UPTEC IT07 014
Examensarbete 20 p
April 2007

OSREP

Open-Set REalistic Pinpointing

Johan Klövstedt

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

OSREP - Open-Set REalistic Pinpointing

Johan Klövstedt

Using what researchers call backseat games the kids could be occupied while traveling in a car and not bother parents with nag like "Are we there yet". These kinds of games transform boring trips into exciting adventures by combining and connecting real world and computer generated data. One of these backseat games is a research prototype called Backseat Playground (BSP). This prototype is based on narrative stories which evolve through interaction with the surrounding environment. By using a customized gaming device for acquiring position of where the player is located and direction that the player is pointing the device kids are able to interact with the surroundings of the car. In the current BSP prototype the player can only pinpoint objects that are defined by BSP itself while traveling through a variety of surrounding environments. In this thesis project it is investigated if there is a possibility to make the pinpointing more realistic. Realistic in the way that the algorithm senses if the player has started to pinpoint an interesting object and further on it also pinpoints that object's location. This new generation of pinpointing is called Open-Set REalistic Pinpointing (OSREP) since it extends a closed set of objects into the amount of objects that the player(s) would like to put into the world of the BSP. Furthermore this opens up for the possibility for users to create their own content at that location. The purpose of this thesis project is to create algorithms that serves as proofs of concept showing that it is possible to carry out the necessary calculations needed to produce the results wanted for the BSP game prototype. Therefore there are some extensive investigations and examinations made as to how a player would be pinpointing objects and also how the sensing of players sight can be calculated. These examinations were in fact test runs made by myself in the same environment that the BSP prototype is tested. To be able to conduct such test runs a simplistic data acquirement tool was created. The investigations carried out was done by first study the relevant publications made in the area of mobile AR games and also study the publications made regarding the BSP prototype. Consecutively there were some studies made in the areas of mathematical statistics, geometry and probability to show the theories behind the OSREP algorithm.

One objective during development of the OSREP algorithm was that they should be easy to integrate into the BSP prototype. This was accomplished through a thoroughly investigation of the current version of the code written for the BSP prototype. Although some modifications regarding the handling of direction data has been made and also regarding the conversions of positional data. A reason for this was to be able to create the OSREP algorithms as a standalone process so that there would not be a need for running the actual BSP game or its simulator. Another objective was to visualize the OSREP performance in the real world. Although this has not been completely finished there are results presented in this thesis in a more primitive way than wished for. The reason for not completing such a visualization application is that the time constraint put on this kind of thesis project is too narrow. But at least a foundation has been made and could certainly be continued by another master student.

Although some setbacks have been experienced regarding the way the OSREP algorithms was visualized the algorithms seem to work in both in theory and on the acquired sensor data. The primitive visualization made only serves as proof-of-concept and could be used as an aid for making sure further efforts of integration of the OSREP algorithms into BSP is worth the cost in effort and time. The major result of the OSREP algorithm was that it shows a sufficiently accurate location of the pinpointed object. A very nice feature of the algorithm is that it does not need any external triggering to make the distinction as to when the player has started to aim. It could rather be used continuously to examine the recorded data to calculate the location of pinpointed objects.

Handledare: Anton Gustafsson
Ämnesgranskare: Catharina Carlemalm Logothetis
Examinator: Anders Jansson
ISSN: 1401-5749, UPTEC IT07 014
Tryckt av: Ångströmlaboratoriet, Uppsala Universitet

Sammanfattning

Denna rapport är en del av ett examensarbete utfört hos Interactive Institute's Mobility Studio. Examensarbetet är en del av min utbildning till civilingenjör i Informationsteknologi vid Uppsala Universitet.

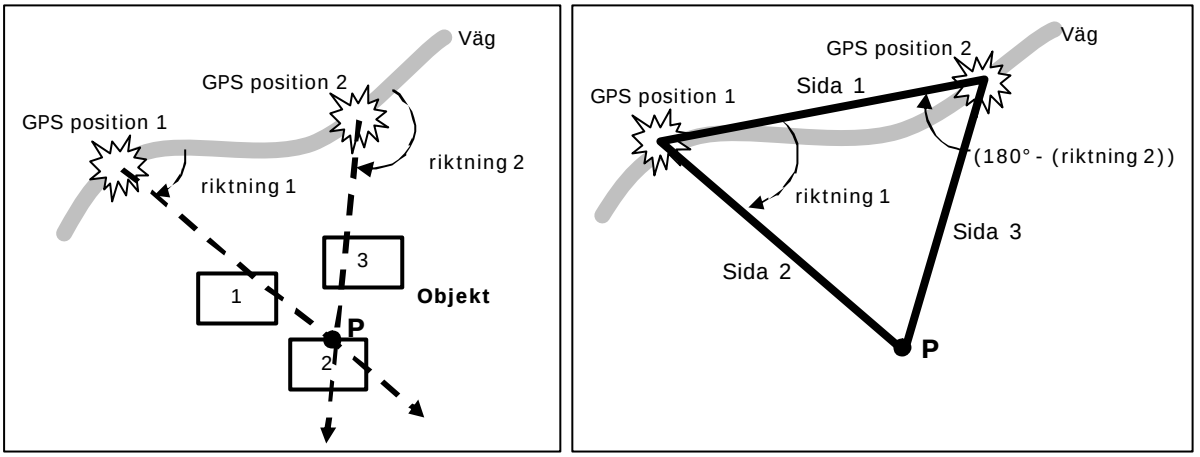
Vi vet alla att det kan vara oerhört tråkigt att som passagerare i en bil sitta inaktiv bredvid eller bakom föraren. Detta gäller speciellt barn. "Är vi framme snart?" är troligtvis en av de vanligaste fraserna en förälder kan höra från baksätet i en bil. Men hopp finns även för dessa barn. I form av baksätesspel kan barn och ungdomar roa sig på egen hand.

Ett sådant spel är det så kallade Backseat Playground (BSP) vilket är ett mobilt verklighetsförstärkt spel (dvs. ett dataspel med en förstärkt koppling till verkligheten). BSP-spelprototypen är enligt en nyligen publicerad rapport av Gustafsson, Bichard, Brunnberg, Juhlin och Combetto (2006) beskrivet som ett spel baserat på berättande historier som utvecklas genom interaktion med miljön som omringar bilen medan man reser genom vägnätet. Den skiftande miljön triggar händelser och ljud som kan ta spelaren till nästa steg i spelet. Men ett sådant steg kan vanligtvis bara tas om spelaren lyckas utföra ett uppdrag. Uppdragen består bland annat av att lokalisera ursprunget av ljud eller lokalisera objekt som ljudet kan komma ifrån. På detta sätt kan BSP förvandla kyrkor, broar och andra objekt vid sidan av vägen till en äventyrlig värld fylld med virtuella varelser, skatter och spännande uppdrag.

Spelkontrollen som spelaren använder i Backseat Playground.

För att kunna koppla ihop världen utanför bilen med spelaren som spelar BSP så behövs en speciell spelkontroll. Spelkontrollen fungerar som en riktbar mikrofon med vilken barnen kan interagera med spelobjekt. Den här riktbara mikrofonen innehåller sensorer för att kunna bestämma dess rumsliga rörelser och riktning.

För att kunna välja ut vilka representabla objekt som BSP kan nyttja under spelets gång används data från så kallad Geografiska Informations System (GIS). Sådan data berättar mer om ett kartobjekt till exempel att en fyrkant på kartan är en kyrka eller sporthall. Med hjälp av av sådan data kombinerad med en karta och en GPS så kan man bestämma spelarens position och därmed också var objektet ligger i förhållande till spelaren. Utöver det så kan man med spelkontrollen se om spelaren siktar på objektet. Detta bestämmer man genom att triangulera GPS-positionerna med skärningspunkten för två vinklar som spelaren riktar kontrollen i. Enligt figuren nedan ser man hur detta är tänkt och fungerar.



Triangulering med två GPS-positioner och med två riktningar spelaren siktar i.

Då detta sätt att sikta gäller för en sluten mängd siktbara objekt (dvs. objekt som BSP definierar som siktbara under spelets gång) skulle det inte fungera i en framtida version av BSP. I en framtida version vill man nämligen kunna implementera användar-skapat-innehåll. För att kunna göra det så måste spelaren kunna få bestämma en position i den verkliga världen där informationen ska ligga. Alltså behövs en ny generation av siktning. Det är denna nya generation som detta examensarbete ska resultera i eller närmare bestämt undersöks om det skulle kunna gå att implementera en algoritm som kan utföra sådan sikting i BSP prototypen. Denna nya generation av siktning kallas OSREP som är en akronym för Open-Set Realistic Pinpointing.

Genom att undersöka hur en spelare siktar; det vill säga hur riktningsdatan ser ut under en sådan siktning liksom när man ska triangulera data, så resulterar det i OSREP algoritmerna. Genom att detektera mönster i riktningsdata så kan den applikation där OSREP är integrerad känna av när spelaren siktar på något. Därefter kan man rent logiskt se när spelaren slutar sikta och följdaktligen triangulerar man positionerna och riktinngarna för tiden emellan start och slut. Positionen som detta resulterar i blir då positionen som spelaren kan lägga ut information på i den framtida versionen av BSP.

Detta kanske låter väldigt lätt implementerbart men faktum är att en hel del ingående studier har skett över publikationer rörande BSP, mobila spel, matematik, statistik och sannolikhet. Dock har mest arbete lagts ned i att analysera den programmerade koden som redan utvecklats för BSP. Syftet med denna analysering är att snabbare kunna utveckla ett verktyg som kan visualisera OSREP med en bakomliggande karta på vilken bilen syns tillsammans med de riktningar och positioner som registrerats under testkörningar. Det arbetet har jag inte hunnit klart med. Därför förenklades visualiseringen till den milda graden att man ser hur de matematiska resultaten ser ut i form av grafer. Ett annat syfte med att analysera BSP-koden är att eftersom OSREP ska kunna integreras i BSP så måste den ju hantera data som även är tillgänglig när BSP spelet körs.

För att tillgodogöra mig datan som OSREP utvecklats utifrån har testkörningar gjorts med ett egenhändigt producerat inspelningsverktyg som körs på BSP-spelkontrollen.

De producerade resultaten är i algoritmens aktuella version tillfredställande eftersom algoritmen producerar en position för det siktade objektet. Dessutom sköter algoritmen behandlingen av insamlad data kontinuerligt vilket medför att spelaren inte behöver trycka på någon knapp för att signalera att han eller hon börjat sikta in sig på ett objekt.

Preface

This report serves as my master thesis project report, the master thesis project being a part of my education at the Department of Information Technology at Uppsala University. This report is a result of the work that I have done at the Interactive Institute's Mobility studio. The work done will hopefully contribute to some extent in the area of game controls in mobile augmented-reality games. Also it will hopefully help the team at the Mobility studio to progress with their Backseat Playground game prototype, and that these algorithms could help improve the game so it may take the game to its next conceptual level. I.e. future development of User Content Creation tools using the algorithms I have made. Not saying this is a report solely about User Content Creation, it is rather an examination of the possibilities of using the currently used gaming device.

The report is intended for people with a strong technical interest since there are a greatly technically and programmatically language used. But of course my intension for this report is to explain every definition and solution in the way it would be easily understood.

I would like to acknowledge the influence of my co-workers at Interactive Institute Mobility studio that has helped me in various ways. The presentations and meetings together with them have given me a good insight into the research industry. A special thanks to Anton Gustafsson, Liselott Brunnberg and Oskar Juhlin that provided me with good feedback and constructive criticism about the report and design issues. Also I would like to thank John Bichard for abstract but useful thoughts and discussions during this thesis project. Another thing that has given me much joy to have been a part of; are the lectures and discussions with some of the guest researchers at Interactive Institute, such as Barry Brown and Louise Barkhuus who has influenced me in various ways.

Johan Klövstedt
Uppsala, April 2007

Table of Contents

Abstract i
Sammanfattning ii
Preface iv
Table of Contents vi
Tables and Figures viii

1. INTRODUCTION 1

1.1 Background 1

1.2 Problem description 2

1.3 Purpose 4

1.4 Delimitations 4

1.5 Related Work 5

2. METHODOLOGY 7

2.1 Customizing a Model 7

3. HARDWARE 9

3.1 Backseat Playground Hardware 9
3.1.1 GPS 10
3.1.2 Accelerometers, Gyros and Magnetometers 12

3.2 OSREP Hardware 14

4. THE OSREP THEORIES 15

4.1 Definitions 15
4.1.1 Investigating valid road segment types 15
4.1.2 How does a player pinpoint an object 17

4.2 First algorithm of OSREP 19

4.3 Second algorithm of OSREP 20
4.3.1 Line Equations and Line Intersections 21
4.3.2 Weighing POIs to Achieve POI Convergence 23
4.3.3 Initial Angle – The Offset 23

5. THE OSREP IMPLEMENTATIONS 25

5.1 The tpPPC Application 25
5.1.1 Part 1 of the model – A tool preparation part 25
5.1.2 Conceptual Design of the tpPPC Application 26

5.2 Conducting Test Runs 28

5.3 Visualization Applications 29
5.3.1 Part 2 of the model – Analyzing and Visualizing 29
5.3.2 Conceptual Design of the GISObjectApp 31
5.3.3 Python Script Hacks 32

5.4 Implementing OSREP Algorithms 32

6. RESULTS 35

6.1 Results for the tpPPC application 35

6.2 Results from the Test Runs 36

6.3 The GISObjectApp visualization application 37

6.4 OSREP Results 39
6.4.1 Results of the First OSREP Algorithm 39
6.4.2 Results of the Second OSREP Algorithm 40

7. DISCUSSING AND CONCLUDING THE RESULTS 45

7.1 The Applications 45
7.1.1 The tpPPC 45
7.1.2 The GISObjectApp 45
7.1.3 The Python Hack 45

7.2 Are the OSREP Algorithms Sufficiently Good? 46

7.3 Lessons learned 46

7.4 Future Work 47

8. ACKNOWLEDGEMENTS 49

9. REFERENCES 51

9.1 Internet 51

9.2 Literature 52

9.3 Publications 52

10. APPENDICES 55

10.1 Appendix A – Calculating a sector of a circle 55

10.2 Appendix B – Maximum valid AngRates 56

10.3 Appendix C – GPS Coordinate Transformation 58

10.4 Appendix D – The OSREP Testing Feedback 59

10.5 Appendix E – The Test Run Notes 61

10.6 Appendix F – Code for the second OSREP Algorithm 62

Tables and Figures

Table 3.1.1.2: *This table shows the sources of error for the GPS positions.* 10
Table 4.1.1.2: *Distance traveled at a certain speed during one second.* 16
Table 4.1.1.3: *Angles extracted from the calculation in Appendix B scenarios.* 16
Table 10.5.1: *Notes taken during 15 test runs at Lidingö.* 61

Sammanfattningsfigur: *Spelkontrollen som spelaren använder i Backseat Playground.* iii
Sammanfattningsfigur: *Triangulering.* iv
Figure 1.1.1: *The gaming device.* 1
Figure 1.1.2; *Left: Triangulation. Right: Imagined triangle.* 2
Figure 2.1.1: *Structural model of execution of the master thesis project.* 8
Figure 3.1.1: *Hardware needed for the Backseat Playground game.* 9
Figure 3.1.1.1: *Acquiring a GPS position.* 10
Figure 3.1.1.3: *Scenario with GPS position errors.* 11
Figure 3.1.2.1: *Shows the Microstrain sensor module's coordinate system.* 12
Figure 3.1.2.2: *Shows the gimbals and how the sensor platform is placed in a gyro.* 13
Figure 3.1.2.3: *In this scenario it seems like the player is tracking object 2.* 13
Figure 3.1.2.4: *In this scenario the concept of the Microstrain sensor errors is shown.* 14
Figure 4.1.1.1: *Road scenarios.* 15
Figure 4.1.1.4: *Approximate road segment given the calculations made.* 17
Figure 4.1.2.1: *Showing how pinpointing should be done from the player's perspective.* 18
Figure 4.2.1: *Result of how the variance looks like for the first OSREP algorithm.* 20
Figure 4.3.1.1: *Line equations and line intersection.* 21
Figure 4.3.1.9: *The pseudo-code for finding where two lines intersect.* 23
Figure 5.1.1.1: *First part of the model.* 25
Figure 5.1.2.1: *The conceptual design of the tpPPC application.* 26
Figure 5.2.1: *Example of test run showing player path and the pinpointed object.* 28
Figure 5.2.2: *Distance B between road and object.* 29
Figure 5.3.1.1: *Second part of the model.* 29
Figure 5.3.2.1: *The conceptual design of GISObjectApp.* 31
Figure 5.3.3.1: *Third part of the model.* 32
Figure 6.1.1: *The tpPPC tool on a PDA.* 35
Figure 6.2.1: *The acquired button states.* 36
Figure 6.2.2: *The acquired GPS positions.* 36
Figure 6.2.3: *The acquired gyro data (with two A4-pages of data cut out).* 37
Figure 6.3.1: *A visual of the GISObjectApp.* 38
Figure 6.4.1.1: *Result of how the variance looks like for the first OSREP algorithm.* 39
Figure 6.4.1.2: *This graph shows that the angle is in fact increasing* 40
Figure 6.4.2.1: *Example of how the result looks like for the second OSREP algorithm.* 41
Figure 6.4.2.2: Close-up of the pinpointed object and its radius. 41
Figure 6.4.2.3: *OSREP with a finite button state interval and offset angle = 30 degrees.* 42
Figure 6.4.2.4: *OSREP without a finite button state interval.* 43
Figure 10.1.2: *The variables used showed in a circle.* 55
Figure 10.2.1: *Explanatory image of the variables used when calculating maximum AngRate.* 56
Figure 10.2.4: *Resulting graphs showing the maximum AngRate at five different speeds.* 57
Figure 10.3.1: *Python code for transforming GPS positions in WGS84 format to RT90 format.* 58
Figure 10.4.1: *Code feedback: With Button State Interval* 59
Figure 10.4.2: *Code feedback: With Button State Interval* 60
Figure 10.6.1: The python code for the second OSREP algorithm 62

1. Introduction

In this chapter you find the background for this thesis project and consecutively the description of the problem/assignment along with the delimitations and the related work that has been done in the area of mobile augmented-reality games and in the area of motion sensitive gaming controls/devices.

1.1 Background

To travel by car can be very tedious, especially for kids that are inactive passengers. “Are we there yet?” is for certain one of the most common phrases that parents could hear from the backseat of a car. The phrase in itself is just another way for kids to express their boredom. Looking at the problem from the children’s point of view there is some hope to be found through development of so called backseat games. One prototype for such a game is called Backseat Playground (BSP) which is a mobile augmented-reality (AR) game (also called pervasive game). I.e. a game using a combination of real world and computer generated data. This game is intended to be played by kids in the backseat of a car. The BSP game prototype is in a recently published article by Gustafsson, Bichard, Brunnberg, Juhlin and Combetto (2006) described as a game based on narrative stories which evolves through interaction with the surrounding environment while traveling through the road network. The changing scenery in the surrounding environment will trigger events and sounds that take the player to the next step in the story, but only if the player succeeds in performing the required actions for them. Such a required action could be to pinpoint the origin of a sound or locating an object that corresponds to the sound played in the headphones that the player is wearing. In this way the game turns churches, bridges and other roadside objects into an adventurous world filled with virtual creatures, treasures and exciting missions.

Figure 1.1.1: The gaming device that is used when playing Backseat Playground. The Pocket PC (PDA) is attached to the device itself.

To make the connection between the players playing the game and the world on the outside of the car, a customized gaming device has to be used (shown in figure 1.1.1). This gaming device[1] works like a directional microphone with which the kids are able to interact with the

[1] Further described in chapter 3 that explains the hardware used in this master thesis project.

game objects[2]. This directional microphone contains sensors to determine its spatial position and direction. For now it is sufficient to say that the device is equipped with two sensors, a GPS for acquiring the physical location of the player and a gyro that senses how the gaming device is spatially pointed. To put the usability of this kind of gaming device in a context, I'm considering what Brunnberg and Juhlin (2003) pointed out in their article "Motion and Spatiality in a Gaming Situation – Enhancing Mobile Computer Games with the Highway Experience". A part of their investigation was the importance of choosing recognizable objects and what makes a significant object for enhancing game experience. The knowledge gathered through that investigation has been assimilated in the BSP prototype making a strong connection between the narrative story and the chosen objects that embodies important triggers and events. BSP uses a Geographical Information System[3] (GIS) to make it possible to distinguish certain kinds of objects. GIS is a system that categorizes map objects; the categorization is of the kind that it tells that a square on a map is a house or a church, etc. In combination with maps it is possible to use GPS for determining the position of the player, but together with the input from a sensor module[4] (in figure 1.1.2 referred to as directions) it is also possible to furthermore determine where the player is located with respect to game object location. Triangulating the positions from the GPS with the directions acquired from the Microstrain sensor module is graphically explained in figure 1.2. Side 1 in an imagined triangle is between GPS position 1 and GPS position 2 while side 2 and side 3 are between point P and the GPS position 1 and point P and GPS position 2. In the figure it is shown that the player is aiming towards object 2, or more precisely has pinpointed the location P.

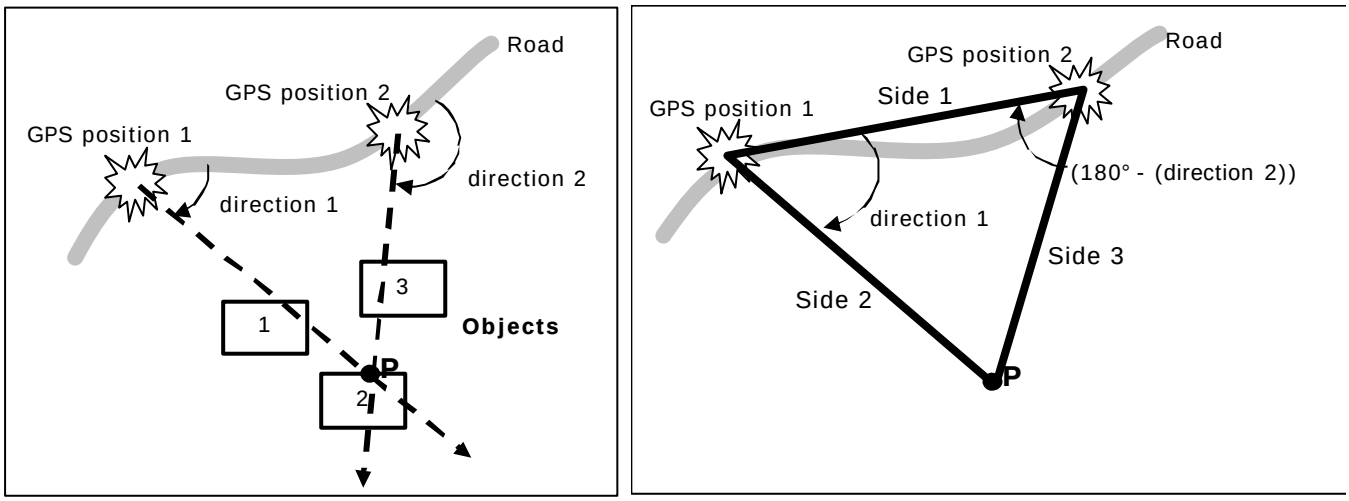


Figure 1.1.2; Left: Triangulation using two GPS positions and the angles acquired from the gyro. Right: The resulting imagined triangle.

1.2 Problem description

By using the information from the previous chapter there are a two significant facts that are implied.

[2] To design for physical interaction has earlier been investigated by Brunnberg and Hulterström (2003) in their article "Designing for physical interaction and contingent encounters in a mobile gaming situation".
[3] Geographical Information System is a system to provide for more information about map objects. More information could be found at: http://www.gis.com/whatisgis/index.html.
[4] The sensor module with its accelerometers, gyros and magnetometers is thoroughly explained in chapter 3.1.

1. Firstly, since there has to be at least two GPS positions the player has to be moving (i.e. if the player is not moving there will not be a side in the triangle between GPS position 1 and GPS position 2). This has to be greatly emphasized since it is of great importance for triangulation purposes.

2. The second fact is a bit more complex to explain. To do so one need to understand how BSP has been using the term "pinpointing" objects up to current date. The BSP game is using a quite rudimentary and non-realistic kind of pinpointing. This kind of pinpointing is first of all triggered by an already set object in the surrounding environment and then the user is pinpointing the object through the use of a 3D sound. In this way the pinpointing becomes more of an interactive approach to pinpoint an already set object, making it restricted through a close link to geographically set objects. The reason for this is that the player is aiming with the help of a sound mapped by the game to the specific location that the player should pinpoint. (E.g. the intensity of that sound is decreased the further away from the object the player is aiming and concurrently the sound increases in intensity when the player is aiming closer towards it.) This old kind of pinpointing is further on called Closed-Set Pinpointing (CSP) because of its limited range of available objects to pinpoint.

What this master thesis does instead, is to examine whether there is a possibility that a more realistic kind of pinpointing could be done. More realistic in the sense that the player experiences the physically direction he or she points the gaming device in as the direction perceived by the game as a location where something useful or interesting is taking place. Using CSP the game would ignore that the player is pinpointing a specific location where there has not been an object placed by the game.

Constructing a new generation of pinpointing would in contrast to CSP result in an increased gaming experience since the player feels like he or she is in control of the game and not the opposite. This kind of thought is applicable for a future possibility to develop BSP for User Content Creation (i.e. when users apply their own content in a game). Because of this increased experience the player would feel the game to be more realistic. Also the new generation widens the set of objects that could be pinpointed. Thereby the player could aim for objects that are not actually a part of GIS- or map information collected for the BSP prototype, the objects are rather based on the players own interest of the roadside and therefore the set of objects increases. Another major advantage of this approach is that the game could be further developed in an area that allows the user to create his or her own content (i.e. User Content Creation, UCC). Because the new kind of pinpointing allows for new objects to be generated in a realistic kind of way it is called Open-Set REalistic Pinpointing (OSREP).

Moreover development of OSREP is divided into two separate parts. The first part is to actually examine an algorithm that could sense that the player had started to aim towards a location by his or her own choice. The algorithm that should sense this was constructed in this thesis and serves as a proof of concept. If there are not any satisfying results achieved trough mathematical computations the final approach is to consider whether the realism could be satisfactory using another input. This kind of input could be the use of a button to signal that the player has started to aim towards an object or an interesting location.

The second part follows consecutively by doing some triangulation; with the help of the GPS positions and the gyro angles, during the time span that the player is aiming[5]. The result of the second part is then a triangulated location in the real world. That location could be used

[5] This is in a way real-time, but some constraints are put into this thesis and will be further explained in the theory chapter under the heading "Real-time". (I.e. the goal is to make the algorithm's performance as good as it would seem like it is in real-time.)

to compare if the location of an object in the BSP game database is present or if that is not the case the player could place user created content at that specific location. The theories for these two parts are presented in chapter 4 while the methodology for how it all was done is presented in chapter 3.

In the second part it would be sufficient to develop an algorithm to do the triangulation in real-time and leave out the comparisons with BSP object database. The major reason for this is to be able to build a stand-alone module that serves as a proof of concept, which allows for future development and integration by the team that develops the BSP game engine. My motivation for making this assumption is that the area of UCC is an ongoing study in the BSP prototype.

1.3 Purpose

The purpose of this master thesis project is to examine if a new generation of a pinpointing algorithm could be made for a mobile AR game, such as the BSP game prototype. This entails constructing a new generation of a pinpointing algorithm, called OSREP, which is more realistic in its use and allows for future development and integration of User Content Creation tools. The algorithms for this is to be constructed as proofs of concept showing that it is possible to carry out the necessary calculations needed to produce the results wanted for the BSP game prototype. To make the definition of “possible to carry out” is not as much the mathematical approach as to actually doing the mathematical calculation within the time requirements. The requirements of performance in time will be retrieved during test runs made. They will show the maximal time that the algorithm could take to perform its designated assignment.

The examination will be carried out by first study the relevant publications made in the area of mobile AR games and also study the publications made regarding the BSP prototype. Consecutively there has to be some studies in the areas of mathematical statistics, geometry and probability to show a significantly good theory for the approach of to the solution.

Another goal of this master thesis is also to visualize OSREP when it has processed acquired data from real world test runs. The visualization shows that OSREP is applicable on data identical to data that the BSP game uses. To acquire data test runs are carried out on Lidingö, a small island north east of Stockholm. This island has been chosen because the BSP test runs have been done there previously, therefore the data from my test runs are applicable to the BSP prototype. Another reason is that GIS data is available for that area.

1.4 Delimitations

- One of the things that will not be a part of this master thesis is the mentioned future development of the Backseat Playground game. That is to not further develop any User Content Creation tools.

- Full integration of the algorithms with the BSP game is not a part of the thesis project. The algorithms serve as proofs of concepts and are presented as modules that are applicable to BSP (with some slight modifications). This implies that the errors that could be experienced in the GPS positions will not be a part of this thesis, although they will be explained so that the reader is fully aware of their existence. The reason for this is that the corrections for the errors should be made on the server side of the BSP prototype, as further explained in section 3.1.1. Also, the integration or more specifically the calculated (triangulated) locations with the OSREP algorithms will not implement any comparisons with the generated BSP game objects.

- Time will be a major factor for being able to do much of integration, testing and further development between the algorithm and the rest of the game. The only thing to do is to try to reach the goals set and from there on do as much as possible.

1.5 Related Work

Since the basis of this master thesis is related to GIS objects and the area of mobile augmented-reality games there is not much related work to be found that combines those two areas with aiming and pinpointing GIS objects in mobile augmented-reality games. As pointed out in Gustafsson et al. (2006) there has been some work done in the areas of location-based games and location-based storytelling. But the major difference with a research prototype like *Treasure*, presented in an article by Barkhuus, Chalmers, Tennent, Hall, Marek Bell, Sherwood and Brown (2005), which is played by two teams running around in a physical environment while chasing coins in the vicinity of WiFi access points, and Backseat Playground is the extensive significance of physical movement. In BSP the player needs to move himself in the real world to make progression in the story of the game while the content in a *Treasure* stays much the same during gameplay. The major consequence of being an offspring of BSP is that it affects this thesis by being in the cutting edge of game researching. This fact affects this master thesis both negatively and positively, negatively in the way that there is not much information to gather in this area beside the publications made by the developers of the BSP game prototype. It has a positively affect on this thesis because it makes me, the author, more motivated since I can participate with valuable results taking the BSP prototype to the next level.

The closest I consider any game control of any game consol to be, is that of the Nintendo Wii Remote and Nunchuk[6]. The Wii Remote and the Nunchuk are both motion sensitive but are originally only used indoors and most definitely not used together with a pervasive storytelling game like BSP as described in an article about pervasive storytelling in vast location based games by Bichard, Brunnberg, Combetto, Gustafsson and Juhlin (2006) which also further describes the further uses of GIS to make further connections to the real world. Also, there is the obvious fact that the BSP gaming device is used together with spatial movement of a car which is the factor that makes it possible to triangulate positions in the surrounding environment.

In parallel to this thesis project there is research in progress regarding change of platform for the BSP game. Instead of a complex gaming device the intent is to use an ordinary mobile phone with image recognition to determine direction. This will of course only be possible in full extent in a couple of years when all phones contain a GPS receiver to be able to determine the position of the player. Thereof this master thesis will be of even greater interest in the future.

[6] An overview of the Nintendo Wii: http://www.nintendo.com/overviewwii, offers a short description of their motion sensitive game controllers.

2. Methodology

The methodology chapter tells you what approach has been made regarding how to examine if a new generation of pinpointing could be done. As mentioned in chapter 1.2 Problem Description, the major approach to this investigative project is to divide the assignment into two parts. To make the course of action understandable I have created a model that clearly describes in which order things are done and why I have chosen to execute them in this way. This customized model states three different parts that will be done to clearly visualize the sequence of work and the results which allows for a discussion and some conclusions to be made about them. More detailed descriptions of each part of the model are found in the implementation chapter.

2.1 Customizing a Model

Since this thesis project evaluates the possibility to extend an already existing module, or as mentioned derive a new generation of pinpointing (i.e. namely the OSREP) this project needs to have a quite customized methodology. But there are some basic steps to take before embarking on the model description. These basic steps are the following two: to gather knowledge about the BSP prototype and consider what requirements have to be met for the results of this thesis project to be usable in the future. After this has been done the model (in figure 2.1.1) is considered to be the model for the sequence of work. This model has been created to get an overview of what is done in this project and it constitutes the methodology. First of all it is a model dividing the project into three parts and taking the parts one at the time we get:

1. The first part is to create a simplistic tool for making the initial acquirement of empirical data. This tool preparation part is to be done in a systematic way to make it easy to collect the necessary data mainly from the GPS- and gyro modules respectively.

2. From the empirical data mathematical theories are deductively (i.e. the logical method used[7]) derived. In this way a sufficiently good algorithm can be presented. A sufficiently good algorithm is an algorithm that is able to show performance differences in realism between running the BSP game with the derived algorithm and running the BSP game without the algorithm. The OSREP algorithms are developed as modularized[8] applications that are able to process the collected set of empirical data (i.e. the data from part 1 of the model). The output data from runs on the empirical data from part 1 processed through the algorithm should then be able to show how plausible it is that the algorithm serves the purpose of OSREP regarding realism.

[7] A logical deduction method is when you look at e.g. a set of data and make hypotheses based on some common distinguishable trademark for all the values of that set. Concurrently you say that the values appear this way because of this trademark, you have then deductive hypothesis as to the importance of that trademark.

[8] Modularized applications are highly desirable, as Ghezzi, Jazayeri and Mandrioli (1991) points out in their book Fundamentals of Software Engineering.

3. The third part will be to show through a sufficiently good visualization of the algorithm that it holds or not holds for the theories presented. A sufficiently good visualization is to be able to see the performance of the algorithm as to whether it shows that the player is aiming or not and also show a calculated location through triangulation. If not there are analysis made to why the algorithm derived is not sufficiently good and what could have been done instead.

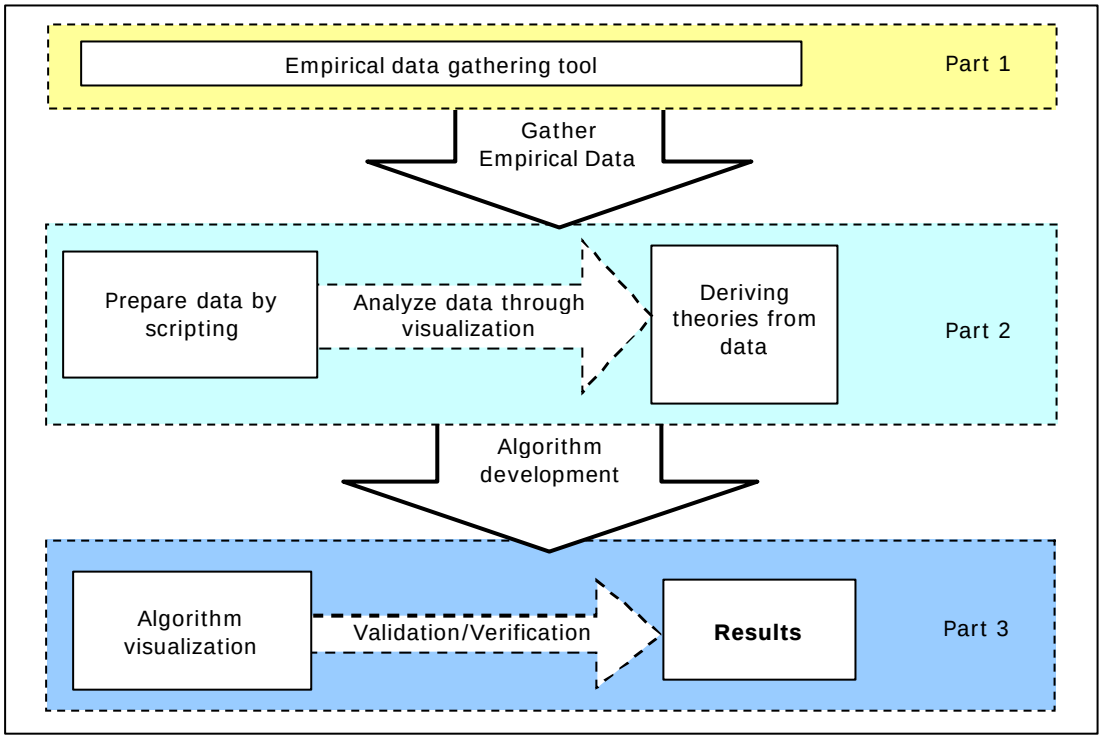


Figure 2.1.1: Structural model of execution of the master thesis project. Progression is made by moving from left to right for each part and downwards when a part is finished.

The final visualization of the verification and validation part was simplified due to time constraints put on this master thesis. It will certainly suffice to show convergent results (using simple graphs of the result) without a fancy interface with GIS objects and maps. This argument holds if the theories behind the algorithm in a conclusive way show that the results could be applied to a majority of test runs. To do so in a simplistic and minimalist way is called modularization and is further explained by Ghezzi et al. in the “Modularization techniques” chapter in the book “Fundamentals of Software Engineering”, where it is furthermore explained that the information (i.e. in this case the handling of extracted data and the theories derived from it) should be offered in a minimalist way to avoid unnecessary complexity of the design. Although since the handling of the data is hidden in the implementation of the design and therefore it is not completely avoided. It is hidden in the code developed for this thesis but have been well-documented for others to be able to continue the work with OSREP or to develop future generations of pinpointing applications.

3. Hardware

This chapter describes the hardware needed to play BSP. Why I describe the hardware of BSP is because of the fact that OSREP needs some parts of it to be able to acquire necessary data. The necessary data is the GPS positions and the gyro angles for direction when doing the triangulation.

3.1 Backseat Playground Hardware

To be able to play the Backseat Playground game in its current prototype version the user needs to have some hardware. This hardware (shown in figure 3.1.1) includes a server running on a laptop, a client (the gaming device), a Bluetooth GPS receiver, a wireless access point and a 12V-220V power converter.

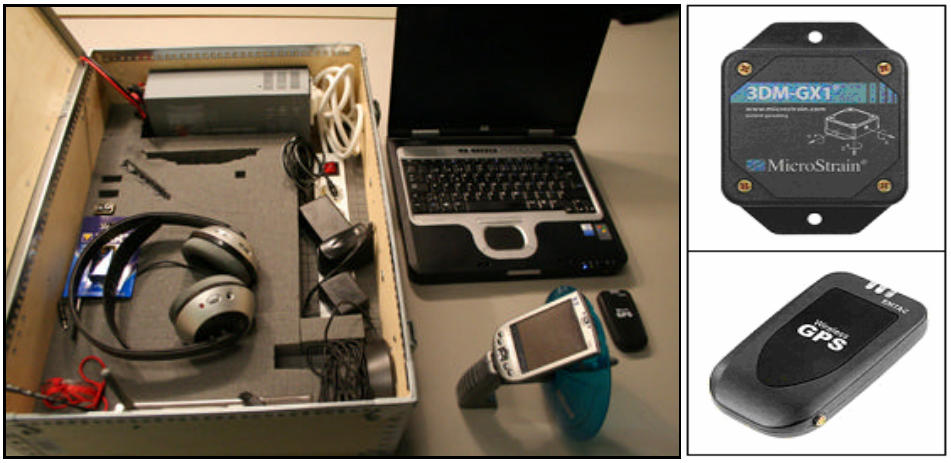


Figure 3.1.1: Hardware needed for the Backseat Playground game. The gyro module is seen in the top right image and the GPS receiver is seen in the lower right image.

The gaming device contains a directional microphone, the Pocket PC (also called PDA) and the headphones, while the rest of the devices could be installed at suitable locations in the car. The directional microphone is equipped with a Microstrain 3DM GX1 sensor module[9] which includes accelerometers, gyros and magnetometers to be able to sense orientation and motion in three dimensions. The data from this gyro sensor module is transmitted from the directional microphone to the Pocket PC through a cable connection. The Pocket PC uses wireless LAN to communicate with the server and the Bluetooth GPS receiver (EMTAC Bluetooth GPS) could be connected to either the Pocket PC or the laptop. The GPS signal is relayed over the wireless LAN in either case. The Pocket PC provides both the visual interface through its display and the audio interface through the connected headphones.

[9] This sensor module is further technically described in chapter 3.1.2 and in the documents at Microstrains web-site: http://www.microstrain.com/3dm-gx1_docs.aspx

3.1.1 *GPS*

This sub-chapter describes the GPS but not as much as it tells the reader more about the errors that are present when dealing with GPS positioning. The obvious motivation for this is to make the reader aware of the error factors that affect calculations and to show which factor that has the greatest impact.

The GPS works similar to the triangulation that was mentioned earlier. This is done by the GPS module itself and produces a converted result in the form of latitude and longitude. In this case of triangulation, called dilateration (in case of two satellites, trilateration if three satellites are used), the calculation is done between the GPS module that the player is carrying with him and two satellites. In reality several satellites are used to produce more accurate result and to determine the elevation of the GPS module, which is the height over the sea level. But in a simple example using two satellites; the time it takes for each signal from respective satellite constitutes the distance to a Point-Of-Intersection (POI) for which we know ought to be on the surface of the earth. From that POI the latitude and longitude of the GPS module is extracted. This is shown in a very simplistic way in figure 3.1.1.1 which has been retrieved from the kowoma web-site[10], in which point A is the valid POI. The shadowed area is considered to be the area of valid POIs since point B is far out in space.

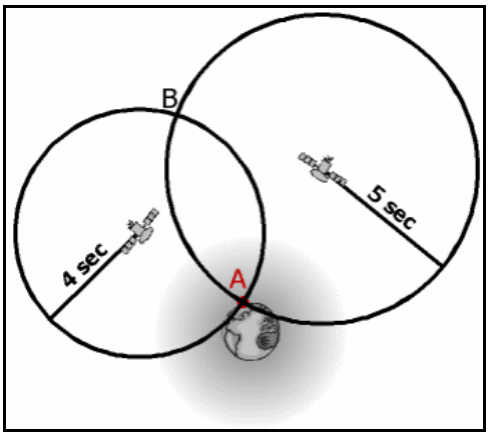


Figure 3.1.1.1: Acquiring a GPS position.

As seen in table 3.1.1.2 the largest contributor to positioning errors are the ionospheric effects. This information have been gathered from two independent web-sites[11] thereof the intervals in the parenthesizes. But regarding the ionospheric errors an assumption is made that could be argued to be very logic.

Ionospheric effects	± 5.0 meter
Shifts in satellite orbits	± 2.5 meter
Satellite clock errors	± (1.5 - 2.0) meter
Multipath effect	± (0.6 – 1.0) meter
Tropospheric effects	± 0.5 meter
Receiver errors (calculation and rounding errors)	± (0.3 – 1.0) meter

Table 3.1.1.2: This table shows the sources of error for the GPS positions.

[10] The figure has been retrieved from http://www.kowoma.de/en/gps/positioning.htm.
[11] The web-sites are: http://www.hacking-gps.com/articles.php?url=2&id=200503281930 and http://www.kowoma.de/en/gps/errors.

Since the ionospheric errors depends on the weather it is most logical to say that this kind of error would not affect single GPS positions it would rather affect the whole set of GPS positions received at the same time of day (or when the weather is the same) making the complete position set usable anyway. This is valid because distances and speeds received from the GPS module are all equally affected by the same error cause, making the GPS positions in figure 1.1.2 not differ because they are received with only one second apart. I.e. GPS positions received in one of my test runs are not affected by the ionospheric (or shifts in satellite orbits and satellite clock errors for that matter) since those effects changes to slow. Moreover satellite clock errors are by listening to Samuel J. Wormley, who administrates a quite extensive web-site[12] about GPS errors, also slow in consideration of how long my own test runs have been performed (maximal length of one test run was 40 seconds). He speaks of an average error of 1-2 meters for 12 hour updates. Considering these time spans I argue for a non-significant change between GPS positions in my test runs.

As I already mentioned in the delimitation chapter, I do not correct these kinds of errors nor do I implement a solution for this kind of problems. It could be argued as making a non-sufficient algorithm for the triangulation purpose.

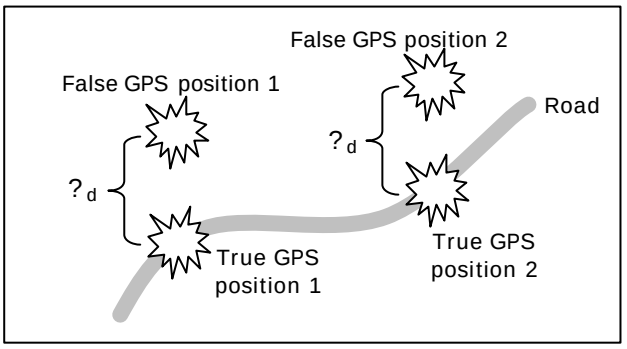


Figure 3.1.1.3: Scenario with GPS position errors.

To try to make up for that I hereby explain how the solution were the correction would take place on the server-side of the BSP prototype could work like. This solution imply that the corrections would compare known GIS objects on a map to make the shift of the GPS positions (the shift to make being $?_d$; as shown in figure 3.1.1.3). That the correction would need GIS objects explain why it should be done on the server-side of the BSP prototype (i.e. the GIS software module in BSP is located in the server-side). The shift being applied to every received GPS position; in reality, does not make a difference for any distance calculations between any two GPS position since it is a simple translation of the origin for a local coordinate system in a global coordinate system.

This is simply shown by using a translation matrix[13];

$$\begin{bmatrix} 1 & 0 & \Delta_{latitude} \\ 0 & 1 & \Delta_{longitude} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} latitude \\ longitude \\ 1 \end{bmatrix} = \begin{bmatrix} latitude + \Delta_{latitude} \\ longitude + \Delta_{longitude} \\ 1 \end{bmatrix} \qquad \text{(Formula 3.1.1.4)}$$

where

[12] The accuracy section of the web-site of Samuel J. Wormley could be found here: http://edu-observatory.org/gps/gps_accuracy.html.
[13] The formula for this has been found on p. 159 in “Modern Geometry” by David A. Thomas.

$$\begin{bmatrix} 1 & 0 & \Delta_{latitude} \\ 0 & 1 & \Delta_{longitude} \\ 0 & 0 & 1 \end{bmatrix} = \Delta_d \qquad \text{(Formula 3.1.1.5)}$$

is as explained a constant matrix of the distance difference (caused by the errors for a sufficiently discretisized time span) used for every GPS position in the finite set that is to be translated.

3.1.2 *Accelerometers, Gyros and Magnetometers*

The accelerometer, gyros and magnetometers described in this sub-chapter are all contained in the Microstrain 3DM GX1 sensor module. Although this sensor module contains three kinds of sensors they are all referred to as the gyro sensor. The reason for this becomes evident when reading through this section.

The accelerometer in the Microstrain sensor module is a triaxial DC accelerometer that tracks static orientation. This means that it senses the accelerations in 3D and the values of the acceleration is presented as the magnitude of the acceleration along the x-, y- and z-axis (i.e. how fast the gaming device is moved along either axis). These three axes are shown in figure 3.1.2.1.

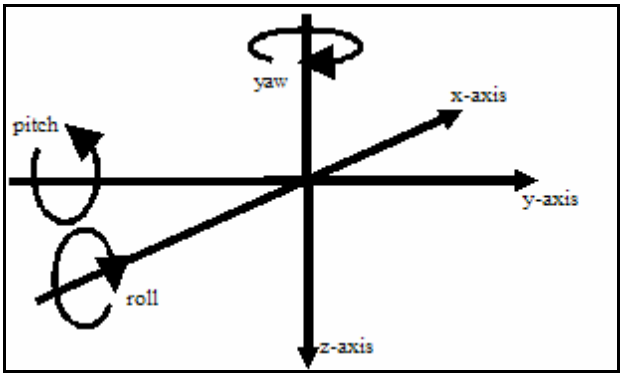


Figure 3.1.2.1: Shows the Microstrain sensor module's coordinate system.

The magnetometers in the Microstrain sensor module are triaxial and give the absolute direction of the magnetic field in relation to the gaming device. These magnetometers offer great precision and by using its values the direction of the gaming device with regard to the earth's magnetic field is acquired. Calculation applied to values from two separate times from this sensor shows towards which direction the gaming device is moved. Further calculation could with this type of sensor give how the gaming device has been rotated.

The final sensor is the gyro (also called gyroscope), which determines how the device is rotated around each axis; namely the rotations roll, pitch and yaw. The explanation for how this is done is rather complex therefore the conceptual thought of how it works is presented[14]. The gyro as it is in the Microstrain sensor module is considered to be three gyros mounted with their axes at right angles to one another on a platform, that platform is placed inside a set of gimbals, that platform will remain rigid as the gimbals rotate in anyway they

[14] A conceptual figure is shown in figure 3.1.2.2 and the explanation of how the gyro works has been found at: http://www.howstuffworks.com/gyroscope.htm and http://www.gyroscopes.org/how.asp. Although these two web-sites are considered to be somewhat popular scientific, combined they give a conceptual idea of the most significant traits of a gyro in an inertial navigation system (INS). Since these three web-sites show much the same thing it is considered to be a universal description of how a gyroscope works in an INS.

please. Then with sensors on the gimbals' axles it is detected when the platform rotates. These sensors then produce signals that the INS uses to understand the gaming device's rotations relative to the platform. The sensors have of course been placed on the platform and consist of the earlier mentioned accelerometer. The exact heading and the motion in all three directions and consecutively the rate of angle around either axis could then be retrieved.

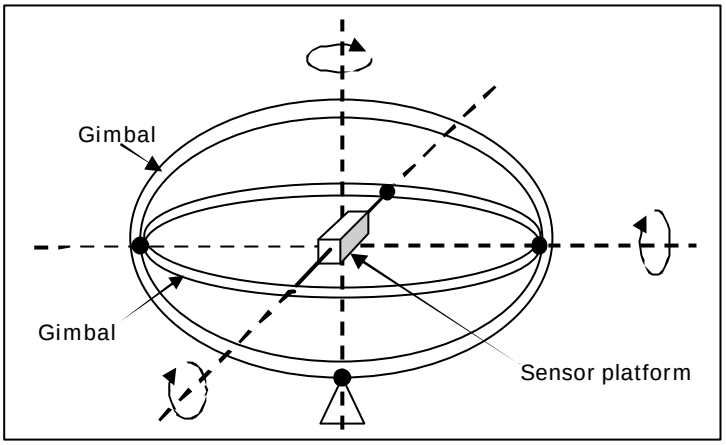


Figure 3.1.2.2: Shows the gimbals and how the sensor platform is placed in a gyro.

Of course there are sources of error in the Microstrain sensor module as well. The theoretical approach to how these would be corrected is briefly mentioned in the end of this section. The issues with pinpointing objects arise because the Microstrain sensor module that records the angular movements of the gaming device will contain errors in the recorded sensor data. In the most general case without considering any errors it could look like figure 3.1.2.3 (which is similar to figure 1.1.2) where the player is trying to pinpoint object 2. The figure shows where the car is positioned and how the player is aiming. Since the gyro records the angles and the GPS records the positions, the intersection point P that the player is aiming for can be calculated, earlier mentioned as triangulating.

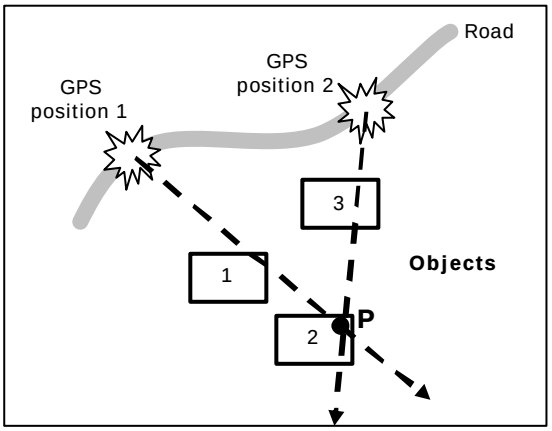


Figure 3.1.2.3: In this scenario it seems like the player is tracking object 2.

But in the case with direction errors in the recorded angles the scenario looks rather like in figure 3.1.2.4 and requires a much more complex solution. The greatest difficulty is to determine where the intersection point P really is located and at the same time determine if the player is actually pinpointing anything (e.g. the player could just be waving the gaming device). As we see in figure 3.1.2.4 it looks like the player is tracking object 1 instead of 2 because of the angle errors ? in the gyro. The gyroscopically errors occurs because of temperature changes, vibrations and noise in the signal amplifiers in the Microstrain module. This causes the gyro to drift and to compensate the magnetometers calibrates the gyro according to its own state (e.g. the magnetometers tries to calibrate the gyro with respect to

its own registered direction). But since the magnetometers are affected by the magnetic fields of the car this calibration could be erroneous. As seen in the figure 3.1.2.4 there are two arrows for each GPS position. The dashed arrows marked show the angular error caused by the drift. In a sufficiently discretisized time space the error angle ? will be the same for both positions which simplifies the error model needed to solve the problem.

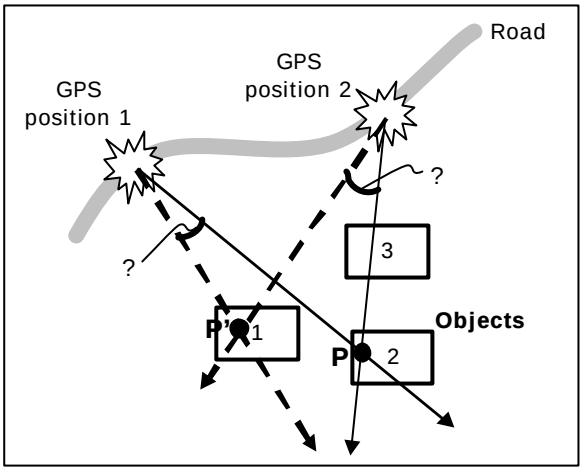


Figure 3.1.2.4: In this scenario the concept of the Microstrain sensor errors is shown.

3.2 OSREP Hardware

The OSREP hardware has of course been greatly influenced by BSP since OSREP is an extension of the CSP in the BSP prototype. But there are some differences. First of all no sound is needed to gather GPS and directional data. Neither is there any use of a nice looking graphical interface which makes the exclusion of the BSP software on the PDA an obvious fact[15]. The software is further explained in chapter 3.3.

The Bluetooth GPS receiver that the BSP prototype uses will also be used when gathering GPS positions during the test runs. The update frequency of the Bluetooth GPS receiver is approximately one set of values every second (i.e. frequency equals ~1 Hz). To gather the direction data for the gaming device the Microstrain sensor module is required. This will emit sets of direction data approximately every 1/70 second (70 Hz). Furthermore, the GPS receiver outputs latitude and longitude. The latitude and longitude are therefore converted to ordinary x- and y coordinates for easier calculations.

To list some other hardware needs; there is the fact that the PDA could run low on batteries and therefore the power converter could be a nice addition when conducting test runs. Another fact is the need of a laptop since the development of OSREP algorithms requires compilers etc. to develop applications. But since there are not to be any client-server application made, there is no need for having a wireless AP while conducting the test runs.

[15] This is a significant fact that greatly affects part 1 of the customized model.

4. The OSREP Theories

Explained in the problem description in chapter 1 the major approach for this thesis is to divide the new generation of pinpointing called OSREP into two parts. The first is to determine when the player is aiming and the second is to determine where in the real world the player is aiming. These two parts are as told presented here as the new OSREP algorithms. In this chapter definitions and mathematical theories are presented and in a deductive manner they present final theories for the two algorithm parts of the OSREP that is used for implementation.

To start things off, some definitions has to be made as to in which kind of situations the OSREP algorithms should work and also some physical/spatial logical reasoning that is used in the theories.

4.1 Definitions

To use logic reason is very efficient in many situations. The logic reasoning and calculations presented here severely decreases the complexity of the OSREP algorithms. To reduce the complexity there are definitions and investigations made as for road types and the spatiality of how a player pinpoints objects.

4.1.1 *Investigating valid road segment types*

When considering for this project a valid OSREP situation the figure 4.1.1.1 could be a good representation of how the roads traveled on could look like. The types of roads that are investigated in this thesis are the classic S-curve, a strong curve, a less strong curve and finally a straight road. The reason for not investigating roads with more curves than an S-curve is on the one hand that the time between two GPS positions are approximately only one second and on the other hand that the speed limits and constructions of the roads in Sweden are highly regulated to ensure high traffic safety. These reasons will become more evident when reading this sub-chapter.

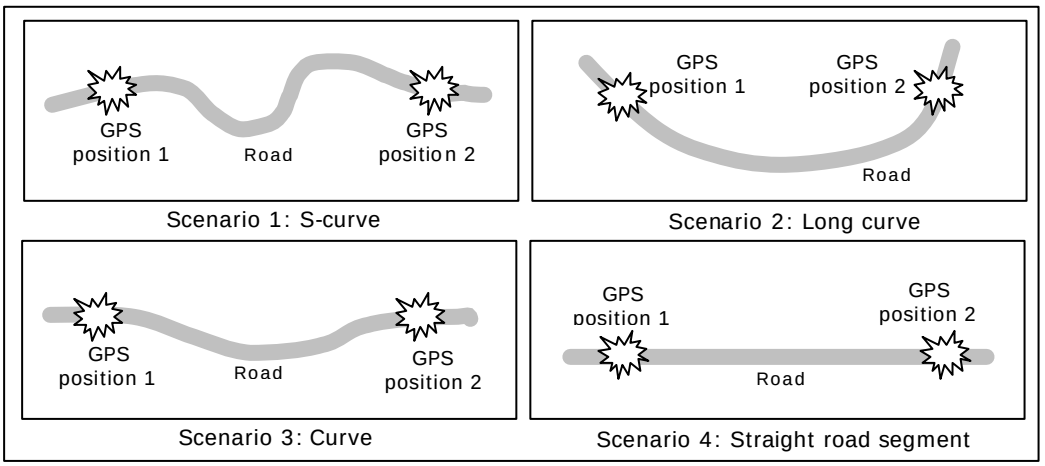


Figure 4.1.1.1: Road scenarios.

From this figure there are some significant conclusions to be drawn but also there are some facts that become obvious when considering real world objects (e.g. the geometry of a car, width of a road, etc.). Combining those facts with some physical properties like how far a car can get in one second (the time between two GPS positions) the complexity is reduced. Firstly, the distance that a car can travel is dependent on the speed of the car. When I did my test runs a speed of approximately 50 km/h was held (due to speed limit). But since I want to do the forthcoming calculations for several speeds the table 4.1.1.2 shows the distances depending on speed while the time interval is considered to be one second long.

Speed [km/h]	Distance [m]
30	8.33
50	13.89
70	19.44
90	25
110	30.56

Table 4.1.1.2: Distance traveled at a certain speed during one second.

These distances show how long the road segment between two GPS positions could possibly be while traveling at the different speeds. But using logic reasoning it is an obvious fact that scenario 1 (an S-curve) cannot occur for any speeds since the distances are too short. At least the distances are too short for a car (e.g. a normal sized car is assumed to be about 4.5 meters in length) to be able to pass through so many curves in only one second at those speeds. Although when given these distances and considering the standards that the Swedish Road Administration (called VV, because they are known as Vägverket in Sweden) uses to construct roads the calculations in Appendix B could be done. Those calculations are done using the distances in table 4.1.1.2 as the sector of a circle in the formula for a sector of a circle[16] together with the standards from VV for minimal radiuses of a horizontal curve[17] as the radius of the circle. Table 4.1.1.3 shows the results from those calculations.

Speed [km/h]	Radius [m]	Calculated Angle [degrees]
30	20	23.86
50	90	8.84
70	200	5.57
90	400	3.58
110	600	2.92

Table 4.1.1.3: Angles extracted from the calculation in Appendix B scenarios.

Through this it could be argued that the angle (i.e. how narrow a curve can be) is very small which concludes that there could not be any sharp curves passed while traveling at the considered speed limits. Therefore the kind of road segment (between two GPS positions) for which the forthcoming theories are to be presented for is only straight ones. I.e. the road segments for which positions and directions are calculated for are discretisized as being straight lines. For the reader to grasp somewhat how this could look like, figure 4.1.1.4 gives an approximate straight line road segment (of length 8.09 meters). Since the length of the circle sector is 8.33 meters (the distance in table 4.1.1.2) not much of a curvature could be present. For the higher speeds there would not be any differences since the radius becomes exponentially larger while the length of the circle sector increases linearly (compare distance in table 4.1.1.2 with the radius in table 4.1.1.3). These investigations offer a severe decrease

[16] See Appendix A for reference to how such a calculation is done.

[17] Using a "Low" standard of the road along with the highest percent of askew, this allows for a shorter radius which implies sharper curves. More information could be seen on page 60 in the Linjeföring document (in Swedish) at: http://www.vv.se/templates/page3___8165.aspx.

of complexity regarding the OSREP algorithms since it is easier to do straight line computations.

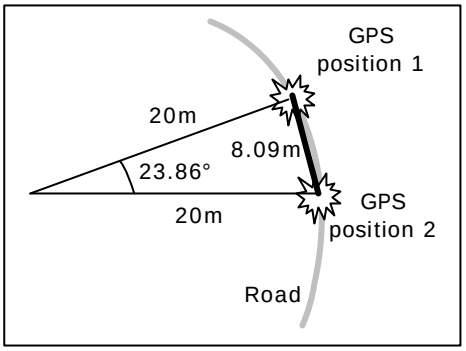


Figure 4.1.1.4: Approximate road segment given the calculations made.

Now the reader might think; what if the GPS is not as good that it continuously would output positions every second, I think I need to state that it happens rarely. Experience gathered from test runs along with what BSP developers say; the GPS positions have extremely seldom been received with more than one second apart. Based on the data gathered from more than 50 test runs I argue that it happens so rarely that it is close to a non-existent case. Furthermore that the player would aim during such a case is thereby even more unlikely to happen. But it could happen; for instance in a tunnel there would not be a GPS position acquired, but in that case the sight range (i.e. the range that the player is assumed to have) would be significantly less and perhaps even non-existent. In addition to this there are new GPS receivers that are even better than the one used during this thesis project when considering update frequencies.

4.1.2 *How does a player pinpoint an object*

In this sub-section I clarify what I have considered to be a valid way of pinpointing objects, valid in the sense that when the forthcoming algorithms will be applied or not. The theory for how a player is pinpointing an object is dependent on two things; the speed of the car and the distance to the object in question. The distance to the object is regarded to be how far out from the road the object is laying.

Based on the experiences I have got during the acquirement of my own sensor data I believe that I have a great insight into how a player would behave while pinpointing an object. My behavior during my test runs are considered to be of interest since during those kinds of test runs the focus have laid on the actual pinpointing of objects. The focus on the PDA only occurs when initializing a test run (i.e. starting a recording) making it possible to stay focused on the object that is to be pinpointed. This is achieved due to the fact that I did design the tpPPC[18] for blended focus of attention[19]. This means that I as a player can focus on the things happening outside of the window instead of focusing on what happens on the screen of the PDA. I.e. there is no need for me to blend my focus of attention with anything else but to pinpoint an object on the roadside.

Going over to the more physical part of pinpointing figure 4.1.2.1 visualizes how the gaming device supposedly is spatially moved during an event of pinpointing. I argue that to make a valid attempt to pinpoint an object it is required that the player aims toward the object

[18] tpPPC is an acronym for thesis project Pocket PC. The data acquirement application is further described in chapter 5.1.

[19] Blended focus of attention (or to balance the focus of attention) is a term explained by Brunnberg and Juhlin (2006) as a way of designing; the blended attention in this way of design occurs when players engage in gameplay and interact with a computer in various ways.

before the car has passed it and keep pinpointing the object until sometime after the car has passed it. This is clarified by figure 4.1.2.1.

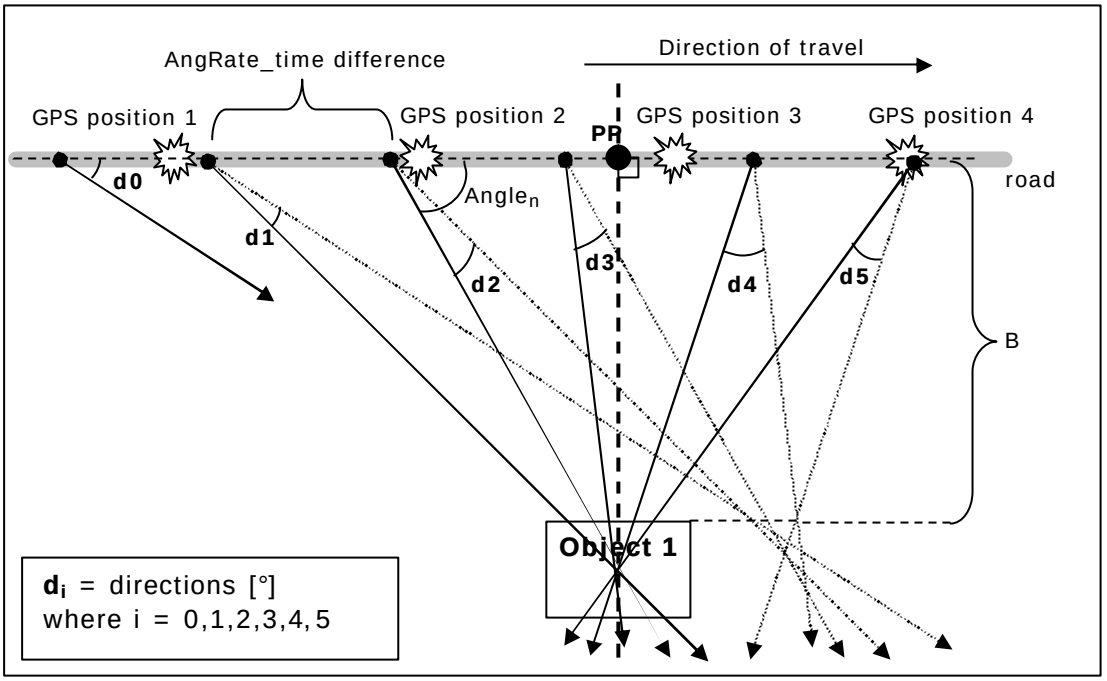


Figure 4.1.2.1: Showing how pinpointing should be done from the player's perspective

Although the case in the figure is an ideal case it clarifies what constitute as a valid attempt. The difference between a previous direction and the current is denoted as $\mathbf{d_i}$, this is retrieved from the AngRate output from the Microstrain sensor module. The AngRate is the rate at which the gaming device is rotated around the z-axis in radians/second. The direction or in other words the angle is a recursive result from calculating with the time of reception (in seconds) for the previous angle and the current angle rate (in radians/second) and its time of reception (in seconds) like in formula 4.1.2.2.

$$angle_n = \sum_{i=0}^{n} \left(AngRate_i * \left(AngRate\_time_i - AngRate\_time_{i-1} \right) \right) \qquad \text{(Formula 4.1.2.2)}$$

Since the angle depends on a time difference that actually is decided by the update frequency of the Microstrain sensor module; in a discrete case that time difference would be the same, since the sensor module emits sensor values continuously every seventieth of a second (but for two different modes, for mode 2 every 1/35 second and for mode 3 every 1/35 second). This means that formula 4.1.2.2 could be generalized to be:

$$angle_n = \frac{1}{35} \left(\sum_{i=0}^{n} AngRate_i \right) \qquad \text{(Formula 4.1.2.3)}$$

Furthermore the angle is increasing when approaching an object while it will decrease when the player has passed the object. This means that the absolute peak for the angle is 90 degrees; being the angle when the player is located at position PP in figure 4.1.2.1. The calculation of the maximum AngRate in degrees shown in Appendix B indicates that when the player is passing an object the angle depends on the velocity of the car, v and the distance B between road and object like in formula 4.1.2.4 (i.e. when calculating with a right triangle at the time when the car is passing the object).

$$angle_n = \cos^{-1}\left(\frac{\left(v\frac{1}{35}\right)}{\sqrt{B^2+\left(v\frac{1}{35}\right)^2}}\right) \qquad \text{(Formula 4.1.2.4)}$$

The above formula is of course for a single step in time for the directional angles. As seen in appendix B the general version of the formula; 1/35 is exchanged with time t which would be equivalent to the time difference in formula 4.1.2.2.

Another observation from my test runs is that I was keeping the gaming device quite still when not aiming towards anything. (Or in the other end if the player is sweeping the gaming device quickly from side to side the variance, described in chapter 4.2, of the angle would be too great and therefore I would assume that the player is not pinpointing anything.) But as soon as I saw an interesting object I quickly pointed the device towards it and tried to keep it as steady as possible. This could give a hint as to how the first algorithm of OSREP could sense that the player is trying to pinpoint an object. A long with this behavior while pinpointing I noticed that when I stopped aiming my body was pretty much twisted backwards since I tried to aim for the interesting object as long as it was in my line of sight. Therefore when I moved so that I sat up straight the gaming device tagged along with my movements. This created a great leap in degrees over a short period of time and the variance was greatly increased.

4.2 First algorithm of OSREP

The first algorithm of OSREP is the algorithm that in some way should sense when the player has started to pinpoint an object of choice. The most obvious way this could be done is through calculating the variance of the acquired AngRate values. This type of calculation shows how much each value differs from a mean value. The mean value is calculated through the formula 4.2.1 for n gathered values in an interval [a, b]. The need for a closed interval is to be able to define a current set of angle data to do computations on. Where n would have sufficed to be approximately the angle values during 200 / (speed of the car) seconds to distinguish them from other cases (such as a random constantly increasing AngRate). 7 seconds since it is half of the time it takes to pass an object if the player has a sight range of 200 meters and traveling at speed v (e.g. 200/(50km/h/3.6) = 14.4 seconds).

$$\overline{\text{angle}} = \frac{1}{n}\sum_{k=a}^{b}\text{angle}_k \qquad \text{(Formula 4.2.1)}$$

From this the variance is calculated by using formula 4.2.2:

$$s_A^2 = \sum_{i=1}^{n}(\text{angle}_i - \overline{\text{angle}})^2 \qquad \text{(Formula 4.2.2)}$$

As I mentioned earlier there was a phenomenon with a great variance in the end of each event of pinpointing is clearly seen in figure 4.2.1. This is considered to be a signal that the player has stopped aiming. But it will only be signaling this if and only if this algorithm has detected that the player has started to pinpoint (i.e. you can not stop anything that has not been started). As a security measure I also recorded the button states which were used as described in the “Conducting Test Runs” section of the implementation chapter. The button

states is set to 1 (one) if the button is pushed to signal that I have started to aim and when released it signals a 0 (zero).

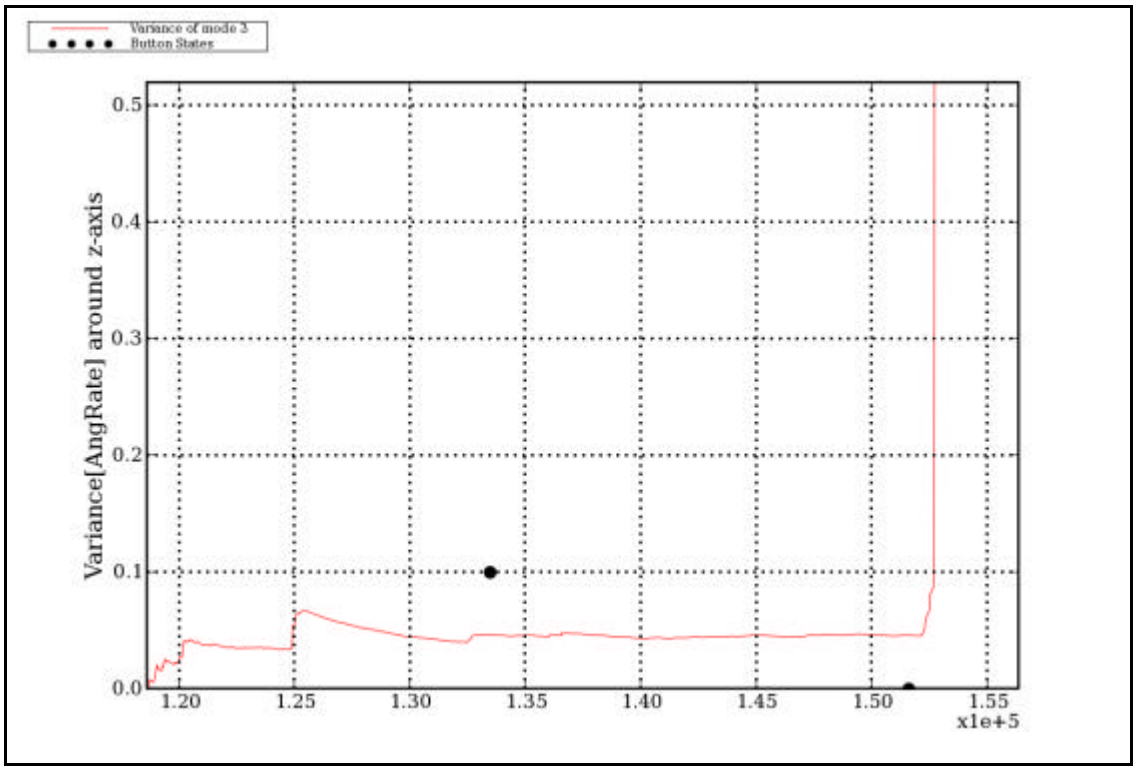


Figure 4.2.1: Example of how the variance looks like for the first algorithm of OSREP. This is from Test run 1.The dots are button pushes used as key references for comparisons between test run and computed values.

Although these did result in some nice graphs (e.g. an example is shown in the result chapter). There were some interesting discoveries made during the minor development-test cycle that lead me to believe that the importance of the first OSREP algorithm had been highly overrated. The proof of this is seen in the result chapter in the form of graphs computed from real world test runs. Along with that; Appendix D shows the feedback from running the code with a closed interval versus letting the second OSREP algorithm run without closing a set with the recorded button states which would correspond to letting the first OSREP algorithm (as far as it has been worked, seen above in this section) decide in which interval to run second OSREP algorithm. Also see figures 6.4.2.2 and 6.4.2.3 for graph results of these facts. Because of this development progress the theory of the first OSREP algorithm is not further pursued and reason for presenting it at all is for future development. It could be useful to have this kind of approach in mind. Someone else who is a lot better in the area of signal processing (e.g. probability theories) could do something nice future work, I think.

4.3 Second algorithm of OSREP

The foundation of the second algorithm of the OSREP algorithms is the data found in between the start and stop defined by the first OSREP algorithm. The angle of every direction data is calculated and further on the location in the world were the object lays is retrieved through triangulation by finding all line intersections.

To understand this it is necessary to know how the line intersections are calculated. In the next sub-chapter (4.3.1) the complete calculation is shown with its corresponding figures explaining it graphically. It shows that given two GPS positions (converted from the WGS84

format into the Swedish RT90 format[20]) and the direction data in between those, all intersection points for all direction line could be calculated. Following this an area of interest could be estimated and it is assumed that the center of this area is considered to be the location of the object that the player has tried to pinpoint.

4.3.1 *Line Equations and Line Intersections*

When two GPS positions has been received, all the directional data (i.e. the AngRate with timestamps between the timestamp of GPS position 1 and the timestamp of GPS position 2) could be used to derive the location of an object. Or more specifically this is done in the interval where start and stop are given by the timestamps of the result gathered from the first algorithm of the OSREP algorithms.

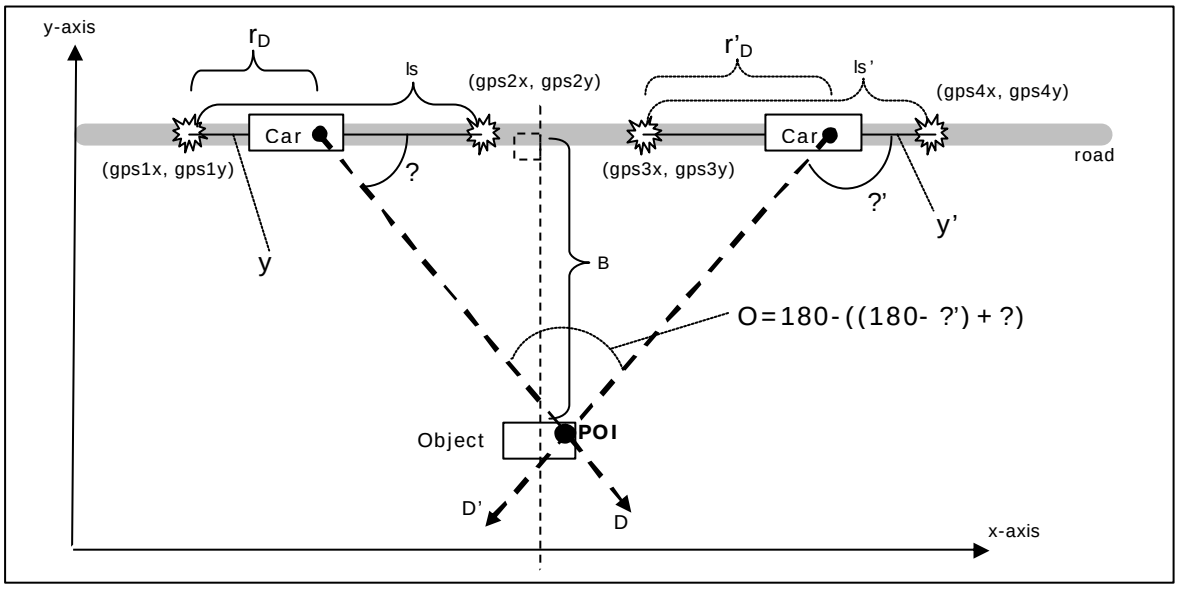


Figure 4.3.1.1: Line equations and line intersection.

From known data and as illustrated in figure 4.3.1.1 I derive the line going through GPS position 1 and GPS position 2 by using the formula for the equation of a line[21] when two points are known, the result is seen in formula 4.3.1.2.

$$y = \left(\frac{(x - x_1)}{x_2 - x_1} \right)(y_2 - y_1) + y_1 \qquad \text{(Formula 4.3.1.2)}$$

Where $x_1 \neq x_2$ and $y_1 \neq y_2$. From this the position of the car can be decided by taking the time stamp from the acquired direction data (D in the figure) to see how far between the GPS positions the player has traveled.

$$r_D = \left(\frac{\text{timestamp}_D - \text{timestamp}_{gps1}}{\text{timestamp}_{gps2} - \text{timestamp}_{gps1}} \right) \qquad \text{(Formula 4.3.1.3)}$$

[20] The transformation for doing this is shown for the Python Programming Language in Appendix C. A very brief description is that the WGS84 geodetic latitude and longitude is transformed into grid coordinates x and y through the use of a Direct projection available at the web-site of Lantmäteriet (http://www.lantmateriet.se).

[21] To get the equation of a line I used the formula on p. 133 in "Åtta kapitel om geometri" by Anders Tengstrand.

Where r_D is the progress the car has made between GPS position 1 and GPS position 2. To get the x- and y-coordinate for the origin of D the magnitude of ls has to be calculated. This is done by formula 4.3.1.4.

$$|ls| = \sqrt{(gps1_x - gps2_x)^2 + (gps1_y - gps2_y)^2} \qquad \text{(Formula 4.3.1.4)}$$

By multiplying the magnitude of ls with r_D the ratio (r_D*|ls|) / (|ls|) gives the point of origin (POO) for the direction D as shown in formula 4.3.1.5.
If $\left(s = \overline{ls} - r_D * \overline{ls}\right)$, then Formula 4.3.1.5 looks like:

$$POO_D = (D_x, D_y) = \left(\frac{\left(\left(r_D * \overline{ls}\right) * gps2_x + s * gps1_x\right)}{(r_D + s)} ; \frac{\left(\left(r_D * \overline{ls}\right) * gps2_y + s * gps1_y\right)}{(r_D + s)} \right)$$

This will help significantly when calculating every point of intersection (POI as in figure 4.3.1.1). But first the equation of the line for direction D has to be calculated which is done by using formula 4.3.1.6 which is found on p.80 in the "Mathematics Handbook for science and engineering" by Råde and Westergren.

$$y = (x - D_x)\tan q + D_y \qquad \text{(Formula 4.3.1.6)}$$

When x=0; formula 4.3.1.6 becomes the formula below (Formula 4.3.1.7):

$$y = D_y - D_x \tan q = \frac{\left(\left(r_D * \overline{ls}\right) * gps2_y + s * gps1_y\right)}{(r_D + s)} - \frac{\left(\left(r_D * \overline{ls}\right) * gps2_x + s * gps1_x\right)}{(r_D + s)} \tan q$$

Further on the equation of the direction line then becomes; by using formula 4.3.1.2 with ($x_1 = D_x$; $y_1 = D_y$) and ($x_2 = 0$; $y_2 = D_y - D_x * \tan ?$):

$$y = \left(\frac{(x - D_x)}{0 - D_x}\right)\left(\left(D_y - D_x \tan q\right) - D_y\right) + D_y$$

$$\Leftrightarrow y = \left(1 - \frac{x}{D_x}\right)\left(-D_x \tan q\right) + D_y$$

$$\Leftrightarrow y = x \tan q + D_y - D_x \tan q$$

$$y = x\tan q + \frac{\left(\left(r_D * \overline{ls}\right) * gps2_y + s * gps1_y\right)}{(r_D + s)} - \frac{\left(\left(r_D * \overline{ls}\right) * gps2_x + s * gps1_x\right)}{(r_D + s)} \tan q$$

The last row is formula 4.3.1.8 which shows the equation of any direction line between GPS positions 1 and 2. Thereof the intersection between directions D and D' is found where they have the same x and y coordinates. These coordinates is found through testing coordinates for both lines and comparing the result in the same cycle in a for-loop. This will also reduce redundant comparisons. If the maximum line of sight the player is able to see is assumed to be 200 meters (circle radius = 200 meters of obstacle free line of sight) the pseudo-code looks like in figure 4.3.1.9.

```
for(x = 0;x ≤ 200;x++){
```

$$y = \left(x\tan q + \frac{\left(\left(r_D * \overline{ls}\right) gps2_y + s * gps1_y\right)}{\left(r_D + s\right)} - \frac{\left(\left(r_D * \overline{ls}\right) gps2_x + s * gps1_x\right)}{\left(r_D + s\right)} \tan q \right);$$

$$y' = \left(x\tan q' + \frac{\left(\left(r_{D'} * \overline{ls'}\right) gps4_y + s' * gps3_y\right)}{\left(r_{D'} + s'\right)} - \frac{\left(\left(r_{D'} * \overline{ls'}\right) gps4_x + s' * gps3_x\right)}{\left(r_{D'} + s'\right)} \tan q' \right);$$

```
    if(y == y'){
            return(x,y);
     }
}
```

Figure 4.3.1.9: The pseudo-code for finding where two lines intersect.

Since the (x, y)-coordinate that is returned is the corresponding coordinates for the POI in the global coordinate system these values could be used directly to find the location in the real world. Although the whole set of these points will create a geometry which I have chosen as a circle where the radius is defined to be the center of the spanning area. To make the representational geometry more valuable the geometry is weighted by some factors that are reasonably logical to have some effect on how plausible each of the POI is.

4.3.2 *Weighing POIs to Achieve POI Convergence*

The first thing that affects how good the relation is between two POIs is how far away from each other they are. I assume that the largest object the player would consider as interesting enough to leave some information at would be an arena which could span up to perhaps 200 meters in length. Although keeping this master thesis simple I will instead consider the largest object to be an ordinary transformer housing of 9 meters in width. This of course corresponds well to what kind of objects I was aiming for during my test runs. By using this assumption there is the question of which of the two directions that would be considered to be the right one. This of course has to be dealt with by computing another direction line and if that is closer than 9 meters from any of the other two, those two will be left to be a part of the representative geometry.

Another thing that is done when weighing the POIs is to measure the concentration of each of them. To do so I decided to check every POI against every other POI in the same set. If they lay within an object tolerance of 9 meters (e.g. the 9 meters mentioned to be the largest object width) they get a hit. Then to only get the POI with the greatest hit count would be the most plausible POI and thereby the set of POIs have converged into one Point-Of-Intersection. Actually the POIs with greatest hit count and second greatest are retrieved do determine the object width.

4.3.3 *Initial Angle – The Offset*

As seen in figure 4.1.2.1 there is an initial angle d_0 that could be perceived as not belonging to this part of the OSREP algorithm. I.e. I said earlier that two GPS positions are needed and the d_0–angle is not a part of any GPS position interval. The reason for considering an offset angle is that the OSREP algorithm has to be flexible in the sense that it should be able to be put in use at any time in the BSP prototype. Thereof there are some further computations needed to determine in what situation that OSREP is put in use. It is here the offset angle presents itself as a part of the OSREP algorithm.

To be able to determine an offset that could be used for the rest of the pinpointing interval (from when the OSREP is applied to when it has detected an object) some performances are

evaluated. The result that presents the best performance in valid angles found and in most POIs found is considered to be the accurate offset.

An easier explanation for this is that I iterate the offset angle over an interval from 0 degree to 180 degrees. Then when the iteration is over and the results are evaluated for every iteration. Then the offset angle that produces the valid angles and the most POIs is chosen. A valid angle is an angle that results in y and y' in figure 4.3.1.9 being in the line of sight of the player and y-y' being not more than 9 meters from each other (e.g. the 9 meters are what is called the object tolerance, a resolution for how wide the target area of the pinpointing is).

5. The OSREP Implementations

This chapter entails how the different algorithms that OSREP consists of has been implemented beginning with the data acquirement tool. Mostly this chapter describes the code that has been written but much effort is put on making the chapter understandable, therefore there are more class-diagrams than there is actual code. Starting every sub-chapter is an overview that describes the software in general for each individual part of the customized model[22]. This results in revealing the implementation in a chronological order; starting with the tpPPC application and how the test runs were conducted. Then I am going over to the visualization applications and finally describing how the algorithms were implemented.

5.1 The tpPPC Application

This is a description of the data acquirement tool called thesis project Pocket PC (tpPPC). The first sub-chapter 5.1.1 makes the connection between implementation chapter and the model I used for this master thesis project.

5.1.1 *Part 1 of the model – A tool preparation part*

In this part (shown in figure 5.1.1.1) the data acquirement tool have been created to be able to gather interesting and relevant sensor data.

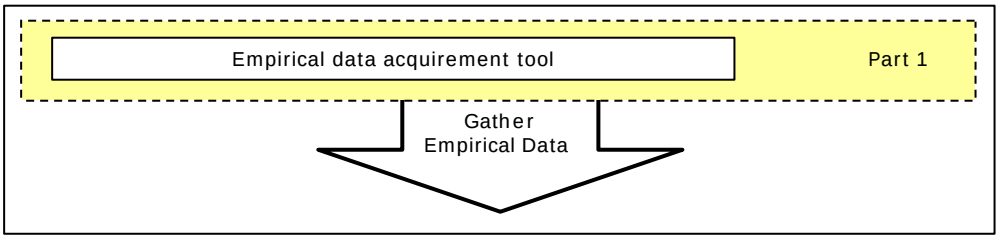


Figure 5.1.1.1: First part of the model.

The tpPPC application is a MFC application made in Microsoft Visual Studio 2005[23] (VS2005), in the C++ language. It has been done this way since was easy to create and debug the application with a Pocket PC. Also another reason for this is that the BSP prototype is already written in C++. The way this is done is quite systematic. The systematic way (with requirements that has been fulfilled) was:

1. Firstly the application made is compatible with the Pocket PC 2003 platform.

2. Furthermore the syntax of the code for the application should resemble the existing code written for each sensor module. Although the code for the Microstrain sensor module is to be reconstructed since it could be argued to be not well structured, this wish has been expressed by the BSP development team. This implies developing the

[22] The customized model presented in chapter 2.1
[23] More about Microsoft Visual Studio 2005 at: http://msdn2.microsoft.com/sv-se/vstudio/default.aspx.

gyro code in a way to make it resemble the GPS code and of course makes it possible for others to more easily grasp the essential parts of the code. To meet the latter requirement the code was written in a more well-structured and well-documented manner (in my own point of view).

3. The parsing of the sensors data was made in a logically way so that further scripting of the output data files was easy. This implied easy access to the different types of data, practically made so that the recordings was saved in separate files (e.g. saved in XXXXXX_logfileNrNN.txt files where XXXXXX is the type to be recorded and NN the batch number of the current test run). The batch numbers serves as the key to knowing if any of the three files recorded in one batch are from the same test run.

4. Finally there were of course demands on the usability of the tool. The tool application should run smoothly and offer adequate performance in speed, because it should not hinder the recording of sensor data while running the tool in a real world environment. That is running the tool while traveling in a car and aiming for GIS objects; the GIS objects have been selected for future reference making it possible to distinguish an actual object to make comparisons with the OSREP calculated locations to ensure the validity and to verify that the triangulation worked.

And as shown in the figure the finalizing arrow means that test runs are made with this tool to gather (i.e. records data into the files mentioned) the necessary data. After that empirical acquirement was done the second part followed consecutively.

5.1.2 *Conceptual Design of the tpPPC Application*

This sub-chapter serves as the more code concentrated description of the tpPPC application. In figure 5.1.2.1 a class view is presented of how the tpPPC application was done. This class view shows the different classes that constitutes three different blocks of the tpPPC application.

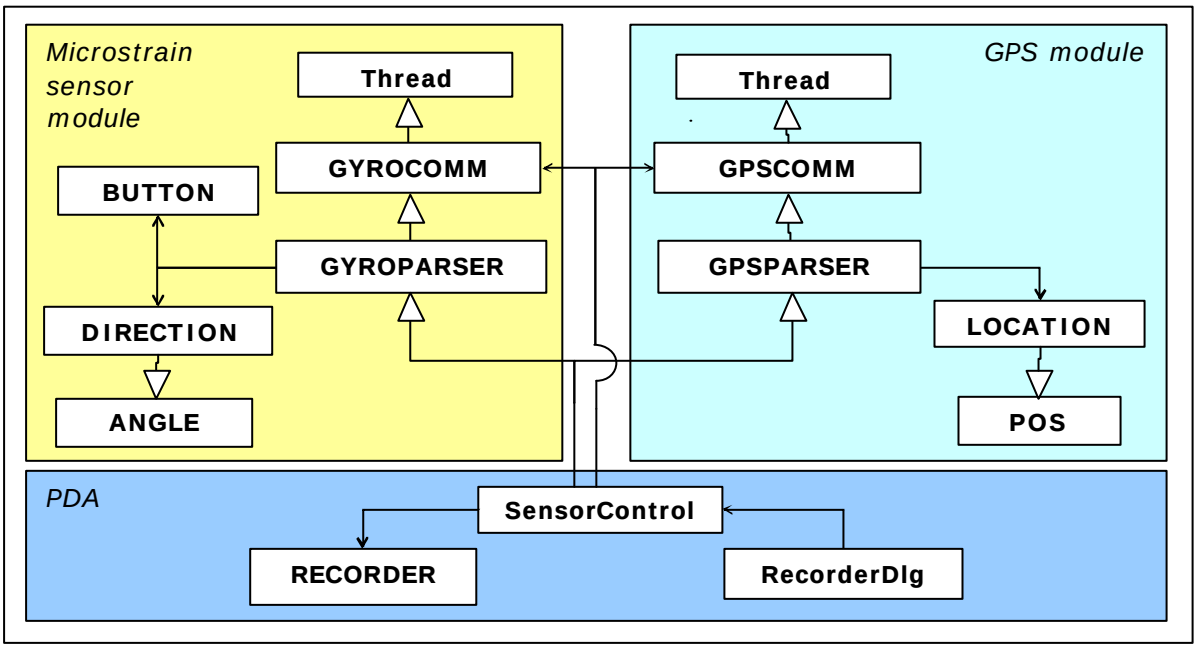


Figure 5.1.2.1: The conceptual design of the tpPPC application.

A list of descriptions corresponding to each block (i.e. the Microstrain sensor module, GPS module and PDA blocks) is hereby declared. The descriptions tell the purpose of the class and when applicable it tells about the class's relations to other classes. In the Microstrain sensor module (MSM) block the following classes are defined:

- BUTTON; Simple class with instance variables that constitutes the properties for button states, time stamps etc. a long with a get-method. The BUTTON class is used by the GYROPARSER since the button hardware is connected to a pin on the Microstrain sensor module hardware. Thereof this is the easiest way to connect the button with the rest of the application.

- ANGLE; Contains instance variable angle and its corresponding get- and set-methods along with some other useful angle calculation methods.

- DIRECTION inherits ANGLE since directions is a form of angle. Although an instance of this class has properties for the different kinds of modes in the sensor data from the Microstrain sensor module. The instance also keeps track of its timestamp.

- GYROCOMM; this class polls direction data through the ports it has opened to the MSM. This class inherits Thread since the port to the MSM has to be kept alive and listened to all the time.

- The GYROPARSER class inherits the GYROCOMM class. For every button data this class checks if the switch of the button is valid. For every byte of direction data this class distinguishes the different types of values and stores them into a DIRECTION object.

In the GPS module block the following classes are defined:

- The POS class contains specific methods for translating geodetic latitude and longitude into degrees or radians, if this is wished for.

- LOCATION inherits POS since locations are positions. Although an instance of LOCATION has properties as to storing latitude and longitude, speed and course. Also an instance of LOCATION stores the timestamp of its corresponding latitude and longitude.

- GPSCOMM; this class establishes connection to the Bluetooth GPS module by opening a port through which an instance of this class receives bytes of location data.

- The GPSPARSER parses the received bytes by distinguishing the different types of values and stores them into a LOCATION object.

To make all this possible to control the PDA block contains:

- A RECORDER class that starts threads to every log file. But before doing so it searches the folder in which the log files are to be placed. This is how the files are numbered consecutively with an increasing batch number corresponding to a unique test run.

- The RecorderDlg class serves as the dialog where the start and stop commands are given. Also the update frequencies for the position and direction data are seen in it along with the button states. This class initiates an instance of the SensorControl class when the user presses start.

- SensorControl inherits both GPSPARSER and GYROPARSER. The SensorControl class is where a RECORDER instance is created for recording every processed sensor data into three different log files as mentioned 5.1.1. Moreover this class deals with the input from the user passed to this instance from the RecorderDlg. I.e. when the user presses start this class creates instances of the GYROCOMM and GPSCOMM. These open the ports and start to listen to incoming bytes. And when those bytes have been parsed the SensorControl answers to a call to each update method that is

implemented as virtual update methods in each parser class. The corresponding update method in SensorControl stores values into the log files through the RECORDER object.

5.2 Conducting Test Runs

The test runs that were conducted took place in the island of Lidingö just north-east of Stockholm, Sweden. To gather useful data this location is used because the BSP development team could use the data for future comparisons when using their own simulator which uses GIS data. The team has such GIS data for Lidingö along with a map over Lidingö.

The main objective for the test runs was to gather button states, GPS positions and direction data simultaneously to be able to analyze player behavior in a BSP similar environment. Therefore I used tpPPC together with a sketch pad (for making notes, which could be found in Appendix E) and I sat as a passenger in a car traveling through the road network on Lidingö. The main things that were done during a single test run were to begin with to initialize tpPPC by starting it. After that I very intensively focused on finding appropriate targets that I also thought could be suitable GIS objects for the BSP stories. When I found one I pressed the button on the gaming device and tried to pinpoint the object. To use this button acts like a security measure if I had not been able to derive the first OSREP algorithm. After that I wrote down what type of object I had tried to pinpoint along with which batch number the test run belonged to. Then it was time to do it all over again.

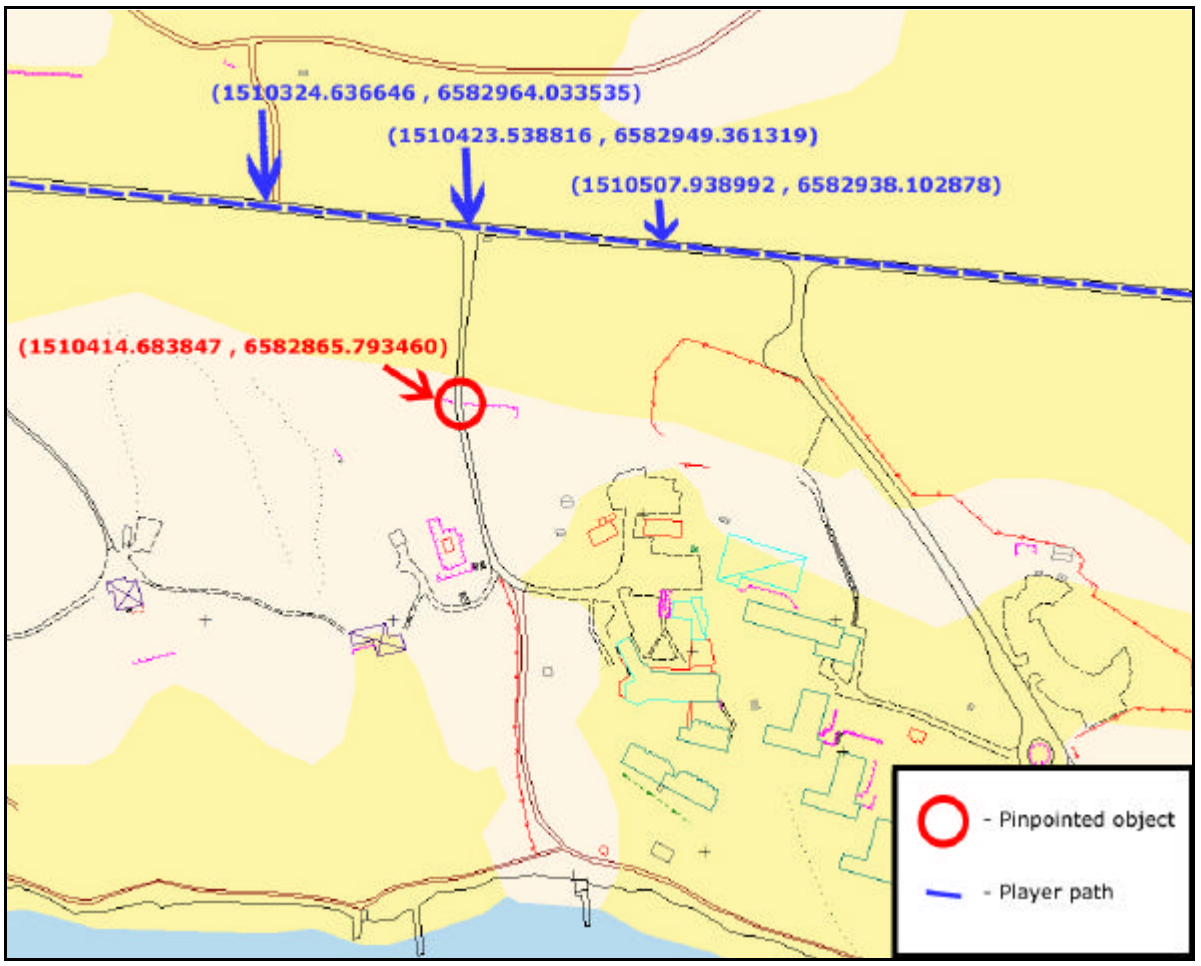


Figure 5.2.1: Example of test run showing player path and also showing where the pinpointed object's coordinates should be.

An example of a traveled route is shown in figure 5.2.1 where I have also marked out the object that I tried to pinpoint (I have also marked out the coordinates for that object using a map tool to extract coordinates in RT90 format , $Object_{x0}$ =1510414.683847 and $Object_{y0}$ =6582865.793460). For future reference the distance B between $Road_{x1}$ =1510423.538816 $Road_{y1}$ =6582949.361319 are calculated using Pythagoras (the unit for these coordinates is meters) like in figure 5.2.2.

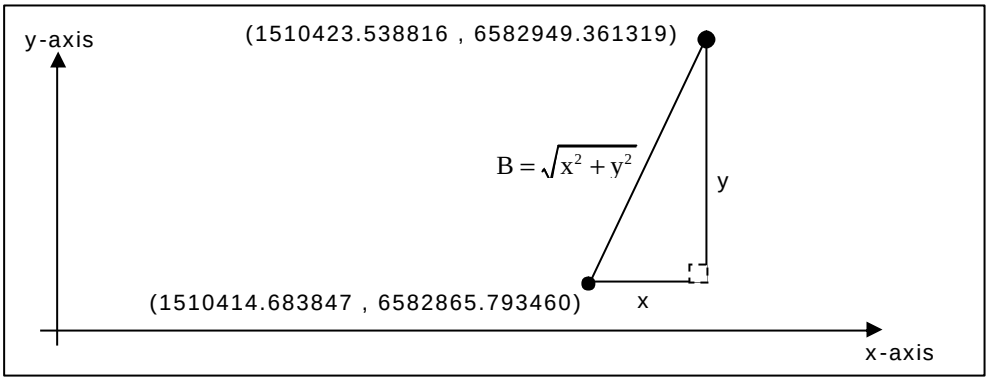


Figure 5.2.2: Distance B between road and object.

Distance B $= \sqrt{(1510423.538816-1510414.683847)^2 + (6582949.361319-6582865.793460)^2} = 84.03569202$ m.

5.3 Visualization Applications

This chapter is constructed much like 5.1. The first sub-chapter 5.3.1 expresses the implementation of the GISObjectApp in the sense of the customized model. But unlike chapter 5.1 this chapter describes two different solutions to visualizing OSREP. Although it is actually one half-finished application whiles the second solution is an ad-hoc Python hack for showing the mathematics behind the OSREP algorithms.

5.3.1 *Part 2 of the model – Analyzing and Visualizing*

The second part; shown in figure 5.3.2.1 from the customized model in figure 2.1.1, have dealt with preparing data and script it so that it could be computed on and visualized.

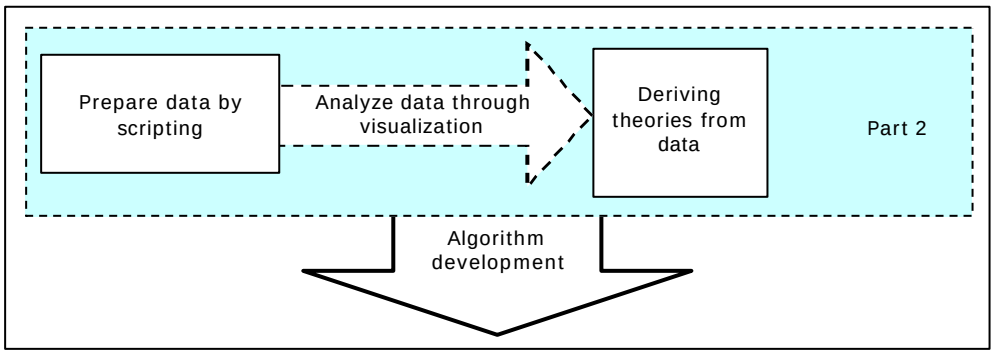


Figure 5.3.1.1: Second part of the model.

The scripting has been made through the use of Python[24] modules extended with a MFC application made in VS2005, which have been chosen because of the same reason as with the tool presented in chapter 5.1, that the BSP prototype has been developed through the use of Python and VS2005. This MFC Application is called GISObjectApp and is essentially a

[24] More about the Python programming language at: http://www.python.org.

dialog with an ActiveX control for showing the GIS objects and a map over the Lidingö area in Stockholm. Why there specifically should be an ActiveX control in this application was because of the MapInfo MapX 5.0 software, which is written in C++ and is very easy to implement as an ActiveX control.

To motivate the choice to even considering the development of this GISObjectApp was to make it work like an easy-to-integrate module for the BSP. In this way an extended Python module would be developed and it would be easy to just copy-paste the code and insert into the server-side/simulator of the BSP prototype. Although this is a nice thought it turned out to be somewhat time consuming. So therefore; as explained in the list below, some other Python modules were used.

The requirements that were brought to attention (and how things were done to meet them) for this part was:

1. A tool that was easy to use was of course a top priority, the application should be able to start and stop simulations (i.e. simulations with the gathered sensor data). Furthermore the ability to activate and deactivate the algorithm and some other basic controls as for viewing the GIS Object Map[25] (e.g. zooming in and out) was also required. To switch on and off the algorithms was of course because of the fact to easily see performance differences. Also to see a graph window presenting calculations made to the gathered sensor data is a nice feature both when presenting result as to quickly see progression of the forthcoming development of the OSREP algorithms. Finally, to implement some simple map controls was an easy task since those are pretty much already accessible functions for them in the MapInfo MapX 5.0 software and the use of them is quite obvious.

2. GISObjectApp was further intended to be a usable tool for the verification and validation (part 3 of the model) but as already mentioned the application was never finished and because of that it offers no usable controls as to presenting and visualizing nice graphs for analysis nor do it allow for viewing GPS positions and directional information on the map. This makes it a rather useless tool in its current state.

3. To be able to finalize this part there was a requirement to be able to see how the algorithm performed on the gathered data while developing it. To do so Python script was developed using the Numpy[26] and Matplotlib[27] modules which made it possible to plot graphs. These graphs was intended to be the graph window part of the GISObjectApp and in the simplest way presented as an image when the simulation had finished processing the sensor data. Unfortunately this integration was never done due to the time constraints in the project; more focus was instead put to deliver any sort of interesting results.

The final output of this part was to implement the algorithms from the derived theories. This is shown as the big arrow pointing downwards in the figure above. That arrow is further explained in the chapter 5.4 as the implementation of the OSREP algorithms.

[25] The MapInfo MapX 5.0 ActiveX control; it shows GIS objects and a map over Lidingö.
[26] More information about the Numpy module could be found here: http://www.scipy.org/NumPy.
[27] More information about the Matplotlib module could be found here: http://matplotlib.sourceforge.net.

5.3.2 *Conceptual Design of the GISObjectApp*

This section offers a more code concentrated description of GISObjectApp. This is presented in this report as an aid for further development if there ever will be any. The design concept is viewed in figure 5.3.2.1.

The block called Python Scripts consists of a python script that reads the log files and for each value it reads carries out some computation on. These computations would then be what have been referred to as the OSREP algorithms. Meanwhile the class PythonModule constitutes the negotiator between the MFC application and the Python scripts. This class is constructed as an extending python module[28], called "sim", which actually extends the Python Script block in figure 5.3.2.1. With this extension the Python Script can use methods in the PythonModule class to interact with the rest of the C++ written application. The major reason for extending a python script like this is because development goes faster.

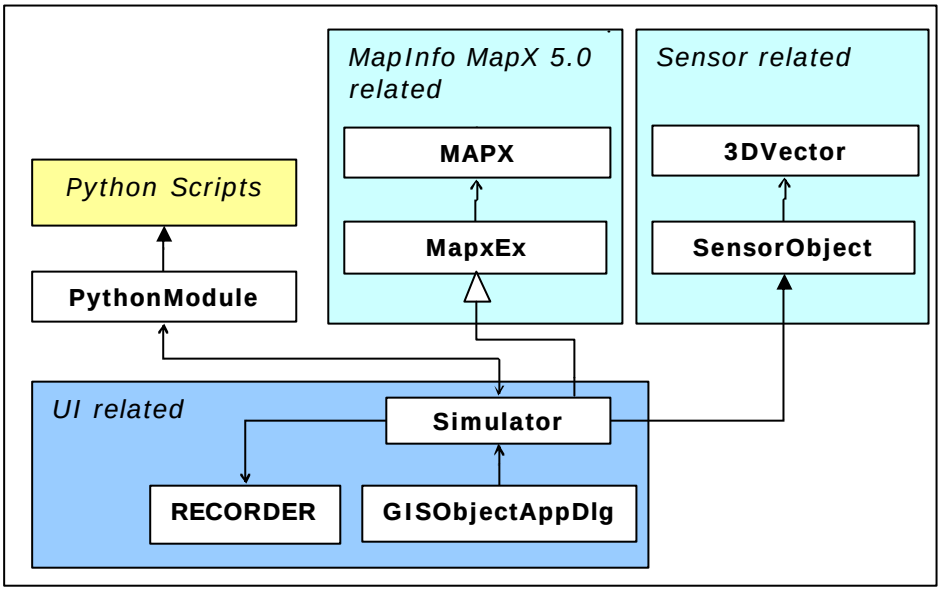


Figure 5.3.2.1: The conceptual design of GISObjectApp.

The two classes named MAPX and MapxEx are described as:

- MAPX is a class containing many pre-defined functions regarding drawing on the map, map layers, etc. MAPX is delivered as a sample class with the MapInfo MapX 5.0 software.

- MapxEx on the other hand is an intermediate class containing functions relevant to the GISObjectApp while using some of the pre-defined functions in the MAPX class.

The block named UI related consists of three classes of which the RECORDER class already has been presented (in chapter 5.1.2). The two classes that are left are:

- Simulator inherits MapxEx since it should be able to draw objects onto the map area[29] of the GISObjectApp, such as gaming device directions and player positions. As a final stage; a geometry representing the object that the player has visualized is placed on the map. Beside this the Simulator class runs two separate threads which listens for position and direction events. When such an event arrives the intention (observe this

[28] More about extending python scripts in the "Extending and Embedding" chapter in the ActivePython 2.4 Documentation found at: http://aspn.activestate.com/ASPN/docs/ActivePython/2.4/python/ext/ext.html.
[29] A visualization of the current version of GISObjectApp is found in chapter 6.3.

is not finished) is that it should be drawn on the map, making it possible for the user of the GISObjectApp to view the simulation of the data gathered from the test runs.
- The GISObjectAppDlg class; which is a simple dialog with a bunch of controls already mentioned in sub-chapter 5.3.1.

Going over to the sensor module block, these two classes is dealing with how to represent sensor data to the simulator class and for easier storing of arriving position and direction events. The 3DVector is sort of self explanatory; it is just a class making it possible to represent the direction data as vectors. The SensorObject class is just a generalization of all sensor types (e.g. GPS sensor, direction sensor and button sensor).

5.3.3 *Python Script Hacks*

It turned out that there were some sweet Python modules that could help in visualizing OSREP results. These two modules have already been described in 5.3.1 as modules making it possible to create plots. I want to emphasize that although this was some last minute python hacks the mathematics behind these simple visualizations are still valid. It is only the simplistic kind of visualization I want to call hack (i.e. a hack in not carefully coded and documented).

The change of visualization application had some impact on how the validation and verification was done. While verifying that the algorithm works; the third part of the methodology model (shown in figure 5.3.3.1 below) acted like a final test. To make the validation and verification there was actually no need for a fancy visualization application. Instead this part was visualized with the more simplistic use of the produced Matplotlib graphs. So there was never any loss of important features due to this change of plan, it only made things look uglier (at least I think so).

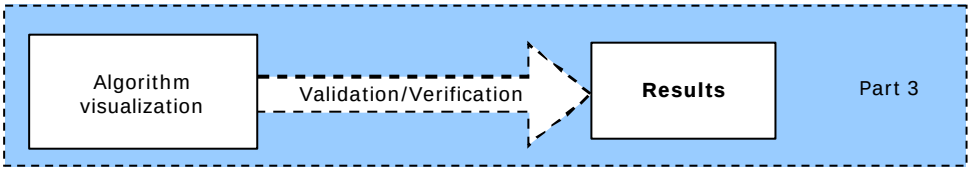


Figure 5.3.3.1: Third part of the model.

5.4 Implementing OSREP Algorithms

As mentioned earlier the actual algorithms that constitute the OSREP are made in the Python Programming Language. In figure 5.3.2.1 it is shown as the block on the top left named Python Script. This Python Script is divided into the following parts:

1. Converted the data in the log files into lists. This allowed for easier handling and computations.

2. Selecting sets of values to simulate real world, because in the real world there is not a finite set of direction data and position data covering the whole scenario of pinpointing an object. Instead the algorithm works for small sets within some interval that in the end will entail the whole pinpointing scenario but to begin with is just a couple of GPS positions with the direction data in between those.

3. For every small set of data (i.e. a small set of data is at the minimum 2 GPS positions and the direction data in between those, as mentioned in earlier chapters to be the interval decided by the first OSREP algorithm) the theories stated in 4.2 and 4.3 are carried out and plotted. Although a remark is to be put here since some very

surprising results came up while implementing OSREP. The need for a user input to signal start and stop of the GPS positions interval was not needed. The second OSREP algorithm[30] seems to be working by its own. More on this in the result chapter.

4. Using the Python Hack a plot is presented showing the path that the player has taken with the directions graphical representation left out but a geometric object (e.g. a filled circle) has been placed where OSREP thinks the player has aimed.

Verification of the fourth step is to compare the computed locations and the correct answers (i.e. the correct answer is extracted from the combination of the notes I wrote during the test runs and a map over Lidingö).

[30] See Appendix F for viewing the python code for the OSREP algorithm.

6. Results

This chapter reveals the results of gathered from the implementation part of this master thesis project. The results presented here are those of the tpPPC, the test runs, visualization applications and finally how the OSREP was performing. The latter is a performance result that will answer the question whether it is possible to create the new generation of pinpointing using the hardware stated in chapter 3. It is not a performance result that will show in more depth as to how good the OSREP algorithm is since the task at hand was not to create the best algorithm in the world for this purpose; it was just to determine if it was a possibility to create OSREP. But more discussions and conclusions are found in chapter 7.

6.1 Results for the tpPPC application

The result of the tool preparation part is an empirical data acquirement tool (seen in figure 6.1.1) that could be run on a PDA. The tool on the PDA is called "thesis project Pocket PC" (shortening viewed in the figure as tpPPC). This PDA is as seen in figure 1.1.1 the PDA placed on the gaming device. This tool makes it possible to see three things while running the acquirement tool and also it allows doing two things.

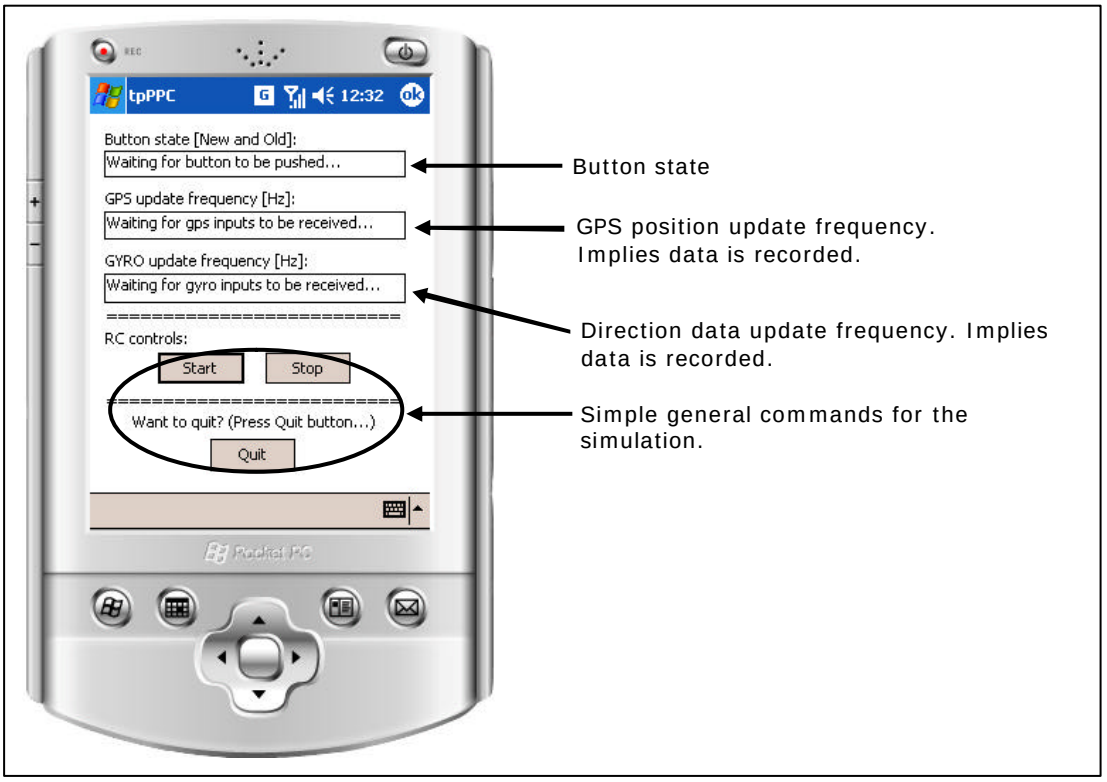


Figure 6.1.1: The tpPPC tool on a PDA.

The three things that could be viewed during any run are:

1. The button state. The trigger button on the gaming device, not any of the buttons on the PDA. It shows the current state and the old state that has been recorded to the button log file. The reason for this is because the trigger button is not chatter free (i.e. chatter is contact bounces which would get the effect of switching on/off the button several times when you've actually only pressed it once).

2. The GPS update frequency (with current GPS module the update rate is around 1 Hz). This implies whether GPS data is received and recorded into the GPS log file or not.

3. The direction data update frequency (with the Microstrain 3DM GX1 module the current update rate is around 70 Hz). This implies whether direction data is received and recorded into the gyro log file.

The two things that could be done are:

1. Start a new sequence (i.e. three new log files are created with a new batch number corresponding to the current batch, batch numbers start at zero).

2. Stop the sequence (i.e. ends the current batch with the batch number created when starting the sequence).

6.2 Results from the Test Runs

The results from the test runs are stored in ordinary .txt-files called log files since they in fact consists of recorded (logged) sensor data. This of course results in very large files therefore only a shortened example of the results is shown here.

One set of recorded button states in one test run (showing one press on the button and one release of the button, and the second value on each row is the system tick count for the corresponding button state) is seen in figure 6.2.1:

```
1,141046
0,144251
```

Figure 6.2.1: The acquired button states.

This is the recorded GPS positions in the same test run:

```
5924.288400,-1757.023300,4.077962,0.076033,2408772992,29840848,142027
5924.288400,-1757.023400,4.077962,0.063596,2418772992,29840848,143316
5924.288500,-1757.023500,4.077962,0.067234,2428772992,29840848,143952
```

Figure 6.2.2: The acquired GPS positions.

The different values in figure 6.2.2 are divided by commas and have the following categorizations:

1. Degrees in minutes and seconds, latitude.
2. Degrees in minutes and seconds, longitude.
3. Course
4. Speed (m/s)
5. High DWORD of FILETIME conversion of the GPS date/time format.
6. Low DWORD of FILETIME conversion of the GPS date/time format.

7. System tick count

The recorded gyro data is seen in table 6.2.3, over the corresponding time span as in figures 6.2.1 and 6.2.2, in the same test run.

```
3,0.14661,-0.03741,-0.48151,0.05511,0.01154,0.99762,-0.04695,-0.04410,0.01193,44.754536,141063
2,0.14301,-0.03418,-0.48315,0.04422,0.00705,1.00146,-0.01894,-0.05473,0.05759,44.767643,141077
3,0.14642,-0.03857,-0.48242,0.02969,-0.01260,1.01407,-0.00700,-0.05344,0.06174,44.780750,141089
2,0.14221,-0.03448,-0.48340,0.04572,0.00705,1.00146,0.03813,-0.05240,0.04565,44.793858,141103
3,0.14313,-0.03821,-0.48187,0.04401,-0.02222,1.00681,0.01686,-0.02335,-0.02257,44.813519,141122
…
(Two pages of data have been cut)
…
3,0.14520,-0.03796,-0.48254,0.03802,0.00983,1.01279,-0.01115,0.00934,-0.04669,47.559475,143876
2,0.14288,-0.03656,-0.48297,0.04422,0.00961,1.00104,0.00337,-0.01427,-0.02075,47.572582,143888
3,0.14282,-0.03638,-0.47949,0.04059,-0.00150,1.00403,-0.00934,-0.03502,0.00674,47.598797,143914
2,0.14221,-0.03668,-0.48315,0.04572,0.00983,1.00104,0.01556,-0.04306,0.00259,47.611904,143927
3,0.14471,-0.04028,-0.48370,0.06366,-0.00940,0.98203,0.01997,-0.02464,0.01660,47.625011,143940
```

Figure 6.2.3: The acquired gyro data (with two A4-pages of data cut out).

The values are divided by commas and the different values have the following categories: (The modes; 2 = INSTANTANEOUS Mode and 3 = GYRO-STABILIZED Mode is saved as a sign of which kind of command has been given to the Microstrain sensor module. The modes also tell which kind of data that is shown in each row of the log file.)

If mode = 2 the following categories apply to the line:

1. Mode = 2
2. INSTANTANEOUS MAGFIELD (x-axis)
3. INSTANTANEOUS MAGFIELD (y-axis)
4. INSTANTANEOUS MAGFIELD (z-axis)
5. INSTANTANEOUS ACCELEROMETER (x-axis)
6. INSTANTANEOUS ACCELEROMETER (y-axis)
7. INSTANTANEOUS ACCELEROMETER (z-axis)
8. INSTANTANEOUS ANGRATE (x-axis)
9. INSTANTANEOUS ANGRATE (y-axis)
10. INSTANTANEOUS ANGRATE (z-axis)
11. Gyro tick count
12. PDA tick count

If mode = 3 the following categories apply to the line:

1. Mode = 3
2. GYRO-STABILIZED MAGFIELD (x-axis)
3. GYRO-STABILIZED MAGFIELD (y-axis)
4. GYRO-STABILIZED MAGFIELD (z-axis)
5. GYRO-STABILIZED ACCELEROMETER (x-axis)
6. GYRO-STABILIZED ACCELEROMETER (y-axis)
7. GYRO-STABILIZED ACCELEROMETER (z-axis)
8. BIAS-COMPENSATED ANGRATE (x-axis)
9. BIAS-COMPENSATED ANGRATE (x-axis)
10. BIAS-COMPENSATED ANGRATE (x-axis)
11. Gyro tick count
12. PDA tick count

6.3 The GISObjectApp visualization application

The result of the development of the GISObjectApp is a not finished application. The desired functionality is not yet implemented and the extension of the Python module is not finished.

Although the work done is a good start I think. In figure 6.3.1 the current version of the GISObjectApp is shown.

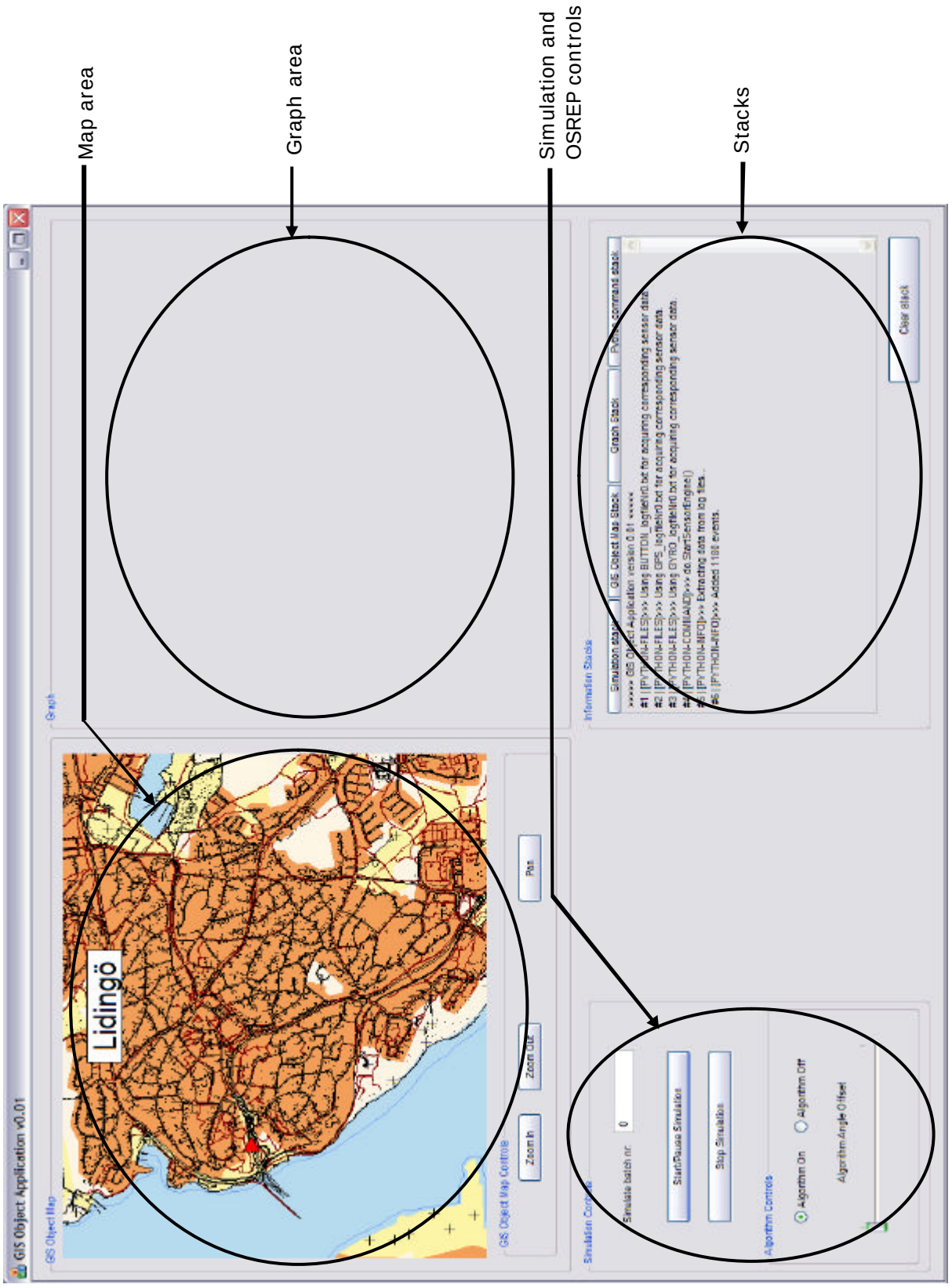


Figure 6.3.1: A visual of the GISObjectApp.

6.4 OSREP Results

Since OSREP was divided into two parts, two separate results are presented here. The graphical visualizations of these results are made with the use of the Python Hack described in chapter 5.3.3.

6.4.1 *Results of the First OSREP Algorithm*

The first result is shown in figure 6.4.1.1. It is s a graph over the first OSREP algorithm for which the purpose was to determine if the player was aiming towards an interesting object. As seen in the graph the player seems to be holding the device steady between the reference points of the button state signaling pinpointing and not pinpointing.

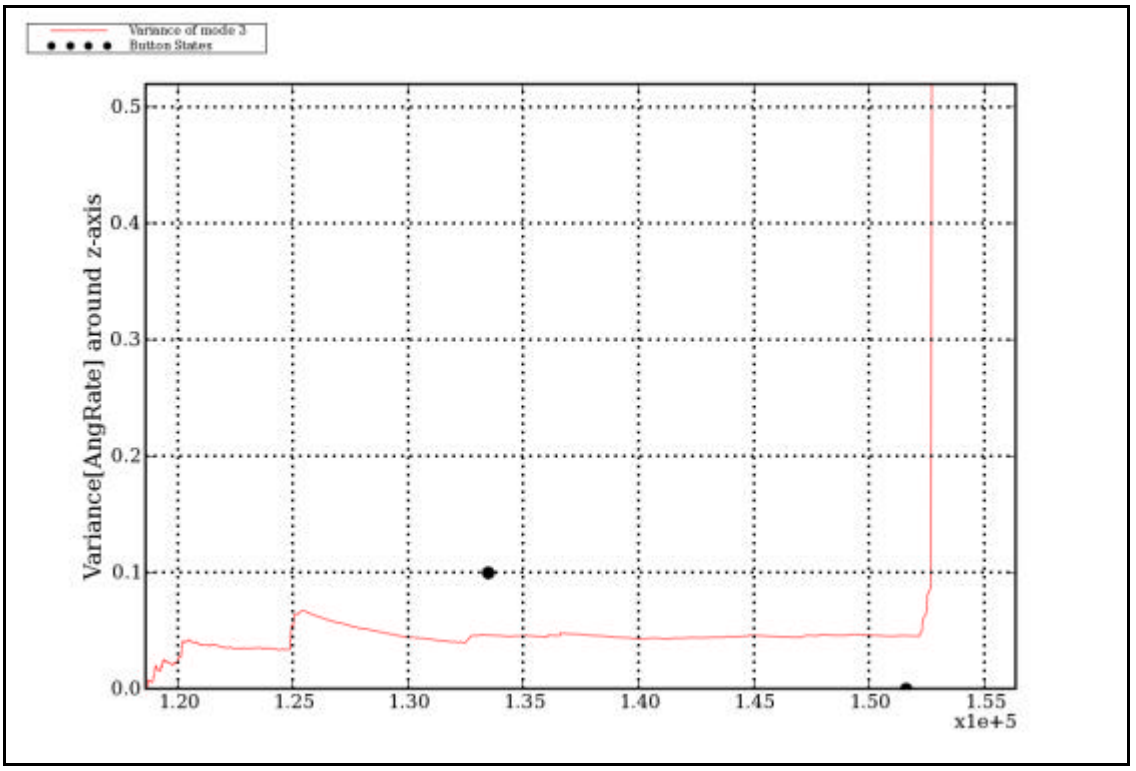


Figure 6.4.1.1: Example of how the variance looks like for the first algorithm of OSREP. This is from Test run 1 with stable mode of the Microstrain sensor module.

Moreover, figure 6.4.1.1 shows the two available modes of AngRate as continuously growing positively up to about 180 degrees. This insinuates that the player has gone from zero degree, when he or she started to pinpoint an object. When the player has reached the 180 degrees; it is shown that the gaming device has swiftly moved back to the starting direction. This implies that the player has stopped aiming and has sat himself comfortably facing forward in the car's direction of travel. In this graph the difference between the gathered instantaneous and stable modes of the Microstrain module is clearly shown. The instantaneous mode is the bottom line which seems to register a greater leap when the player moves back to the starting direction. Therefore it s less appealing to use that mode since it is more logical that the player does act like in the case with the stable mode. I.e. the player does not go 30 degrees past his starting positions, he would probably much rather stop when he is facing forward in direction of travel. The dots spaced equally over the whole graph are when GPS positions have been received while the dashed line presents the current speed of the car. These two are both received from the GPS Bluetooth device. And the two other dots mark the button push (signaling that the player started to pinpoint) and button release (signaling end of pinpointing).

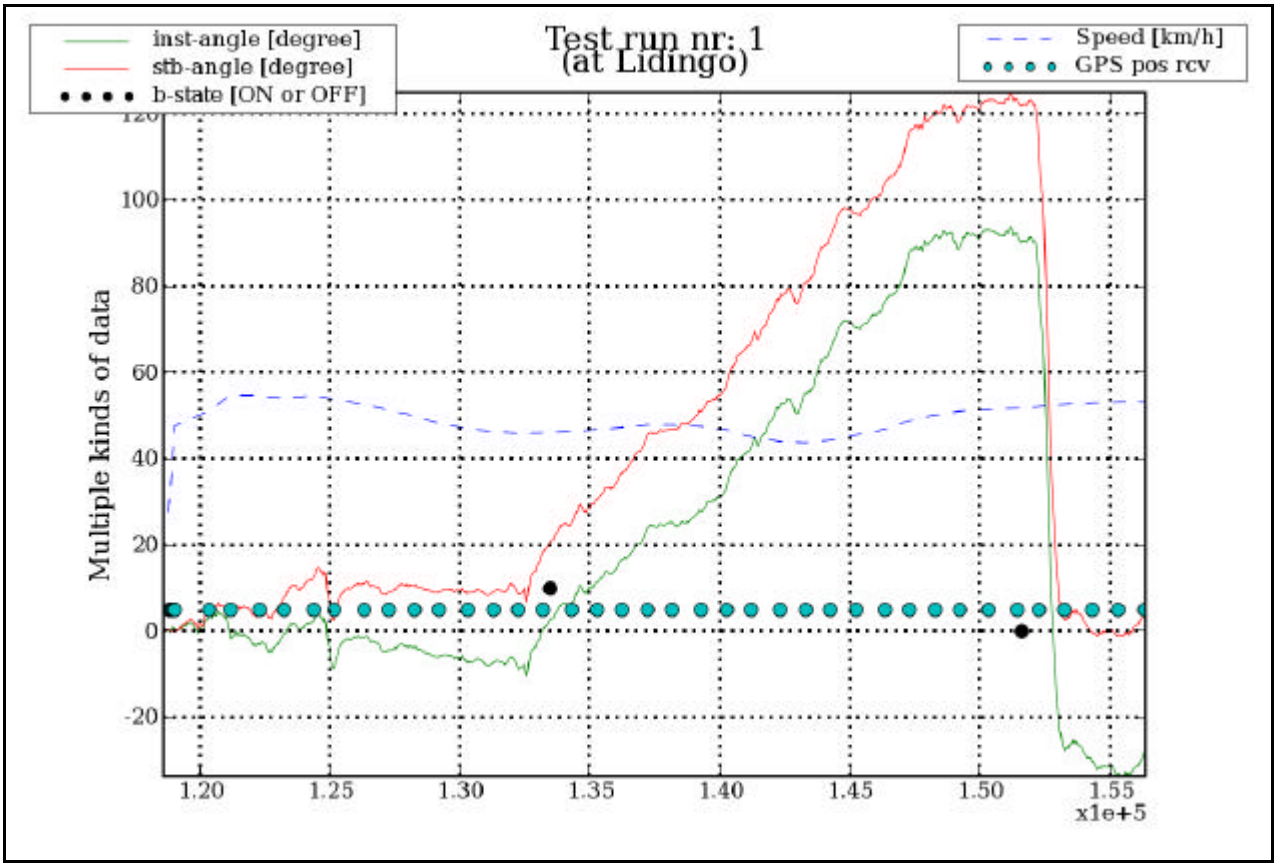


Figure 6.4.1.2: This graph shows that the angle is in fact increasing when the player is pinpointing an object on the left side. It also shows when comparing to figure 6.4.1.2 that the angle does vary in time and that the variance gets almost infinite when the player returns to his starting position.

6.4.2 *Results of the Second OSREP Algorithm*

The second result is more graphs showing the path of the player with the pinpointed object marked beside it. The graph in figure 6.4.2.1 is the result of the implementation the OSREP algorithms. The resulting graph could be compared to figure 5.2.1 that showed an example test run. This figure shows the player path as the connected dots while the POI representing the pinpointed object is shown alongside the path.

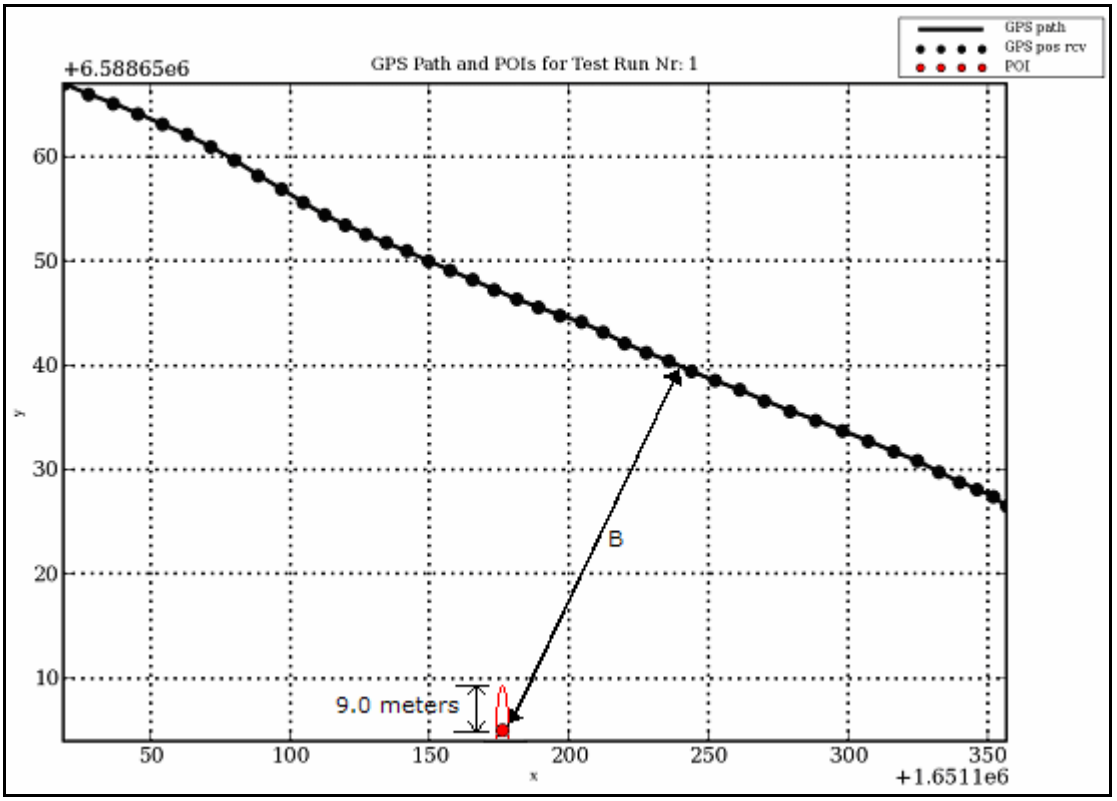


Figure 6.4.2.1: Result of how the result looks like for the second OSREP algorithm. The marked dot is the location of where OSREP thinks the player's pinpointed object lays.

In figure 6.4.2.2 a close-up of the resulting pinpointed object is shown. As seen the calculated radius is 9.0 meters. This is the difference between the POI with the largest amount of hit counts and the POI with the second largest amount of hit counts.

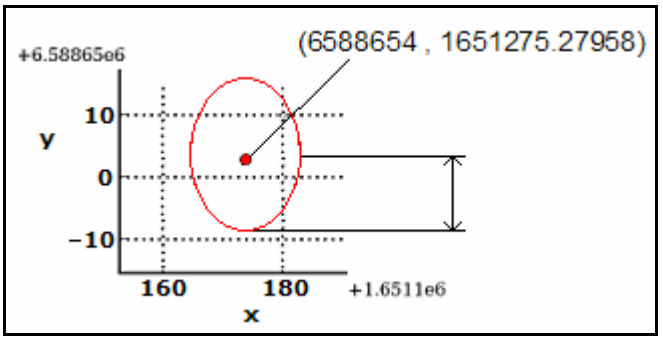


Figure 6.4.2.2: Close-up of the pinpointed object and its radius.

If I do the same distance calculation like for figure 5.2.2 the distance B between road and object becomes B = 94.692 meters. In comparison with the calculated distance B in chapter 5.2 this distance only differs by approximately 10 meters (which is 10% inaccuracy). The inaccuracy is satisfactory since from this test run, the pinpointed object's area (shown as the circle around the pinpointed object) has been calculated to be 9.0 meters. Evidently this radius is almost equal to the 10% inaccuracy and thereof the distance is as said satisfactory. But a result of my own belief in that I should do everything myself is that the coordinates are a bit inaccurate. This is because the transformation presented in Appendix C seems not to be working correctly although I spent almost two weeks of error correcting and trying out other projections (e.g. using the 7-parameter transformation also known as Bursa-Wolf[31]). But as I

[31] This transformation method could be found at Lantmäteriets web-site, here: http://www.lantmateriet.se/templates/LMV_Page.aspx?id=6054.

mentioned the GTrans from Lantmäteriet would most definitely fix that problem. Although this does not present any major coordinate errors. This is due to the fact that the errors are present in all coordinates in the calculation. The correction of these errors would be applied to all coordinates like I discussed in chapter 3.1.1 when I described the errors that are always present in a GPS and how to correct them.

One of the most important things that were shown during the development and testing of the OSREP algorithms was that it is not necessary for the player to press any buttons or in any other way hint that he or she is trying to pinpoint an object. Therefore the first OSREP algorithm is not necessary. This was of course why the first algorithm of the OSREP is a lot less well-defined in the theory chapter. The figures 6.4.2.3 and 6.4.2.4 shows graphs of one and the same test run while trying to pinpoint the same object as in figure 5.2.1. But the graphs in the figures have been processed with two different settings. The difference in setting in these figures are that the graph in figure 6.4.2.3 is the one with a button state interval while the one in figure 6.4.2.4 shows the same test run without any button state interval. A button state interval is explained as when OSREP defines its start and stop from reading the recorded button log file for signals of that the button had been pushed during a test run (e.g. it was always pushed to signal that I aimed, and the times for these button pushes are present in the button log file).

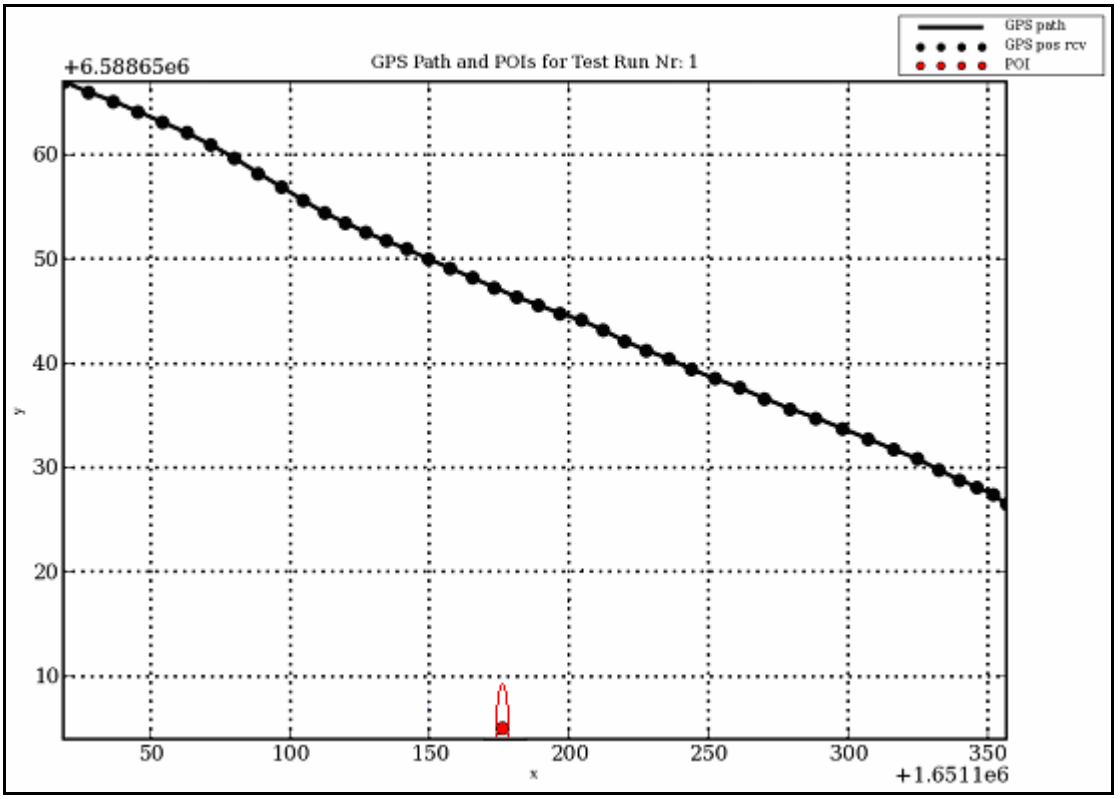


Figure 6.4.2.3: OSREP with a finite button state interval and offset angle set to 30 degrees.

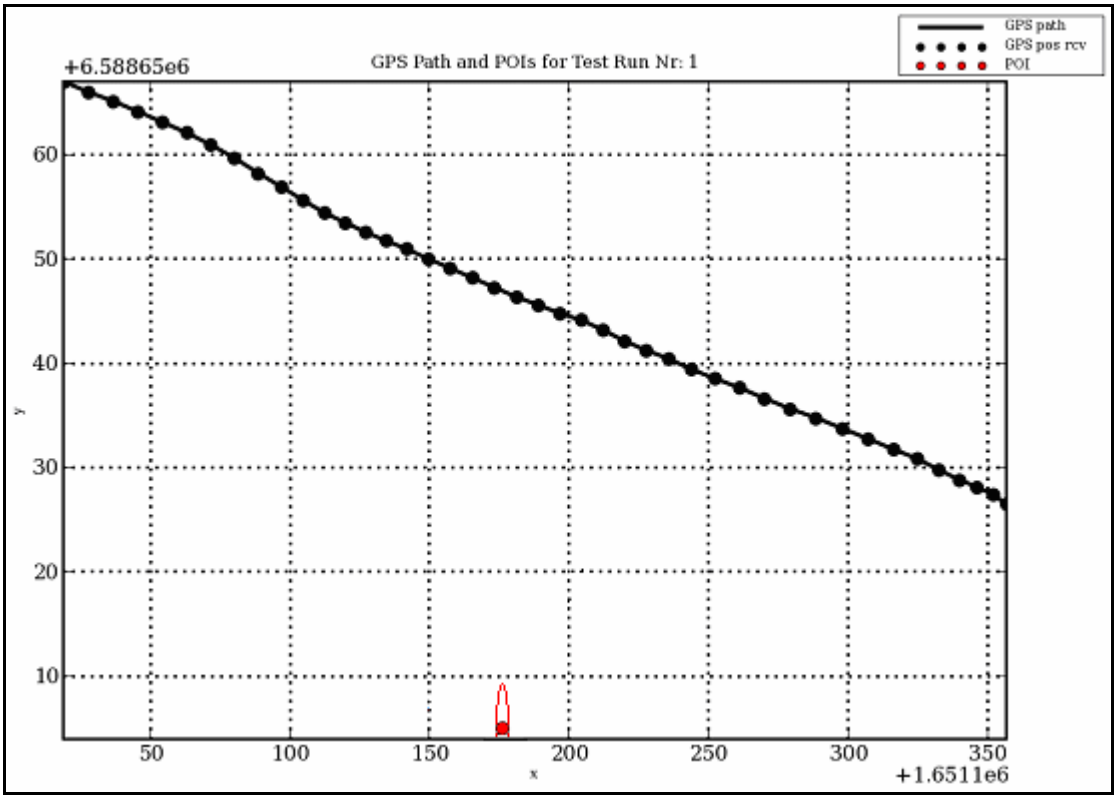


Figure 6.4.2.4: OSREP without a finite button state interval.

The calculation of the offset angle was quite satisfactory. As in the figures above the offset angle was computed to 30 degrees. That the offset angle was computed to be 30 degrees means that the starting direction was 30 degrees since this result in an x- and y-coordinate with the smallest object-tolerance (i.e. the object size was the smallest) present in the test run that these results come from.

7. Discussing and Concluding the Results

Chapter 7 presents the discussions and conclusions about the results presented in chapter 6. In the following sub-chapters several discussions are made and in the final sub-chapter the conclusions as to whether there is a possibility to create sufficiently good OSREP algorithms making it possible to develop tools in BSP for future User Content Creation.

7.1 The Applications

7.1.1 *The tpPPC*

The usability of the tool is in general good. It provides the user with the basic information during test runs. It provides an easy to use Graphical User Interface (GUI) with only two buttons and some text visualizations providing update rates and button state. This could of course have been more complex, providing sliders for changing these update rates. But when designing this tool and viewing the requirements for it there were no need of such complexity. However because of the fact that the GPS module uses Bluetooth there was a rather annoying fact that occurred. Every time a new recording was to be started I had to manually choose the GPS module to be connected. So in the future try to fix this.

To design the tool in this way does not however limit further development of it. Since the design provides only for the basic functionality it is actually offering more flexibility. If there are specific needs, those could easily be added to the existing code creating a more powerful tool.

The textual output files earlier described as log files are easy manageable files containing the most significant data needed for further processing. The choice of creating three different files for button-, GPS- and gyro data was a success since it required minimal effort of searching for a certain sensor output. Meanwhile it also makes it possible to add further sensor output by explicitly writing code for it and recording its data to another file.

7.1.2 *The GISObjectApp*

Not much to say beside the fact that I am disappointed that I did not find the time to finish this application. It would have been a nice demonstrational tool when presenting the master thesis project. If another student is to continue my work he or she could perhaps find this useful for visualizing objects that should be presented on a map using the MapInfo MapX 5.0 software.

7.1.3 *The Python Hack*

Although this is a hack it turned out to be very useful. The hack was actually used as the tool for developing OSREP. It was used as a proof of concept which means that this hack not only is a visualization tool it actually has the OSREP integrated in itself. Or at least that is how it was delivered to the Mobility Studio at Interactive Institute. This was not the way I intended it to be but it seemed to be the only way to present the results when the presentation

deadline was approaching. Neither had I thought that the usability of the Python hack would be anything to talk about, which it was not. But as late as in the last couple of days before presentation some breakthroughs regarding OSREP algorithms were made and thereof it was again interesting to present some GUI that allows the user to change settings in a more presentable way than viewing and editing code.

7.2 Are the OSREP Algorithms Sufficiently Good?

Measuring the validity of the OSREP algorithms is to compare the processed test run data with the notes I took from when I conducted the test runs. These notes reveal that OSREP outputs an adequately good result. It works to the degree that it retrieves a Point-Of-Intersection with quite satisfactory distance between the player path and the pinpointed object (an inaccuracy of 10%). Although comparing that result to the inaccuracy of a GPS it is very satisfactory. Moreover, the position of the pinpointed object is a satisfactory result when comparing it to maps. But since I'm using my own coordinate projection it would be good to change this in the future. My comparisons with maps are made by adding a false northing and false easting of 140885.5 and 5765 meters respectively. But due to the customizations of FN and FE I would like to make the future developers aware of this.

Discussing the absence of the first OSREP algorithm while testing the complete OSREP algorithm I argue that it is not necessary to implement it. It is clearly sufficient to only implement the second algorithm. The main reason for this result is by my understanding the fact that the second algorithm calculates all POIs present between all GPS positions within a time frame set by the developer. A problem or in detail a matter of implementation would be to slide that interval as the player travels trough the road network making it work like a sliding window. This I believe will not offer any further complexity since it is only to implement an exclusion of sufficiently old GPS positions. This solution would be sufficient since the last GPS position in the sliding window would be the last recorded GPS position. Although this introduces a time constraint on the time performance of the OSREP it will mostly be a question of optimizing the code and optimization of the different parameters present in the Python code for the OSREP algorithm. By testing the OSREP I have clearly seen the lack of performance in the algorithm when regarding the response time for it. It takes quite a long time to process all directional data along with the GPS positions. This offers a long delay from when the player has stopped pinpointing the object which could be quite distracting. But if the BSP developers would like to they can put some effort into presenting this delay as useful. E.g. while computations are made a voice is presenting information like "We are now processing your information. Play this side mission to earn more points in the game while waiting for the results".

What has proven to be the greatest factors of lack in performance is the O in figure 4.3.1.1 which is the angle that controls how wide the GPS position interval could possibly get. Another very annoying fact is the step length of x in the pseudo-code in figure 4.3.1.9. In the ideal case I would want to take steps of length (speed of the car)*(update frequency of the Microstrain sensor module equal to 1/35 second) which results in steps of approximately 0.3968 meters. But to do so I would have to need a monster computer. The PDA used in the BSP prototype is for certain not a monster computer. Why I would like the x to be able to take small steps is because it determines the x-coordinate of the pinpointed object (the result of this factor is as seen in Appendix D that the x-coordinate is an integer).

7.3 Lessons learned

Firstly the main thing I learned is to be very precise when delimitating a project. Not doing so in a sufficiently good way turned out to be a heavy load. It puts very much stress into to the final touches that has to be made. Mainly some of the areas in the report have suffered from it. This could have been avoided if I had kept a very detailed diary to keep track of when a

new stage was begun and what I did during that stage. Secondly I have realized the importance of planning and in greater detail investigate how time consuming the different stages in the project could be.

Understanding code and the importance of documenting code has been very educational. Not saying that I'm a master at documenting code but that I will from now on always put some nice comments beside an important part of code. Specifically this holds for a thesis like this when the reader could be some other master student continuing the work. Hopefully my approximately 10 000 lines of code is more well-documented and understandable.

But what I was very surprised about was the time it took to start up the project. I remember I was thinking that I was never going to get started but finally I did. This could have been avoided if Uppsala University would hold a project methodology course before the master thesis project. This would allow for the students to feel more confident when initiating a bigger project compared to the ordinary 5 point courses they go through.

7.4 Future Work

Using this kind of gaming device in a mobile AR game is doomed in advance. First of all the Microstrain sensor module costs about 15 000 Swedish crowns which imply that to buy the game you have to be quite rich. If you add the fact that a parallel master thesis is investigating how to find out directions using the camera of a mobile phone the hardware used in this master thesis could be thrown away if that is a success. Although the conceptual ideas of the OSREP that I have come up with could of course be applied to that platform instead. This would show if OSREP could be more applicable to that kind of pinpointing hardware and thereby prove to be of some use after all.

By further developing the Python hack I think there is a lot more to be found. To come up with more ways to weigh the POIs I believe a key point. Moreover I think there has to be some work done to the part in the code where I distinguish the travel direction to discard non-valid POI.

Another thing that the future developer would have to change is my own projection method to the one provided for in GTrans[32]. This tool is available for Python coding through extending a Python module with C++. This will increase the accuracy in the x- and y-coordinates for the GPS input to the OSREP algorithm.

[32] Information about GTrans is found at Lantmäteriets web-site, here: http://www.lantmateriet.se/templates/LMV_Page.aspx?id=1564.

8. Acknowledgements

First I would like to thank Anton Gustafsson for taking me under his wings and for the very precise expertise and constructive feedback in several areas of this master thesis project. Along with him I would also like to thank Oskar Juhlin for valuable and constructive feedback during the progress meetings and halfway presentation. Also I would like to thank the whole team at Interactive Institute's Mobility Studio for their kind support and for being nice team members.

I also would like to thank Tasawar Khan who drove me during one test run. Thank you for your patience. Beside him I would like to thank every other master student who I have talked and discussed various matters with. It was nice to have met you all and nice of you to have shared some of your knowledge with me.

9. References

Here you see all the references I have used for this master thesis project, they are categorized by which type of media they belong to (i.e. the categories are Internet, Literature and Publications).

9.1 Internet

A Literature Review on Reaction Time by Robert J. Kosinski. (2006).
Available: http://biae.clemson.edu/bpc/bp/Lab/110/reaction.htm
Accessed: March 30, 2007.

ActivePython 2.4 Documentation. (2006).
Available: http://aspn.activestate.com/ASPN/docs/ActivePython/2.4/python/ext/ext.html
Accessed: April 12, 2007.

GIS.com – What is GIS? (2007).
Available: http://www.gis.com/whatisgis/index.html
Accessed: March 28, 2007.

GPS Errors & Estimating Your Receiver's Accuracy. (2007).
Available: http://edu-observatory.org/gps/gps_accuracy.html
Accessed: March 27, 2007.

Hacking GPS homepage. (2006).
Available: http://www.hacking-gps.com/articles.php?url=2&id=200503281930
Accessed: March 27, 2007.

How a gyroscope works by Terry Pearson. (1999).
Available: http://www.gyroscopes.org/how.asp.
Accessed: March 27, 2007.

How gyroscopes work by Marshall Brain. (2007).
Available: http://www.howstuffworks.com/gyroscope.htm
Accessed: March 27, 2007.

Kowoma-GPS homepage. (2002-2005).
Available: http://www.kowoma.de/en/gps/
Accessed: March 27, 2007.

Lantmäteriet. (2007).
Available: http://www.lantmateriet.se
Accessed: April 11, 2007.

Matplotlib module. (2007).
Available: http://matplotlib.sourceforge.net
Accessed: March 28, 2007.

Microsoft Visual Studio 2005. (2007).
Available: http://msdn2.microsoft.com/sv-se/vstudio/default.aspx
Accessed: March 28, 2007.

Microstrain, Inc. (2006).
3DM-G communications protocol for Firmware 2.1.00 rev3.01. pp. 7.
Available: http://www.microstrain.com/manuals/3DM-G%20communications%20protocol%20for%20Firmware%202.1.00%20rev3.01.pdf
Accessed: March 27, 2007.

Nintendo homepage, overview section of the Nintendo Wii. (2007).
Available: http://www.nintendo.com/overviewwii
Accessed: March 27, 2007.

Numpy module. (2007).
Available: http://www.scipy.org/NumPy
Accessed: March 28, 2007.

The Python Programming Language. (2006).
Available: http://www.python.org
Accessed: March 28, 2007

The Swedish Road Administration. (2004).
Creating lines for roads.
Available: http://www.vv.se/templates/page3____8165.aspx
Accessed: March 29, 2007.

9.2 Literature

Ghezzi, C., Jazayeri, M. and Mandrioli, D. (1991).
Fundamentals of Software Engineering.
Prentice-Hall International, Inc.

Heath, Michael T. (2002).
Scientific Computing: An Introductory Survey, Second Edition.
The McGraw-Hill Companies, Inc.

Råde, L. and Westergren, B. (1998).
Mathematics Handbook for Science and Engineering.
Studentlitteratur, Lund.

Tengstrand, Anders. (2005).
Åtta kapitel om geometri.
Studentlitteratur, Lund.

Yates, Roy D. and Goodman, David J. (2005).
Probability and Stochastic Processes: A Friendly Introduction for Electrical and Computer Engineers, Second Edition.
John Wiley & Sons, Inc.

9.3 Publications

Barkhuus, L., Chalmers, M., Tennent, P., Hall, M., Marek Bell, M., Sherwood, S., and Brown, B. (2005).
Picking Pockets on the Lawn: The Development of Tactics and Strategies in a Mobile Game.

In Proceedings Ubicomp'05 - The Seventh International Conference on Ubiquitous Computing, Tokyo, LNCS 3660. pp 358-374.

Bichard, J., Brunnberg, L., Combetto, M., Gustafsson, A. and Juhlin, O. (2006).
Backseat Playgrounds: Pervasive Storytelling in Vast Location Based Games.
In Proceedings of the 5th International Conference on Entertainment Computing - ICEC 2006. Springer Verlag, pp. 117-122.
Available: http://www.tii.se/mobility/Files/BSPshort_OJ_060414.pdf
Accessed: March 27, 2007.

Brunnberg, L. and Hulterström, K. (2003).
Designing for physical interaction and contingent encounters in a mobile gaming situation.
Presented at the workshop on real world user interfaces, at MobileHCI'2003.
Available: http://www.tii.se/mobility/Files/BackseatGaming_mobileHCI.pdf
Accessed: April 12, 2007.

Brunnberg, L. and Juhlin, O. (2006).
Keep your eyes on the road and your finger on the trigger - Designing for mixed focus of attention in a mobile game for brief encounters.
In Proceedings of the 4th International Conference on Pervasive Computing. Springer Verlag, pp. 169-186.
Available: http://www.tii.se/mobility/Files/RoadRager_FINAL_060120.pdf
Accessed: March 27, 2007.

Brunnberg, L. and Juhlin, O. (2003).
Movement and Spatiality in a Gaming Situation - Boosting Mobile Computer Games with the Highway Experience.
In Proceedings of Interact'2003 - IFIP TC 13 International Conference on Human-Computer Interaction. IOS Press, pp 407-414.
Available: http://www.tii.se/mobility/Files/BSGFinal.pdf
Accessed: March 27, 2007.

Gustafsson, A., Bichard, J., Brunnberg, L., Juhlin, O. and Combetto, M. (2006).
Believable environments – Generating interactive storytelling in vast location based pervasive games.
In Proceedings of SIGCHI Advances in Computer Entertainment 2006. ACM Press. CD-ROM.
Available: http://www.tii.se/mobility/Files/BelievableEnvironment_060421.pdf
Accessed: March 27, 2007.

10. Appendices

Here you find appendices, short information that could feel superfluous in the ordinary text in the report.

10.1 Appendix A – Calculating a sector of a circle

The formula for such a calculation (where the circle sector = s):

$$s = b * radius \qquad \text{(Formula 10.1.1)}$$

A schematic figure (figure 10.1.2) over how the variables correspond to the circle geometry.

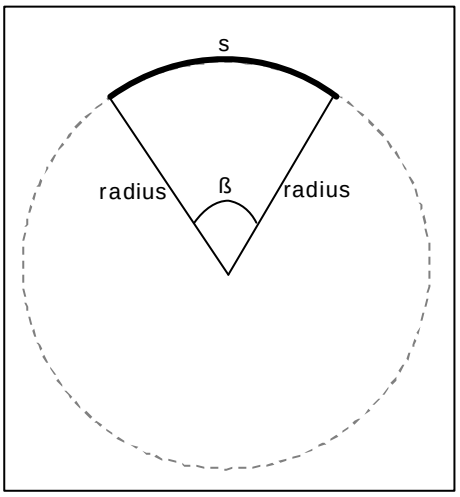


Figure 10.1.2: The variables used showed in a circle.

10.2 Appendix B – Maximum valid AngRates

To get the maximum valid AngRates of the Microstrain sensor module the following calculations has been made by using the situation in figure 10.2.1 and the result is found in the graphs in figure 10.2.4.

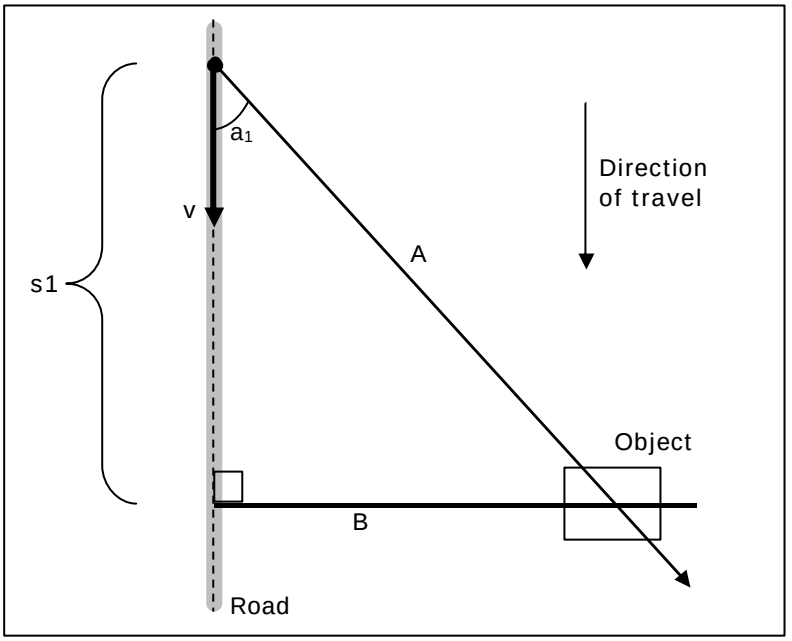


Figure 10.2.1: Explanatory image of the variables used when calculating maximum AngRate.

This figure gives the following relations:

$$\cos a_1 = \frac{s_1}{A} \Leftrightarrow a_1 = \cos^{-1}\left(\frac{s_1}{\sqrt{B^2 + s_1^2}}\right) \qquad \text{(Formula 10.2.2)}$$

By stating the fact b=B and assuming that a constant speed is kept, the distance s1 becomes s1 = (1/35)*v since the angles are taken with 1/35 of second apart as stated earlier. Therefore s1 = (1/35)*v, where v = velocity of the car. This gives formula 10.2.3:

$$\Rightarrow \cos^{-1}\left(\frac{\left(v\frac{1}{35}\right)}{\sqrt{B^2 + \left(v\frac{1}{35}\right)^2}}\right) = a_n = \frac{1}{35}\left(\sum_{i=1}^{n} AngRate_i\right) \qquad \text{(Formula 10.2.3)}$$

For the most common speed limits in Sweden the results become as shown in figure 10.2.4 with the distance between road and object as the x-axis.

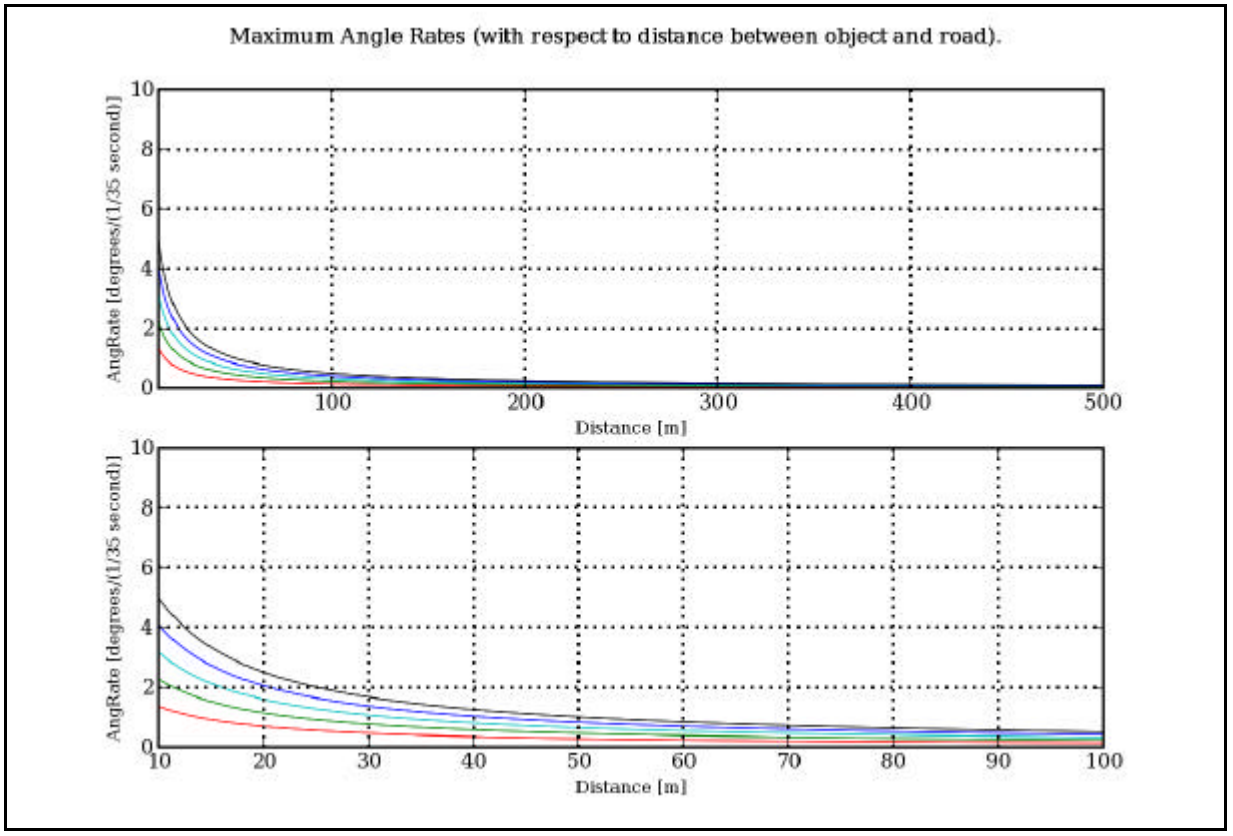


Figure 10.2.4: Resulting graphs showing the maximum AngRate at five different speeds (the lines show from top down the result for the speeds: 110, 90, 70, 50 and 30 km/h).

From these results it could be said that the maximum AngRate is most definitely dependent on where the object is situated and at which speed the player is traveling.

10.3 Appendix C – GPS Coordinate Transformation

To transform the GPS coordinates from WGS84 format (which is equivalent to SWEREF99; they only differ by a couple of decimeters) to RT90 format I am using the direct projection with the Gauss-Krüger formulas[33]. In Python this looks something like this:

```
import math
import cmath
from pylab import *
from Matrix import *
from scipy import *
from numpy import *

#Function for conversion from WGS84 (used in the GPS module in this project, is used with longitud and latitud)
# to RT90 (used in sweden, is metric)

def convertWGS84toRT90(lat, lon):
        #constants for the Gauss-Kruger Conformal Projection in RT90 2.5 gon V0:-15
        a = 6377397.155                                                 #Half great axis
        f = 1.0/299.1528128                                             #Flattening
        n = f/(2.0-f)
        longi_of_central_meridian = convertDegree(1548.22624306)        #lambda0
        scale_on_central_meridian = float(1.00000561024)                #k0
        FN = float(-667.711)                                            #False Northing
        FE = float(1500064.274)                                         #False Easting
        a_ceiling = (a/(1.0+n))*(1.0+(1.0/4.0)*pow(n,2)+(1.0/64.0)*pow(n,4))

        A = pow(e,2)
        B = (1.0/6.0)*((5.0)*pow(e,4)-pow(e,6))
        C = (1.0/120.0)*((104.0)*pow(e,6)-(45.0)*pow(e,8))
        D = (1.0/1260.0)*((1237.0)*pow(e,8))

        conformal_latitude = lat-convertDegree(sin(radians(lat))*cos(lat)*(
                A+B*pow(sin(radians(lat)),2)+C*pow(sin(radians(lat)),4)+D*pow(sin(radians(lat)),6)))

        sigma_longitude = lon-longi_of_central_meridian
        xi_prim = atan(tan(radians(conformal_latitude))/cos(radians(sigma_longitude)))
        eta_prim = real(cmath.atanh(cos(radians(conformal_latitude))*sin(radians(sigma_longitude))))

        beta1 = 0.5*n-(2.0/3.0)*pow(n,2)+(5.0/16.0)*pow(n,3)+(41.0/180.0)*pow(n,4)
        beta2 = (13.0/48.0)*pow(n,2)-(3.0/5.0)*pow(n,3)+(557.0/1440.0)*pow(n,4)
        beta3 = (61.0/240.0)*pow(n,3)-(103.0/140.0)*pow(n,4)
        beta4 = (49561.0/161280.0)*pow(n,4)

        RT90_x = scale_on_central_meridian * a_ceiling * (xi_prim + beta1*sin(2.0*xi_prim)*cosh(2.0*eta_prim) +
                beta2*sin(4.0*xi_prim)*cosh(4.0*eta_prim) + beta3*sin(6.0*xi_prim)*cosh(6.0*eta_prim) +
                beta4*sin(8.0*xi_prim)*cosh(8.0*eta_prim)) + FN
        RT90_y = scale_on_central_meridian * a_ceiling * (eta_prim + beta1*cos(2.0*xi_prim)*sinh(2.0*eta_prim)
                + beta2*cos(4.0*xi_prim)*sinh(4.0*eta_prim) + beta3*cos(6.0*xi_prim)*sinh(6.0*eta_prim) +
                beta4*cos(8.0*xi_prim)*sinh(8.0*eta_prim)) + FE
        RT90_xy = matrix([[RT90_x],[RT90_y]])
```

Figure 10.3.1: Python code for transforming GPS positions in WGS84 format to RT90 format (which is better suited for Sweden).

[33] The Gauss Conformal Projection (also known as the Krüger-formulas or Transverser Mercator) are found at: http://www.lantmateriet.se/upload/filer/kartor/geodesi_gps_och_detaljmatning/geodesi/Formelsaml%EDng/Gauss_Conformal_Projection.pdf.

10.4 Appendix D – The OSREP Testing Feedback

Here I show the resulting feedback (in figures 10.4.1 and 10.4.2) from running tests with the OSREP algorithm applied to direction data. This feedback corresponds to the graphs 6.4.2.3 and 6.4.2.4.

```
#########################################################################

### Start Time for test is: 14:50:27 04/18/07 W. Europe Standard Time###

#########################################################################

#####################################################################

Trying to OSREP fileNR: 1, with offset: 30.0

#####################################################################

Number of valid angles: 0 and current number of POIs are: 0
Number of valid angles: 0 and current number of POIs are: 0
Number of valid angles: 0 and current number of POIs are: 0
Number of valid angles: 0 and current number of POIs are: 0
Number of valid angles: 0 and current number of POIs are: 0
Number of valid angles: 0 and current number of POIs are: 0
Number of valid angles: 0 and current number of POIs are: 0
Number of valid angles: 0 and current number of POIs are: 0
Number of valid angles: 290 and current number of POIs are: 96
Number of valid angles: 364 and current number of POIs are: 229
Number of valid angles: 235 and current number of POIs are: 349
Number of valid angles: 171 and current number of POIs are: 440
Number of valid angles: 256 and current number of POIs are: 628
Number of valid angles: 233 and current number of POIs are: 790
Number of valid angles: 317 and current number of POIs are: 1087
Number of valid angles: 364 and current number of POIs are: 1523
Number of valid angles: 216 and current number of POIs are: 1875
Number of valid angles: 302 and current number of POIs are: 2338
Number of valid angles: 189 and current number of POIs are: 2709
Number of valid angles: 268 and current number of POIs are: 3495
Number of valid angles: 257 and current number of POIs are: 4449


The total number of valid angles are: 3462 and the total number of POIs are: 4449
while traveling WEST and the object is on the LEFT side of the car.
Weighing the set of POIs gathered...
The amount of POIs have been reduced to 2 valid POIs.

Distance B Between the Pinpointed Object and the Road is: 94.6916128411

Finished OSREPing fileNR: 1 with offset: 30.0.

#####################################################################

#####################################################################

Plotting OSREP Results for Test Run Nr: 1...

Finished Plotting OSREP for Test Run Nr:1

The OSREP Coordinates of the Pinpointed Object are: (x , y) = (6588654 , 1651275.27958)

Calulated Radius of the Pinpointed Object: 9.0

#####################################################################

Finished OSREPing log file Nr: 1 in 90.5299999714 seconds.
#########################################################################
```

Figure 10.4.1: Code Feedback: Without Button State Interval.

```
#########################################################################

### Start Time for test is: 14:55:05 04/18/07 W. Europe Standard Time###

#########################################################################

#########################################################################

Trying to OSREP fileNR: 1, with offset: 30.0

#########################################################################

Using BS interval...
Number of valid angles: 0 and current number of POIs are: 0
Number of valid angles: 0 and current number of POIs are: 0
Number of valid angles: 0 and current number of POIs are: 0
Number of valid angles: 0 and current number of POIs are: 0
Number of valid angles: 0 and current number of POIs are: 0
Number of valid angles: 0 and current number of POIs are: 0
Number of valid angles: 0 and current number of POIs are: 0
Number of valid angles: 0 and current number of POIs are: 0
Number of valid angles: 290 and current number of POIs are: 96
Number of valid angles: 364 and current number of POIs are: 229
Number of valid angles: 235 and current number of POIs are: 349
Number of valid angles: 171 and current number of POIs are: 440
Number of valid angles: 256 and current number of POIs are: 628
Number of valid angles: 233 and current number of POIs are: 790
Number of valid angles: 317 and current number of POIs are: 1087
Number of valid angles: 364 and current number of POIs are: 1523
Number of valid angles: 216 and current number of POIs are: 1875
Number of valid angles: 302 and current number of POIs are: 2338
Number of valid angles: 189 and current number of POIs are: 2709
Number of valid angles: 268 and current number of POIs are: 3495
Number of valid angles: 257 and current number of POIs are: 4449


The total number of valid angles are: 3462 and the total number of POIs are: 4449
while traveling WEST and the object is on the LEFT side of the car.
Weighing the set of POIs gathered...
The amount of POIs have been reduced to 2 valid POIs.

Distance B Between the Pinpointed Object and the Road is: 94.6916128411

Finished OSREPing fileNR: 1 with offset: 30.0.

#########################################################################

#########################################################################

Plotting OSREP Results for Test Run Nr: 1...

Finished Plotting OSREP for Test Run Nr:1

The OSREP Coordinates of the Pinpointed Object are: (x , y) = (6588654 , 1651275.27958)

Calulated Radius of the Pinpointed Object: 9.0

#########################################################################

Finished OSREPing log file Nr: 1 in 96.4789998531 seconds.
#########################################################################
```

Figure 10.4.2: Code Feedback: With Button State Interval.

10.5 Appendix E – The Test Run Notes

These are the notes taken from the last test run, shown in table 10.5.1. They are present in this report since I have referred to them so many times I feel compelled to do so. Test Nr 1, 8 and 11 was the attempt to pinpoint the coordinate showed in figure 5.2.1

Test Nr:	Side of the road where the object laid. (Left / Right)	Estimated distance (Close 0-50, Average 50-150, Far 150-200 meters)	Object	Travel direction
0	*Right*	*Close*	*Transformer housing.*	*North*
1	*Left*	*Average*	*Gate posts.*	*West*
2	*Left*	*Average*	*White house on little hill.*	*West*
3	*Left*	*Close- Average*	*White house by the dog field.*	*West*
4	*Left*	*Average*	*Yellow house, aimed while going through a curve.*	*West*
5	*Right*	*Average*	*White house, free from any other obstacles or objects.*	*East*
6	*Right*	*Average*	*Red house, the one closest to the field.*	*East*
7	*Right*	*Far-Far Away*	*Chimney (marked as "torn" in the GIS data).*	*East*
8	*Right*	*Average*	*Gate posts.*	*East*
9	*Left*	*Close*	*Transformer housing.*	*South*
10	*Right*	*Close*	*Transformer housing.*	*North*
11	*Left*	*Average*	*Gate posts.*	*West*
12	*Right*	*Close*	*Yellow house.*	*North-West*
13	*Left*	*Average*	*House.*	*West*
14	*Left*	*Average*	*Red barn, free standing by a field.*	*West*

Table 10.5.1: Notes taken during 15 test runs at Lidingö.

10.6 Appendix F – Code for the second OSREP Algorithm

In figure 10.6.1, here below, I present the python code for the second OSREP algorithm.

```
#-------------------------------------------------------------------------------------------------------------
# The OSREP Algorithm:
#-------------------------------------------------------------------------------------------------------------

def applyOSREPalgorithm(gps_x_list, gps_y_list, gps_time_list, gps_speed_list, dir_angRatez_list, dir_time_list, offset,
object_size_tolerance, line_of_sight_distance, bs_list, bs_time_list, BS_interval_indicator, verbose_mode):
    #OSREP init
    break_status = False
    POI_x_list = []
    POI_y_list = []
    POI_list = [[],[]]
    POO_dir1_list = [[],[]]
    POO_dir2_list = [[],[]]
    theta1_init = offset
    theta2_init = offset
    theta1_sum = float(0.0)
    theta2_sum = float(0.0)
    delta_theta_sum = float(0.0)
    poi_counter = 0
    travel_westing = 0
    travel_northing = 0
    total_amount_of_angle = 0

    #Check the button state interval
    if BS_interval_indicator == int(1):
        start = gps_time_list[0]
        end = gps_time_list[len(gps_time_list)-1]
        found_start = False
        found_end = False
        for p in range(0,len(bs_list)):
            if bs_list[p] == int(1) and found_start == False:
                BS_START = bs_list[p]
                for time in range(gps_time_list[0],gps_time_list[len(gps_time_list)-1]):
                    if (BS_START-time)<=int(1000) and (BS_START-time)>=int(400):
                        start = gps_time_list[gps_time_list.index(BS_START)]
                found_start = True
            if bs_list[p] == int(0) and found_end == False:
                BS_END = bs_list[p]
                for time in range(gps_time_list[0],gps_time_list[len(gps_time_list)-1]):
                    if (BS_END-time)<=int(1000) and (BS_END-time)>=int(400):
                        end = gps_time_list[gps_time_list.index(BS_END)]
                found_end = True
        if found_start == True and found_end == True:
            if verbose_mode == True:
                print 'Using BS interval...'

    #Selecting two pairs of GPS positions
    for n in range(1, len(gps_time_list),1):
        angle_counter = 0
        #GPS position pair 1
        gps1_x = gps_x_list[n-1]
        gps1_y = gps_y_list[n-1]
        gps1_timestamp = gps_time_list[n-1]
        gps1_speed = gps_speed_list[n-1]
        gps2_x = gps_x_list[n]
        gps2_y = gps_y_list[n]
        gps2_timestamp = gps_time_list[n]
        gps2_speed = gps_speed_list[n]
        #GPS position pair 2
        gps3_x = gps_x_list[len(gps_time_list)-n-1]
        gps3_y = gps_y_list[len(gps_time_list)-n-1]
        gps3_timestamp = gps_time_list[len(gps_time_list)-n-1]
        gps3_speed = gps_speed_list[len(gps_time_list)-n-1]
        gps4_x = gps_x_list[len(gps_time_list)-n]
        gps4_y = gps_y_list[len(gps_time_list)-n]
        gps4_timestamp = gps_time_list[len(gps_time_list)-n]
        gps4_speed = gps_speed_list[len(gps_time_list)-n]
```

```
    #Calculate direction of travel
    if gps1_x < gps4_x:
        travel_westing = -1 #going east
    else:
        travel_westing = 1 #going west
    if gps1_y < gps4_y:
        travel_northing = -1 #going south
    else:
        travel_northing = 1 #going north

    #Calculate valid time interval between GPS position 1 and 4, in milliseconds
    valid_time_interval = 2.0*(line_of_sight_distance/((gps1_speed+gps4_speed)/2.0))*1000.0
    if (gps4_timestamp-gps1_timestamp) <= valid_time_interval:
        if BS_interval_indicator == int(1):
            if found_start == True and found_end == True:
                if gps1_timestamp >= start and gps4_timestamp <= end:
                    #Distances between GPS positions within the pairs
                    ls1 = sqrt(pow((gps1_x-gps2_x),2)+pow((gps1_y-gps2_y),2))
                    ls2 = sqrt(pow((gps3_x-gps4_x),2)+pow((gps3_y-gps4_y),2))

                    #Select directions
                    dir1_angRatez_list = []
                    dir1_time_list = []
                    dir2_angRatez_list = []
                    dir2_time_list = []

                    for i in range(0,len(dir_angRatez_list)):
                        #Select the directions which lays in between GPS positions 1 and 2. (I.e. the directions in a time interval)
                        if dir_time_list[i] >= gps1_timestamp and dir_time_list[i] <= gps2_timestamp:
                            dir1_angRatez_list.append(dir_angRatez_list[i])
                            dir1_time_list.append(dir_time_list[i])
                        #Select the directions which lays in between GPS positions 3 and 4. (I.e. the directions in a time interval)
                        if dir_time_list[i] >= gps3_timestamp and dir_time_list[i] <= gps4_timestamp:
                            dir2_angRatez_list.append(dir_angRatez_list[i])
                            dir2_time_list.append(dir_time_list[i])

                    theta1 = theta1_init
                    theta2 = theta2_init
                    for k in range(1,len(dir1_time_list)-1):
                        #Direction Angles modulus 360 degrees
                        theta1 = fmod(theta1 + (dir1_angRatez_list[k] * 180.0/pi) * (dir1_time_list[k] - dir1_time_list[k-1]), 360.0)
                        theta1_sum = fabs(theta1_sum + fmod((dir1_angRatez_list[k] * 180.0/pi) * (dir1_time_list[k] - dir1_time_list[k-1]), 360.0))

                        #Direction's progress between GPS positions 1 and 2
                        dir1_ratio = (dir1_time_list[k]-gps1_timestamp)/(gps2_timestamp-gps1_timestamp)

                        #Point-Of-Origin of the direction
                        s1 = ls1-dir1_ratio*ls1
                        (POO_dir1_x, POO_dir1_y) = ((((dir1_ratio * ls1)*gps2_x + s1*gps1_x)/(dir1_ratio + s1)), (((dir1_ratio * ls1)*gps2_y +
s1*gps1_y)/(dir1_ratio + s1)))

                        for l in range(1,len(dir2_time_list)-1):
                            #Direction Angles modulus 360 degrees
                            theta2 = fmod(theta2 + (dir2_angRatez_list[l] * 180.0/pi) * (dir2_time_list[l] - dir2_time_list[l-1]), 360.0)
                            theta2_sum = fabs(theta2_sum + fmod((dir2_angRatez_list[l] * 180.0/pi) * (dir2_time_list[l] - dir2_time_list[l-1]),
360.0))/l

                            if ((180-theta2)+theta1)<=float(170.0) and ((180-theta2)+theta1)>=float(2.0):
                                angle_counter = angle_counter + 1
                                #Direction's progress between GPS positions 3 and 4
                                dir2_ratio = (dir2_time_list[l]-gps3_timestamp)/(gps4_timestamp-gps3_timestamp)

                                #Point-Of-Origin of the direction
                                s2 = ls2-dir2_ratio*ls2
                                (POO_dir2_x, POO_dir2_y) = ((((dir2_ratio * ls2)*gps4_x + s2*gps3_x)/(dir2_ratio + s2)), (((dir2_ratio * ls2)*gps4_y
+ s2*gps3_y)/(dir2_ratio + s2)))

                                if POO_dir1_x > POO_dir2_x:
                                    poo_x_max = POO_dir1_x + line_of_sight_distance
                                    poo_x_min = POO_dir2_x - line_of_sight_distance
                                else:
                                    poo_x_max = POO_dir2_x + line_of_sight_distance
                                    poo_x_min = POO_dir1_x - line_of_sight_distance
```

```
                        #Calculate Point-Of-Intersections
                        x = float(poo_x_min)
                        while x >= poo_x_min and x <= poo_x_max:
                            y1 = x*tan(theta1) + POO_dir1_y - POO_dir1_x*tan(theta1)
                            y2 = x*tan(theta2) + POO_dir2_y - POO_dir2_x*tan(theta2)

                            if y1 <= (y2+object_size_tolerance) and y1 >= (y2-object_size_tolerance):
                                if (checkPOIdistanceToPOOs(x, y1, POO_dir1_x, POO_dir1_y, POO_dir2_x, POO_dir2_y,
line_of_sight_distance) == True) and (checkPOIdistanceToPOOs(x, y2, POO_dir1_x, POO_dir1_y, POO_dir2_x, POO_dir2_y,
line_of_sight_distance) == True):
                                    POI_list[0].append(x)
                                    POI_list[1].append(y1)
                                    POO_dir1_list[0].append(POO_dir1_x)
                                    POO_dir1_list[1].append(POO_dir1_y)
                                    POO_dir2_list[0].append(POO_dir2_x)
                                    POO_dir2_list[1].append(POO_dir2_y)
                                    poi_counter = poi_counter + 1
                            x = x + 0.5
                    delta_theta_sum = theta1_sum/k - theta2_sum
            elif BS_interval_indicator == int(0):
                #Distances between GPS positions within the pairs
                ls1 = sqrt(pow((gps1_x-gps2_x),2)+pow((gps1_y-gps2_y),2))
                ls2 = sqrt(pow((gps3_x-gps4_x),2)+pow((gps3_y-gps4_y),2))

                #Select directions
                dir1_angRatez_list = []
                dir1_time_list = []
                dir2_angRatez_list = []
                dir2_time_list = []

                for i in range(0,len(dir_angRatez_list)):
                    #Select the directions which lays in between GPS positions 1 and 2. (I.e. the directions in a time interval)
                    if dir_time_list[i] >= gps1_timestamp and dir_time_list[i] <= gps2_timestamp:
                        dir1_angRatez_list.append(dir_angRatez_list[i])
                        dir1_time_list.append(dir_time_list[i])
                    #Select the directions which lays in between GPS positions 3 and 4. (I.e. the directions in a time interval)
                    if dir_time_list[i] >= gps3_timestamp and dir_time_list[i] <= gps4_timestamp:
                        dir2_angRatez_list.append(dir_angRatez_list[i])
                        dir2_time_list.append(dir_time_list[i])

                theta1 = theta1_init
                theta2 = theta2_init
                for k in range(1,len(dir1_time_list)-1):
                    #Direction Angles modulus 360 degrees
                    theta1 = fmod(theta1 + (dir1_angRatez_list[k] * 180.0/pi) * (dir1_time_list[k] - dir1_time_list[k-1]), 360.0)
                    theta1_sum = fabs(theta1_sum + fmod((dir1_angRatez_list[k] * 180.0/pi) * (dir1_time_list[k] - dir1_time_list[k-1]), 360.0))

                    #Direction's progress between GPS positions 1 and 2
                    dir1_ratio = (dir1_time_list[k]-gps1_timestamp)/(gps2_timestamp-gps1_timestamp)

                    #Point-Of-Origin of the direction
                    s1 = ls1-dir1_ratio*ls1
                    (POO_dir1_x, POO_dir1_y) = ((((dir1_ratio * ls1)*gps2_x + s1*gps1_x)/(dir1_ratio + s1)), (((dir1_ratio * ls1)*gps2_y +
s1*gps1_y)/(dir1_ratio + s1)))

                    for l in range(1,len(dir2_time_list)-1):
                        #Direction Angles modulus 360 degrees
                        theta2 = fmod(theta2 + (dir2_angRatez_list[l] * 180.0/pi) * (dir2_time_list[l] - dir2_time_list[l-1]), 360.0)
                        theta2_sum = fabs(theta2_sum + fmod((dir2_angRatez_list[l] * 180.0/pi) * (dir2_time_list[l] - dir2_time_list[l-1]), 360.0))/l

                        if ((180-theta2)+theta1)<=float(170.0) and ((180-theta2)+theta1)>=float(2.0):
                            angle_counter = angle_counter + 1
                            #Direction's progress between GPS positions 3 and 4
                            dir2_ratio = (dir2_time_list[l]-gps3_timestamp)/(gps4_timestamp-gps3_timestamp)

                            #Point-Of-Origin of the direction
                            s2 = ls2-dir2_ratio*ls2
                            (POO_dir2_x, POO_dir2_y) = ((((dir2_ratio * ls2)*gps4_x + s2*gps3_x)/(dir2_ratio + s2)), (((dir2_ratio * ls2)*gps4_y +
s2*gps3_y)/(dir2_ratio + s2)))

                            if POO_dir1_x > POO_dir2_x:
                                poo_x_max = int(POO_dir1_x + line_of_sight_distance)
                                poo_x_min = int(POO_dir2_x - line_of_sight_distance)
```

```
                    else:
                        poo_x_max = int(POO_dir2_x + line_of_sight_distance)
                        poo_x_min = int(POO_dir1_x - line_of_sight_distance)

                    #Calculate Point-Of-Intersections
                    x = float(poo_x_min)
                    while x >= poo_x_min and x <= poo_x_max:
                        y1 = x*tan(theta1) + POO_dir1_y - POO_dir1_x*tan(theta1)
                        y2 = x*tan(theta2) + POO_dir2_y - POO_dir2_x*tan(theta2)

                        if y1 < (y2+object_size_tolerance) and y1 > (y2-object_size_tolerance):
                            if (checkPOIdistanceToPOOs(x, y1, POO_dir1_x, POO_dir1_y, POO_dir2_x, POO_dir2_y, line_of_sight_distance)
== True) and (checkPOIdistanceToPOOs(x, y2, POO_dir1_x, POO_dir1_y, POO_dir2_x, POO_dir2_y, line_of_sight_distance) == True):
                                POI_list[0].append(x)
                                POI_list[1].append(y1)
                                POO_dir1_list[0].append(POO_dir1_x)
                                POO_dir1_list[1].append(POO_dir1_y)
                                POO_dir2_list[0].append(POO_dir2_x)
                                POO_dir2_list[1].append(POO_dir2_y)
                                poi_counter = poi_counter + 1
                        x = x + 0.5
            delta_theta_sum = theta1_sum/k - theta2_sum

    total_amount_of_angle = total_amount_of_angle + angle_counter
    if verbose_mode == True:
        print 'Number of valid angles: ' + str(angle_counter) + ' and current number of  POIs are: ' + str(poi_counter)

    #If gps2_x == gps3_x and gps2_y == gps3_y then all calculations needed has been carried out.
    # Continuing beyond will be to calculate the same POIs one more time, which is unnecessary :)
    if gps2_x == gps3_x and gps2_y == gps3_y: #even gps positions
        if verbose_mode == True:
            print '\n'
        break_status = True
        break
    elif gps1_x == gps3_x and gps1_y == gps3_y: #uneven gps positions
        if verbose_mode == True:
            print '\n'
        break_status = True
        break

  if break_status == True:
    if verbose_mode == True:
        print 'The total number of valid angles are: ' + str(total_amount_of_angle) + ' and the total number of  POIs are: ' + str(poi_counter)
    valid_POI_list = [[],[]]
    if delta_theta_sum > 0.0 and travel_westing == 1: #going east and pinpointing object on left.
        for poi_element in range(0, len(POI_list[0])-1):
            #POI on the left side
            if POI_list[1][poi_element] > POO_dir1_list[1][poi_element] and POI_list[1][poi_element] > POO_dir2_list[1][poi_element]:
                POI_x_list.append(POI_list[0][poi_element])
                POI_y_list.append(POI_list[1][poi_element])
        if verbose_mode == True:
            print 'while traveling EAST and the object is on the LEFT side of the car.'
    elif delta_theta_sum < 0.0 and travel_westing == 1: #going east and pinpointing object on right.
        for poi_element in range(0, len(POI_list[0])-1):
            #POI on the right side
            if POI_list[1][poi_element] < POO_dir1_list[1][poi_element] and POI_list[1][poi_element] < POO_dir2_list[1][poi_element]:
                POI_x_list.append(POI_list[0][poi_element])
                POI_y_list.append(POI_list[1][poi_element])
        if verbose_mode == True:
            print 'while traveling EAST and the object is on the RIGHT side of the car.'
    elif delta_theta_sum > 0.0 and travel_westing == -1: #going west and pinpointing object on left.
        for poi_element in range(0, len(POI_list[0])-1):
            #POI on the left side
            if POI_list[1][poi_element] < POO_dir1_list[1][poi_element] and POI_list[1][poi_element] < POO_dir2_list[1][poi_element]:
                POI_x_list.append(POI_list[0][poi_element])
                POI_y_list.append(POI_list[1][poi_element])
        if verbose_mode == True:
            print 'while traveling WEST and the object is on the LEFT side of the car.'
    elif delta_theta_sum < 0.0 and travel_westing == -1: #going west and pinpointing object on right.
        for poi_element in range(0, len(POI_list[0])-1):
            #POI on the right side
            if POI_list[1][poi_element] > POO_dir1_list[1][poi_element] and POI_list[1][poi_element] > POO_dir2_list[1][poi_element]:
                POI_x_list.append(POI_list[0][poi_element])
                POI_y_list.append(POI_list[1][poi_element])
```

```
            if verbose_mode == True:
                print 'while traveling WEST and the object is on the RIGHT side of the car.'
        if verbose_mode == True:
            print 'Weighing the set of POIs gathered...'
            (valid_POI_list[0], valid_POI_list[1], counter) = checkPOIconcentrations(POI_x_list, POI_y_list, object_size_tolerance)
        if verbose_mode == True:
            print 'The amount of POIs have been reduced to ' + str(counter) + ' valid POIs.\n'
        old_distance_to_road = float(99999999999.999999) #init
        distance_to_road = float(99999999999.999999) #init
        for poo_elements in range(0,len(POO_dir1_list)-1):
            distance_to_road1 = sqrt(pow((valid_POI_list[0][0]-POO_dir1_list[0][poo_elements]),2)+pow((valid_POI_list[1][0]-
POO_dir1_list[1][poo_elements]),2))
            distance_to_road2 = sqrt(pow((valid_POI_list[0][0]-POO_dir2_list[0][poo_elements]),2)+pow((valid_POI_list[1][0]-
POO_dir2_list[1][poo_elements]),2))
            if distance_to_road1 <= distance_to_road2 and distance_to_road1 <=old_distance_to_road:
                distance_to_road = distance_to_road1
                old_distance_to_road = distance_to_road
            elif distance_to_road2 < distance_to_road1 and distance_to_road2 <=old_distance_to_road:
                distance_to_road = distance_to_road2
                old_distance_to_road = distance_to_road
        print 'Distance B Between the Pinpointed Object and the Road is: ' + str(distance_to_road) + '\n'
        return (valid_POI_list[0], valid_POI_list[1])
    else:
        return ([],[])


#Check if POI is sufficiently close to the Point-Of-Origins (POO_dir1 and POO_dir2).
def checkPOIdistanceToPOOs(poi_x, poi_y, POO_dir1_x, POO_dir1_y, POO_dir2_x, POO_dir2_y, line_of_sight_distance):
    distance1 = sqrt(pow((poi_x-POO_dir1_x),2)+pow((poi_y-POO_dir1_y),2))
    distance2 = sqrt(pow((poi_x-POO_dir2_x),2)+pow((poi_y-POO_dir2_y),2))
    if distance1 <= line_of_sight_distance and distance2 <= line_of_sight_distance:
        return True
    else:
        return False


#Check POI concentrations
def checkPOIconcentrations(POI_x_list, POI_y_list, object_size_tolerance):
    if POI_x_list != [] and POI_y_list != []:
        x_list = []
        y_list = []
        valid_POI_list = [[],[],[]] #[[x],[y],[number of valid neighbours]]
        for n in range(0,len(POI_x_list)-1):
            valid_status = False
            hit_counter = 0
            for m in range(n,len(POI_x_list)-1):
                if POI_x_list[n] != POI_x_list[m] and POI_y_list[n] != POI_y_list[m]:
                    d_between_POIs = sqrt(pow((POI_x_list[n]-POI_x_list[m]),2)+pow((POI_y_list[n]-POI_y_list[m]),2))
                    if d_between_POIs <= object_size_tolerance:
                        if POI_x_list[n] in valid_POI_list[0]:
                            hit_counter = valid_POI_list[2][valid_POI_list[0].index(POI_x_list[n])] + 1
                            valid_POI_list[2][valid_POI_list[0].index(POI_x_list[n])] = hit_counter
                        else:
                            valid_POI_list[0].append(POI_x_list[n])
                            valid_POI_list[1].append(POI_y_list[n])
                            valid_POI_list[2].append(1)
                            valid_POI_list[0].append(POI_x_list[m])
                            valid_POI_list[1].append(POI_y_list[m])
                            valid_POI_list[2].append(1)

        x_list.append(valid_POI_list[0][valid_POI_list[2].index(max(valid_POI_list[2]))])
        y_list.append(valid_POI_list[1][valid_POI_list[2].index(max(valid_POI_list[2]))])
        x_list.append(valid_POI_list[0][valid_POI_list[2].index(max(valid_POI_list[2]))-1])
        y_list.append(valid_POI_list[1][valid_POI_list[2].index(max(valid_POI_list[2]))-1])
        return (x_list, y_list, 2)
    else:
        return ([],[],0)
```

Figure 10.6.1: The code for the second OSREP algorithm.