, 8f, 0.01f, 8f);
                    meshLocations[i] = new Vector3(0f + 4, 0f, 0f + 4);
                }
                else if (i == 65)// Draw walls Y-axis
                {
                    meshes[i] = Mesh.Box(device, 0.01f, 8f, 8f);
                    meshLocations[i] = new Vector3(0f, 0f + 4, 0f + 4);
                }
                else if (i == 66)// Draw details on wall Y-axis
                {
                    meshes[i] = Mesh.Box(device, 0.4f, 8f, 0.4f);
                    meshLocations[i] = new Vector3(0.2f, 0f + 4, 0f + 2);
                }
                else if (i == 67)// Draw details on wall Y-axis
                {
                    meshes[i] = Mesh.Sphere(device, 0.1f, 360, 36);
                    meshLocations[i] = new Vector3(4f, 4f, 2f + 2);
                }

                // Compute bounding box for a mesh.
                VertexBufferDescription description = meshes[i].VertexBuffer.
                Description;
                vertexData = meshes[i].VertexBuffer.Lock(0, 0, LockFlags.
                ReadOnly);
```

```
                Geometry.ComputeBoundingBox(vertexData, meshes[i].
                NumberVertices, description.VertexFormat, out
                meshBoundingBoxMinValues[i], out meshBoundingBoxMaxValues[i])
                ;
                meshes[i].VertexBuffer.Unlock();
            }

            device.Transform.Projection = Matrix.PerspectiveFovRH((float)Math
            .PI / 4.0F, (float)this.ClientSize.Width / (float)this.ClientSize
            .Height, 1.0f, 100f);

            device.RenderState.Ambient = Color.White;
        }

        protected override void OnPaintBackground(PaintEventArgs e)
        {
            // Do nothing.
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            Material material = new Material();

            // Begin the scene and clear the back buffer to black.
            device.BeginScene();
            device.Clear(ClearFlags.Target | ClearFlags.ZBuffer, Color.Black,
             1.0f, 0);

            // Draw each mesh to the screen
            // The active mesh is drawn in red.
            Color color1, color2, color3;
            color2 = Color.FromArgb(200, 200, 200);
            color1 = Color.FromArgb(220, 220, 220);
            for (int i = 0; i < numberOfMeshes; i++)
            {
                if (i % 8 == 0)
                {
                    color3 = color1;
                    color1 = color2;
                    color2 = color3;
                }
```

```
                if (activeMesh == meshes[i])
                    material.Ambient = Color.Red;
                else if (activeMesh2 == meshes[i])
                    material.Ambient = Color.Blue;
                else
                {
                    if (i == 64)
                        material.Ambient = Color.White;
                    else if (i == 65)
                        material.Ambient = Color.Snow;
                    else if (i == 66)
                        material.Ambient = Color.Silver;
                    else if (i == 67)
                        material.Ambient = Color.Yellow;
                    else
                    {
                        if (i % 2 > 0)
                            material.Ambient = color1;
                        else
                            material.Ambient = color2;
                    }
                }

                device.Transform.World = Matrix.Translation(meshLocations[i]);
                device.Material = material;
                meshes[i].DrawSubset(0);
            }

            //**********************************************************
            device.Transform.Projection = Matrix.PerspectiveFovRH((float)Math.PI / 4.0F, (float)this.ClientSize.Width / (float)this.ClientSize.Height, 1.0f, 100f);

            // Time to poll the get queue
            DcupApi.QueueStatus qs = DcupApi.CheckTheGetQueue();
            //if status is nonzero,
            //we have an incoming message waiting for us
            if (qs.status != 0)
            {
                // Convert the win32 SYSTEMTIME struct into a .net
```

```
                // DateTime class
                System.DateTime time = new System.DateTime(qs.time.wYear, qs.time.wMonth, qs.time.wDay, qs.time.wHour, qs.time.wMinute, qs.time.wSecond, qs.time.wMilliseconds);

                ProcessDcupEvent(qs.evnt, qs.msgid, time);
            }

            System.Threading.Thread.Sleep(100);

            Matrix matView = new Matrix();
            if (mnuFollow.Checked)
            {
                //Vector3 vFromPt = new Vector3(4f, 4f, 2f);
                //Vector3 vLookatPt = new Vector3(4f, 0, 2f);
                Vector3 vFromPt = new Vector3(posX*2, posY*2, posZ*2);
                Vector3 vLookatPt = new Vector3(0, posY * 2, posZ * 2);
                Vector3 vUpVec = new Vector3(0.0f, 0.0f, 1.0f);

                matView = Matrix.LookAtRH(vFromPt, vLookatPt, vUpVec);

                Matrix rotateX = Matrix.RotationAxis(vUpVec, gyroRoll);
                Matrix rotateY = Matrix.RotationAxis(new Vector3(1, 0, 0), gyroPitch);
                Matrix rotateZ = Matrix.RotationAxis(new Vector3(0, 1, 0), gyroYaw-(float)Math.PI);

                matView.Multiply(rotateZ);
                matView.Multiply(rotateX);
                matView.Multiply(rotateY);
            }
            else if (mnuFloor.Checked)
            {
                Vector3 vFromPt = new Vector3(4f, 4f, 11f);
                Vector3 vLookatPt = new Vector3(4f, 4f, 0f);
                Vector3 vUpVec = new Vector3(-0.4f, 0.0f, 0.5f);
                matView = Matrix.LookAtRH(vFromPt, vLookatPt, vUpVec);
            }
            else if (mnuOverview.Checked)
            {
                Vector3 vFromPt = new Vector3(14f, 14f, 5f);
```

```
                Vector3 vLookatPt = new Vector3(2f, 2f, 2f);
                Vector3 vUpVec = new Vector3(0.0f, 0.0f, 1.0f);
                matView = Matrix.LookAtRH(vFromPt, vLookatPt, vUpVec);
            }
            device.SetTransform(TransformType.View, matView);

            // Display gyro data
            if (mnuGyro.Checked)
            {
                font.DrawText(null, "Pitch:\t" + (int)(gyroPitch * 360 / (2 *
                 Math.PI)) + "\r\nRoll:\t" + (int)(gyroRoll * 360 / (2 * Math
                .PI)) + "\r\nYaw:\t" + (int)(gyroYaw * 360 / (2 * Math.PI)),
                new Rectangle(10, this.Height - 90, this.Width, this.Height),
                 DrawTextFormat.NoClip | DrawTextFormat.ExpandTabs |
                DrawTextFormat.WordBreak, Color.Red);
            }
            // Display position
            if (mnuPos.Checked)
            {
                font.DrawText(null, "X:\t" + posX + "\t" + posXUncertainty +
                "\r\nY:\t" + posY + "\t" + posYUncertainty + "\r\nZ:\t" +
                posZ + "\t" + posZUncertainty, new Rectangle(this.Width -
                200, this.Height - 90, this.Width, this.Height),
                DrawTextFormat.NoClip | DrawTextFormat.ExpandTabs |
                DrawTextFormat.WordBreak, Color.Red);
            }
            // Display Message
            if (showmessage)
            {
                font.DrawText(null, textMessage, new Rectangle(10, 30, this.
                Width, this.Height), DrawTextFormat.NoClip | DrawTextFormat.
                ExpandTabs | DrawTextFormat.WordBreak, Color.Red);
            }

            // Finish the scene and present it on the screen.
            device.EndScene();
            device.Present();

            // Render again
            this.Invalidate();
        }
```

```

        System.IO.Ports.SerialPort comportServer = new System.IO.Ports.SerialPort("COM1", 57600, System.IO.Ports.Parity.None, 8, System.IO.Ports.StopBits.One);

        float gyroYaw, gyroPitch, gyroRoll;

        private void ComReadHandler(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            try
            {
                string readline = comportServer.ReadLine();
                string[] s = readline.Split('|');

                    gyroPitch = float.Parse(s[0].Replace('.', ','));
                    gyroRoll = float.Parse(s[1].Replace('.', ','));
                    gyroYaw = float.Parse(s[2].Replace('.', ','));
                    activeMesh = meshes[int.Parse(s[3])];
            }
            catch { }
            this.Invalidate();
            Application.ExitThread();
        }

        private void showMessage(string message)
        {
            textMessage = message;
            showmessage = true;

        }

        private void DevoMobile_FormClosing(object sender, FormClosingEventArgs e)
        {
            comportServer.Close();
            Application.Exit();
        }

        private void DevoMobile_MouseDown(object sender, MouseEventArgs e)
        {
```

```
            // The technique used here is to create a ray through the entire
            // logical 3d space and then perform a bounding box-ray
            // intersection.
            for (int i = 0; i < numberOfMeshes; i++)
            {
                Vector3 nearVector = new Vector3(e.X, e.Y, 0);
                Vector3 farVector = new Vector3(e.X, e.Y, 1);

                // Create ray.
                nearVector.Unproject(device.Viewport, device.Transform.Projection, device.Transform.View, Matrix.Translation(meshLocations[i]));

                farVector.Unproject(device.Viewport, device.Transform.Projection, device.Transform.View, Matrix.Translation(meshLocations[i]));

                farVector.Subtract(nearVector);

                // Perform ray-box intersection test.
                if (Geometry.BoxBoundProbe(meshBoundingBoxMinValues[i], meshBoundingBoxMaxValues[i], nearVector, farVector))
                {
                    // Perform operation on detection of click on mesh object
                    // In this case we designate the mesh as the active
                    // mesh and invalidate the window so that it is redrawn.
                    //activeMeshIndex = i;
                    activeMesh2 = meshes[i];

                    // Send active mesh to device
                    comportServer.WriteLine("a" + i + "\r\n");

                    this.Invalidate();
                    break;
                }
            }
        }

        private void mnuFollow_Click(object sender, EventArgs e)
        {
            mnuFollow.Checked = true;
```

```
            mnuFloor.Checked = false;
            mnuOverview.Checked = false;
        }

        private void mnuFloor_Click(object sender, EventArgs e)
        {
            mnuFloor.Checked = true;
            mnuOverview.Checked = false;
            mnuFollow.Checked = false;
        }

        private void mnuOverview_Click(object sender, EventArgs e)
        {
            mnuOverview.Checked = true;
            mnuFollow.Checked = false;
            mnuFloor.Checked = false;
        }

        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            comportServer.Close();
            Application.Exit();
        }

        /************************** POSITIONING WITH IPS ****************/

        float posX, posY, posZ;
        float posXUncertainty, posYUncertainty, posZUncertainty;

        float[] posfloatX = { 0, 0, 0, 0, 0 };
        float[] posfloatY = { 0, 0, 0, 0, 0 };
        float[] posfloatZ = { 0, 0, 0, 0, 0 };

        /*
        * FUNCTION    ProcessDcupEvent
        *
        * ACCESS      private
        *
        * PURPOSE     Dispatch method for different dcup events.
        *
        * PARAM       int evnt    Event
```

```
        *               int msgid    Message id
        *               DateTime time
        *
        * FUNCTION CALL -
        *
        * RETURN        -
        */
        private void ProcessDcupEvent(int evnt, int msgid,
            System.DateTime time)
        {
            int ip = 0;
            // Force detection of microposition when cluster is detectet
            if (evnt == DcupApi.DBAS_CLUSTER_DETECTION)
              evnt = DcupApi.DBAS_MICROPOSITION;

            switch (evnt)
            {
                // Use the right retrieval function to retrieve
                // the data. If a message isn't retrieved,
                // it will stay on top of the get queue and block
                // other incoming messages until it is removed by
                // the garbage collector, by default xx seconds.
                // therefore, make sure to call the appropriate
                // fetch function or use DropPacket if you aren't
                // interested in the packet
                case DcupApi.DBAS_POLL:
                    //a poll reply from a detector
                    DcupApi.DbasPoll pollData = DcupApi.FetchDbasPoll(msgid,
                    out ip);
                    break;
                case DcupApi.DBAS_MICROPOSITION:
                    // A tag has been detected on a specified
                    // detector, and a 3D position has been found
                    // as well
                    // For a 3D positioning system, this is
                    // the major event to react on
                    DcupApi.DbasMicroPosition mposData = DcupApi.
                    FetchDbasMicroPosition(msgid, out ip);

                    // Oppdater position
                    posXUncertainty = mposData.tagPosUncertaintyX;
```

```
                    posYUncertainty = mposData.tagPosUncertaintyY;
                    posZUncertainty = mposData.tagPosUncertaintyZ;
                    if ((posXUncertainty != 1000 && mposData.tagPosX > 0) || (posXUncertainty < 0.1 && Math.Abs(posX - mposData.tagPosX) > 20 && mposData.tagPosX > 0))
                    {
                        posfloatAdd('X', mposData.tagPosX);
                        posX = posGetMean('X');
                    }
                    if ((posYUncertainty != 1000 && mposData.tagPosY > 0) || (posYUncertainty < 0.1 && Math.Abs(posY - mposData.tagPosY) > 20 && mposData.tagPosY > 0))
                    {
                        posfloatAdd('Y', mposData.tagPosY);
                        posY = posGetMean('Y');
                    }
                    if ((posZUncertainty != 1000 && mposData.tagPosZ > 0) || (posZUncertainty < 0.1 && Math.Abs(posZ - mposData.tagPosZ) > 20 && mposData.tagPosZ > 0))
                    {
                        posfloatAdd('Z', mposData.tagPosZ);
                        posZ = posGetMean('Z');
                    }

                    // Send position
                    comportServer.WriteLine("p|" + posX + "|" + posY + "|" + posZ + "\r\n");

                    break;

                default:
                    // We're not interested in this packet,
                    // drop it on the floor
                    try { DcupApi.DropPacket(msgid); }
                    catch { }
                    break;
            }
        }

        private void mnuPollIPS_Click(object sender, EventArgs e)
        {
```

```
            showMessage("IPS Polled");
        }

        private void posfloatAdd(char akse, float value)
        {
            switch (akse)
            {
                case 'X':
                    posfloatX[4] = posfloatX[3];
                    posfloatX[3] = posfloatX[2];
                    posfloatX[2] = posfloatX[1];
                    posfloatX[1] = posfloatX[0];
                    posfloatX[0] = value;
                    break;
                case 'Y':
                    posfloatY[4] = posfloatY[3];
                    posfloatY[3] = posfloatY[2];
                    posfloatY[2] = posfloatY[1];
                    posfloatY[1] = posfloatY[0];
                    posfloatY[0] = value;
                    break;
                case 'Z':
                    posfloatZ[4] = posfloatZ[3];
                    posfloatZ[3] = posfloatZ[2];
                    posfloatZ[2] = posfloatZ[1];
                    posfloatZ[1] = posfloatZ[0];
                    posfloatZ[0] = value;
                    break;
            }
        }

        private float posGetMean(char akse)
        {
            switch (akse)
            {
                case 'X':
                    return ((posfloatX[0] + posfloatX[1] + posfloatX[2] +
                    posfloatX[3] + posfloatX[4]) / 5);
                case 'Y':
                    return ((posfloatY[0] + posfloatY[1] + posfloatY[2] +
                    posfloatY[3] + posfloatY[4]) / 5);
```

```
                case 'Z':
                    return ((posfloatZ[0] + posfloatZ[1] + posfloatZ[2] +
                    posfloatZ[3] + posfloatZ[4]) / 5);
            }
            return 0;
        }
    }
}
```

Appendix C

Mobile Application Code

```
using System;
using System.Drawing;
using System.Windows.Forms;
using Microsoft.WindowsMobile.DirectX;
using Microsoft.WindowsMobile.DirectX.Direct3D;

namespace DevoMobile
{
    class Devo : Form
    {
        private Microsoft.WindowsMobile.DirectX.Direct3D.Font font;

        const int numberOfMeshes = 68;
        Mesh[] meshes;

        float posX, posY, posZ;
        float posXUncertainty, posYUncertainty, posZUncertainty;

        Vector3[] meshLocations;
        Vector3[] meshBoundingBoxMinValues;
        Vector3[] meshBoundingBoxMaxValues;

        Mesh activeMesh;
        Mesh activeMesh2;
        int activeMeshIndex;

        Device device;
```

```

        public Devo()
        {
            PresentParameters present = new PresentParameters();

            this.Text = "Devo Mobile";

            // Enable the form to be closed.
            // Required so that Hwnd of Form changes.
            this.MinimizeBox = false;

            present.Windowed = true;
            present.AutoDepthStencilFormat = DepthFormat.D16;
            present.EnableAutoDepthStencil = true;
            present.SwapEffect = SwapEffect.Discard;

            device = new Device(0, DeviceType.Default, this, CreateFlags.None
            , present);

            device.DeviceReset += new EventHandler(OnDeviceReset);
            OnDeviceReset(null, EventArgs.Empty);

            font = new Microsoft.WindowsMobile.DirectX.Direct3D.Font(device,
            new System.Drawing.Font("Arial", 10.0f, FontStyle.Regular));

            if (!comportServer.IsOpen)
                comportServer.Open();
            comportServer.DataReceived += ComReadHandler;
        }

        private void OnDeviceReset(object sender, EventArgs e)
        {
            // Meshes must be recreated whenever the device
            // is reset, no matter which pool they are created in.
            meshes = new Mesh[numberOfMeshes];
            meshLocations = new Vector3[numberOfMeshes];
            meshBoundingBoxMinValues = new Vector3[numberOfMeshes];
            meshBoundingBoxMaxValues = new Vector3[numberOfMeshes];
            activeMesh = null;

            // Create several meshes and associated data.
```

```
        for (int i = 0; i < numberOfMeshes; i++)
        {
            GraphicsStream vertexData;

            if (i < 64)
            {
                meshes[i] = Mesh.Box(device, 1f, 1f, 0.01f);
                meshLocations[i] = new Vector3((float)(((i % 8) * 1) +
                0.5f), (float)(((i / 8) * 1)+0.5f), 0f);
            }
            else if (i == 64)// Draw walls X-axis
            {
                meshes[i] = Mesh.Box(device, 8f, 0.01f, 8f);
                meshLocations[i] = new Vector3(0f+4, 0f, 0f+4);
            }
            else if (i == 65)// Draw walls Y-axis
            {
                meshes[i] = Mesh.Box(device, 0.01f, 8f, 8f);
                meshLocations[i] = new Vector3(0f, 0f+4, 0f+4);
            }
            else if (i == 66)// Draw details on wall Y-axis
            {
                meshes[i] = Mesh.Box(device, 0.4f, 8f, 0.4f);
                meshLocations[i] = new Vector3(0.2f, 0f + 4, 0f + 2);
            }
            else if (i == 67)// Draw details on wall Y-axis
            {
                meshes[i] = Mesh.Sphere(device, 0.1f, 360, 36);
                meshLocations[i] = new Vector3(4f, 4f, 2f + 2);
            }

            // Compute bounding box for a mesh.
            VertexBufferDescription description = meshes[i].VertexBuffer.
            Description;
            vertexData = meshes[i].VertexBuffer.Lock(0, 0, LockFlags.
            ReadOnly);
            Geometry.ComputeBoundingBox(vertexData, meshes[i].
            NumberVertices, description.VertexFormat, out
            meshBoundingBoxMinValues[i], out meshBoundingBoxMaxValues[i])
            ;
            meshes[i].VertexBuffer.Unlock();
```

```
            }

            device.Transform.Projection = Matrix.PerspectiveFovRH((float)Math
            .PI / 4.0F, (float)this.ClientSize.Width / (float)this.ClientSize
            .Height, 1.0f, 100f);

            device.RenderState.Ambient = Color.White;
        }

        protected override void OnPaintBackground(PaintEventArgs e)
        {
            // Do nothing.
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            Material material = new Material();

            // Begin the scene and clear the back buffer to black.
            device.BeginScene();
            device.Clear(ClearFlags.Target | ClearFlags.ZBuffer, Color.Black,
             1.0f, 0);

            // Draw each mesh to the screen
            // The active mesh is drawn in red.
            Color color1, color2, color3;
            color2 = Color.FromArgb(200, 200, 200);
            color1 = Color.FromArgb(220, 220, 220);
            for (int i = 0; i < numberOfMeshes; i++)
            {
                if (i % 8 == 0)
                {
                    color3 = color1;
                    color1 = color2;
                    color2 = color3;
                }
                if (activeMesh == meshes[i])
                    material.Ambient = Color.Red;
                else if (activeMesh2 == meshes[i])
                    material.Ambient = Color.Blue;
                else
```

```
                {
                    if (i == 64)
                        material.Ambient = Color.White;
                    else if (i == 65)
                        material.Ambient = Color.Snow;
                    else if (i == 66)
                        material.Ambient = Color.Silver;
                    else if (i == 67)
                        material.Ambient = Color.Yellow;
                    else
                    {
                        if (i % 2 > 0)
                            material.Ambient = color1;
                        else
                            material.Ambient = color2;
                    }
                }

                device.Transform.World = Matrix.Translation(meshLocations[i])
                ;
                device.Material = material;
                meshes[i].DrawSubset(0);
            }

            //**************************************************************
            device.Transform.Projection = Matrix.PerspectiveFovRH((float)Math
            .PI / 4.0F, (float)this.ClientSize.Width / (float)this.ClientSize
            .Height, 1.0f, 100f);

            // Set the view matrix.
            GetGyroAxis();
            System.Threading.Thread.Sleep(10);

            Matrix matView;
            Vector3 vFromPt = new Vector3(posX * 2, posY * 2, posZ * 2);
            Vector3 vLookatPt = new Vector3(0, posY * 2, posZ * 2);
            //Vector3 vFromPt = new Vector3(4f, 4f, 2f);
            //Vector3 vLookatPt = new Vector3(0, 4f, 2f);

            Vector3 vUpVec = new Vector3(0.0f, 0.0f, 1.0f);
            matView = Matrix.LookAtRH(vFromPt, vLookatPt, vUpVec);
```

```

            Matrix rotateX = Matrix.RotationAxis(vUpVec, gyroRoll);
            Matrix rotateY = Matrix.RotationAxis(new Vector3(1, 0, 0), gyroPitch);
            Matrix rotateZ = Matrix.RotationAxis(new Vector3(0, 1, 0), gyroYaw-(float)Math.PI);
            matView.Multiply(rotateZ);
            matView.Multiply(rotateX);
            matView.Multiply(rotateY);

            device.SetTransform(TransformType.View, matView);

            font.DrawText(null, "X:\t" + posX// + "\t" + posXUncertainty + "\r\nY:\t" + posY// + "\t" + posYUncertainty + "\r\nZ:\t" + posZ,// + "\t" + posZUncertainty, new Rectangle(10, this.Height - 90, this.Width, this.Height), DrawTextFormat.NoClip | DrawTextFormat.ExpandTabs | DrawTextFormat.WordBreak, Color.Red);

            font.DrawText(null, "Pitch:\t" + (int)(gyroPitch * 360 / (2 * Math.PI)) + "\r\nRoll:\t" + (int)(gyroRoll * 360 / (2 * Math.PI)) + "\r\nYaw:\t" + (int)(gyroYaw * 360 / (2 * Math.PI)), new Rectangle(this.Width-150, this.Height - 90, this.Width, this.Height), DrawTextFormat.NoClip | DrawTextFormat.ExpandTabs | DrawTextFormat.WordBreak, Color.Red);

            // Finish the scene and present it on the screen.
            device.EndScene();
            device.Present();

            // Render again
            this.Invalidate();
        }

        // This method demonstrates picking.
        protected override void OnMouseDown(MouseEventArgs e)
        {
            // The technique used here is to create a ray through the entire
            // logical 3d space and then perform a bounding box-ray
            // intersection.
            for (int i = 0; i < numberOfMeshes; i++)
            {
```

```
                    Vector3 nearVector = new Vector3(e.X, e.Y, 0);
                    Vector3 farVector = new Vector3(e.X, e.Y, 1);

                    // Create ray.
                    nearVector.Unproject(device.Viewport, device.Transform.
                    Projection, device.Transform.View, Matrix.Translation(
                    meshLocations[i]));

                    farVector.Unproject(device.Viewport, device.Transform.
                    Projection, device.Transform.View, Matrix.Translation(
                    meshLocations[i]));

                    farVector.Subtract(nearVector);

                    // Perform ray-box intersection test.

                    if (Geometry.BoxBoundProbe(meshBoundingBoxMinValues[i],
                    meshBoundingBoxMaxValues[i], nearVector, farVector))
                    {
                        // Perform operation on detection of click on mesh object
                        // In this case we designate the mesh as the active
                        // mesh and invalidate the window so that it is redrawn.
                        activeMeshIndex = i;
                        activeMesh = meshes[i];
                        this.Invalidate();
                        break;
                    }
                }
            }

            static void Main()
            {
                Application.Run(new Devo());
            }

            private void InitializeComponent()
            {
                this.SuspendLayout();
                // Devo
                this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Inherit;
                this.ClientSize = new System.Drawing.Size(240, 320);
```

```
            this.ControlBox = false;
            this.KeyPreview = true;
            this.Location = new System.Drawing.Point(0, 0);
            this.MinimizeBox = false;
            this.Name = "Devo";
            this.TopMost = true;
            this.WindowState = System.Windows.Forms.FormWindowState.Maximized
            ;
            this.Closing += new System.ComponentModel.CancelEventHandler(this
            .Devo_Closing);
            this.KeyDown += new System.Windows.Forms.KeyEventHandler(this.
            Devo_KeyDown);
            this.ResumeLayout(false);
        }

        private void Devo_Closing(object sender, System.ComponentModel.
        CancelEventArgs e)
        {
            device.Dispose();
            comportGyro.Close();
            Application.Exit();
        }

        System.IO.Ports.SerialPort comportGyro = new System.IO.Ports.
        SerialPort("COM8", 38600, System.IO.Ports.Parity.None, 8, System.IO.
        Ports.StopBits.One);

        float gyroYaw, gyroPitch, gyroRoll;
        float oldYaw, oldPitch, oldRoll;

        void GetGyroAxis()
        {
            if (!comportGyro.IsOpen)
                comportGyro.Open();
            byte[] bs = new byte[11];
            int bsCount = 0;
            int c = 14;
            try
            {
                comportGyro.Write(((char)c).ToString());
                for (int i = 0; i < 11; i++)
```

```
                {
                    int input = comportGyro.ReadByte();
                    bs[i] = (byte)input;
                    bsCount++;
                }
                comportGyro.ReadExisting();
            }
            catch { }

            if ((int)bs[0] == c && bsCount > 8)
            {
                gyroRoll = (float)((((int)bs[1] << 8) + (int)bs[2]) * 2 *
                Math.PI / 65536);
                gyroPitch = (float)((Math.PI * 2) - ((((int)bs[3] << 8) + (
                int)bs[4]) * 2 * Math.PI / 65536));
                gyroYaw = (float)(((((int)bs[5] << 8) + (int)bs[6]) * 2 *
                Math.PI / 65536));
            }

            // Send gyro data to server
            if (!comportServer.IsOpen)
                comportServer.Open();
            try { comportServer.WriteLine(gyroPitch + "|" + gyroRoll + "|" +
            gyroYaw + "|" + activeMeshIndex); }
            catch { }
        }

        /********* POSITION **********/

        System.IO.Ports.SerialPort comportServer = new System.IO.Ports.
        SerialPort("COM0", 57600, System.IO.Ports.Parity.None, 8, System.IO.
        Ports.StopBits.One);

        private void ComReadHandler(object sender, System.IO.Ports.
        SerialDataReceivedEventArgs e)
        {
            string readline = "";
            try
            {
                readline = comportServer.ReadExisting();

```

```
                if (readline[0].Equals('a') && readline.Length >= 2)
                {
                    activeMesh2 = meshes[int.Parse(readline.Substring(1))];
                }

                if (readline[0].Equals('p') && readline.Length >= 2)
                {
                    string[] s = readline.Split('|');
                    posX = float.Parse(s[1].Replace(',', '.'));
                    posY = float.Parse(s[2].Replace(',', '.'));
                    posZ = float.Parse(s[3].Replace(',', '.'));
                }
            }
            catch { }
        }

        private void Devo_KeyDown(object sender, KeyEventArgs e)
        {
            if ((e.KeyCode == System.Windows.Forms.Keys.Enter))
            {
                device.Dispose();
                comportGyro.Close();
                Application.Exit();
            }
        }

        float[] yawFloat = { 0, 0, 0, 0, 0 };
        float[] pitchFloat = { 0, 0, 0, 0, 0 };
        float[] rollFloat = { 0, 0, 0, 0, 0 };

        private void gyroAdd(char akse, float value)
        {
            switch (akse)
            {
                case 'Y':
                    yawFloat[4] = yawFloat[3];
                    yawFloat[3] = yawFloat[2];
                    yawFloat[2] = yawFloat[1];
                    yawFloat[1] = yawFloat[0];
                    yawFloat[0] = value;
                    break;
```

```
                case 'P':
                    pitchFloat[4] = pitchFloat[3];
                    pitchFloat[3] = pitchFloat[2];
                    pitchFloat[2] = pitchFloat[1];
                    pitchFloat[1] = pitchFloat[0];
                    pitchFloat[0] = value;
                    break;
                case 'R':
                    rollFloat[4] = rollFloat[3];
                    rollFloat[3] = rollFloat[2];
                    rollFloat[2] = rollFloat[1];
                    rollFloat[1] = rollFloat[0];
                    rollFloat[0] = value;
                    break;
            }
        }

        private float gyroGetMean(char akse)
        {
            switch (akse)
            {
                case 'Y':
                    return ((yawFloat[0] + yawFloat[1] + yawFloat[2] +
                    yawFloat[3] + yawFloat[4]) / 5);
                case 'P':
                    return ((pitchFloat[0] + pitchFloat[1] + pitchFloat[2] +
                    pitchFloat[3] + pitchFloat[4]) / 5);
                case 'R':
                    return ((rollFloat[0] + rollFloat[1] + rollFloat[2] +
                    rollFloat[3] + rollFloat[4]) / 5);
            }
            return 0;
        }
    }
}
```