University College London

Department of Computer Science

MASTERS THESIS

Automatic Collision Avoidance for Virtual Characters using a Navigation Model

MSc Computer Graphics, Vision, and Imaging
Takatoshi Ono

Supervisor: Dr. Oyewole Oyekoya

This report is submitted as part requirement for the MSc Degreee in Computer Graphics, Vision and Imaging at University College London. It is substantially the result of my own work except where explicitly indicated in the text.

September 2011

Abstract

Recent developments in the field of computer graphics have received renewed interest in real time processing for video and online games leading to a more realistic virtual world. However, autonomous behavior of virtual character still lacks realism despite numerous research over the years. Considerable number of research have been published to make plausible behaviors. One of the research developed was an "Attention model" that allocated attention to an avatar in Second Life (SL), which is widely used for virtual world entertainment. Consequently, the Second Life avatar selected an interesting object automatically from the scene and then navigated towards the direction, but it only walked straight because did not adopt a functionality for collision avoidance. Therefore, this paper considers the issues of collision avoidance in order to implement more believable behavior, and proposes a "Navigation model" which computes collision free path for the SL avatar. In order to resolve the issues, various methods within the literature, from robotics to computer animation field, are reviewed and this research adopts the Hybrid Reciprocal Velocity Obstacles (HRVO) algorithm. Navigation model is succeeded by the integration of HRVO algorithm with Second Life source code. The navigation model-controlled avatar is evaluated by two types of experiments: (i) the plausibility of the collision free path planning and (ii) the processing speed. Results of the evaluation experiments demonstrated a navigation model with promising humanoid path planning in simple scenes as well as complex environments. This work increases the realism of autonomous virtual character.

Acknowledgements

I am deeply grateful to my supervisor, Dr Wole Oyekoya for all his guidance and support from the preliminary to the concluding level enabled me to develop an understanding of the subject. I would also like to thank my friends who participated on my experiments. Lastly, I am grateful to my family and my girlfriend for their support over the whole year of my masters.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

In the last few decades, virtual environment technology has been developed rapidly being led by internet and video games, and the developments of computer graphics technology have also provided high quality graphics even in real time processing. This development has given a considerable number of people a deeper sense of immersion within the 3D virtual enviroments through online and video games. These video games and virtual worlds usually provide a game character (avatar) as a graphical representation to reflect user's action, and the user's actions are also affected by the avatar's attitude on the other hand. However, the realism of an avatar's behavior is still insufficient for improving user immersion compared to improving graphic quality. It is becoming increasingly difficult to ignore the issues for producing realistic behavior of avatar. Thus, in order to improve the user immersion, the field of virtual environment requires the development of avatar's realistic behavior.

One of the most popular platform for virtual environments is Second Life[1] published by Linden Lab [2] in 2003, and anyone can download the Second Life viewer[3] to explore the world. This is a part of Massively Multiplayer Online Games (MMOG), but the main difference from normal MMOG is that the user of Second Life does not need to have specific purpose [28]. Each user plays having a different purpose, such as playing small games, meeting someone and having a conference, hence what a user does is entrusted to the user's decision using mouse and keyboard control. In this large scale simulator which consists of approximately two billion square meters area [32], the Second Life world is constructed by many buildings and objects just as actual world, and 11 million people have registered to live in this world [14]. A user is represented by an avatar who resides in the virtual world and can explore the world freely by walking, running, flying and teleporting, chatting with another user, building objects and selling these objects. These game characteristics make the user feel a sense of presence as a human in this virtual world and allow user to immerse into the game. However, users tend not to feel a humanoid behavious from the avatar within the virtual world due to a low level of realism of the avatars. In order to improve virtual characters behavior, Kokkinara [14] researched virtual character's attention

[1] http://secondlife.com (accessed 31 Aug 2011)
[2] http:lindenlab.com/ (accessed 31 Aug 2011)
[3] http://secondlife.com/support/downloads/ (accessed 31 Aug 2011)

and provided “Attention model”.

Human’s attention is influenced by the surrounding environments. When the attention is influenced, human moves its eyes, head or body to attend to the salient object. However, real time virtual character usually looks at one direction or move the head regularly in any scene, and these impassivenesses provide unrealistic behavior of avatar. Attention model gave a function of “attention” to the character. According to the features of scene, the head and gaze are controlled to make it look at interesting object, and as a result, the head and the eye animation of the virtual character shows humanoid behavior. Furthermore, Kokkinara integrated the attention model with Second Life in her project. Since the model is built in Second Life viewer, the avatar presents autonomous and plausible behavior which points the gaze to the most interesting object in the scene (Figure 1.1).

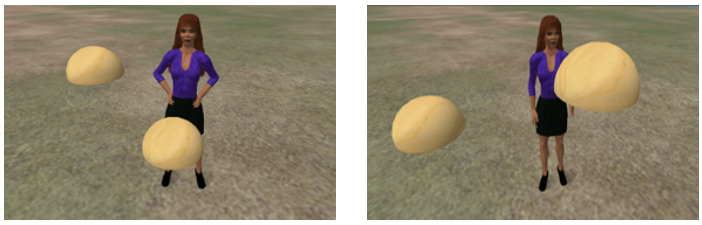


Figure 1.1: Avatar allocates attention to an object

In Second Life application, there is a function called “Autopilot”[4]. This function enables avatar to move toward a particular location or particular object without control by the user. In order to utilize the characteristics of attention model, Kokkinara [14] integrated this model with autopilot function, and named it “Autopilot mode”. The autopilot mode allowed avatar to move automatically towards an object which is chosen by the avatar itself according to the attention.

However, a major problem with this application is that autopilot mode only allows avatar to move in a straight line from current location to the target location, thus avatar’s locomotion is often prevented by obstacles from reaching its goals. This means that there is no difference between a random choice of target object and attention model choice, so the avatar does not show autonomous behavior. The simplest solution is to extend avatar action with collision avoidance. If the avatar chooses collision free path to the target object, user would recognize which object the avatar chose and feel the actual presence of avatar’s motion. It is therefore necessary to extend avatar’s functionality with collision avoidance in the locomotion.

Therefore, the main objective of this research is to develop a “Navigation model” which allows avatar to avoid collision with other obstacles in autopilot mode thus providing autonomous motion to interesting objects (i.e. goal).

[4]http://wiki.secondlife.com/wiki/AGENT _ AUTOPILOT

1.2 Structure of Dissertation

This paper is divided into further 4 chapters organized in following way.

Chapter 2 reviews previous literature in related areas, starting with path planning studies in robotics field, and followed by various research in computer animation field. A theoretical explanation of the preferred HRVO algorithm is also presented at the end of this chapter.

Chapter 3 describes the detail of navigation model. It begins with a study of the autopilot mode, introducing the Second Life world, and looks at the integration of the HRVO algorithm with Second Life application. Short tests are conducted to assess the performance of the model within the application.

Chapter 4 describes two experiments adopted to evaluate navigation model. The experimental design and analysis are given. A brief description of DTW algorithm, which is used for the analysis, is also presented in this chapter. Furthermore, one-way ANOVA is adopted to identify any significant differences between the paths generated by the navigation model and human participants.

Chapter 5 gives a discussion of the experiment results. The findings revealed from the experimental analysis are described here. After the discussion, this chapter concludes this paper with the review and limitation of the research, and suggests future work.

Chapter 2

Background Work

2.1 Path Planning

2.1.1 Robotics Field

In recent years, there has been an increasing amount of research on path planning in computer animation and robotics field. Research on path planning emerged during the late 1980s in the robotics field [2], motion planning [12], autonomous vehicles [11], and interactive humanoid robot [7] [24]. Collision avoidance in robot locomotion is a major topic of recent research, and it is difficult to avoid collision particularly under these unknown environments [6] [15] [16] [31]. Fox et al. [6] proposed dynamic window approach which is constructed by a set of reachable velocities for subsequent step, and the only velocities which are not prevented by obstacles in the space are chosen to advance. This approach showed increased speed of robot locomotion. In order to make a robot understand the configuration of environments, most research conducts learning processes while the robot moves around the static environments or moves to the previous target [16] [31]. Taylor and Kriegman [31] used image data captured from robot to recognize landmarks and obstacles in the environment during the exploration, and created the boundary place graph to compute collision free path.

A recent study by Lee et al. [16] attempted to investigate a humanoid path planning on robot. Their approach firstly assumed to divide the environment surface with 2D square grids including the start and goal position. Center of row lines connected start and goal point, and other horizontal and vertical lines are configured regularly. The sample points were taken at any intersected point of grids as virtual targets (Figure 2.1). The searching algorithm taken in this paper was similar to depth first search (DFS) algorithm. Firstly, the virtual point which is possible to move from start point was taken from first column, preferring a straight locomotion to goal position. If no virtual point was found from first column, robot tried to move horizontal direction, and conducted same search. This search simulation is repeated until goal point is detected in the virtual scene, and after the preliminary simulation, the robot follows these virtual points to reach the goal. It is possible to apply this method to the computer animation field because this process has also been simulated using the virtual space; however, this method was simulated only in the complex office type environment, and still has a weakness for moving objects. Even though,

during the past thirty years much more research has been done to solve this issue, it is still difficult to implement in complex and large environments.

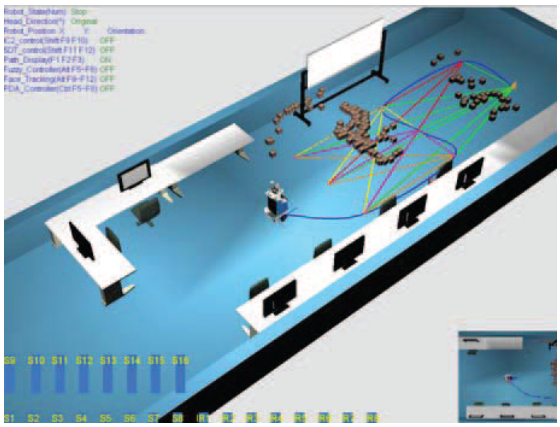


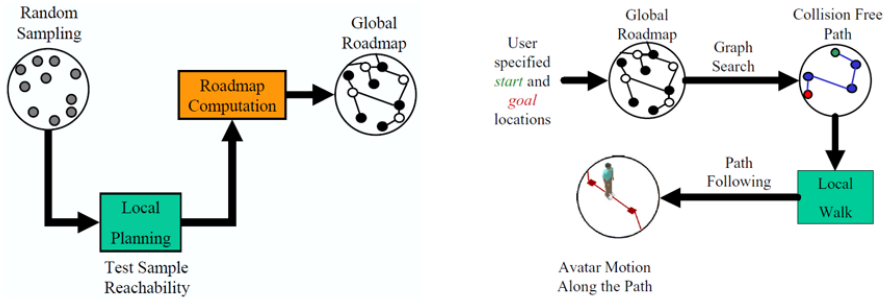
Figure 2.1: The path planning simulation and virtual point configured

2.1.2 Computer Animation Field

These collision free path planning algorithms derived by robotics studies have been applied to computer animation field. These studies are basically divided into avatar navigation and crowd simulation. As an avatar navigation approach, Cavazza [3] demonstrated a fundamental method to generate collision free path for humanoid character in virtual scene. He settled discretized grid in static scenes with single avatar and searched goal position by breadth first search (BFS) algorithm. These 2D grid sampling methods were considered to improve performance, and smooth roadmap has been generated instead of a grid of the scene in order to reduce computation [21]. Nowadays, the algorithm which uses probabilistic roadmap has become popular. Li and Ting [17] launched the initial idea of probabilistic roadmap as a randomized roadmap planner. This study was adapted to predict the next locomotion where an avatar moves to, and it showed successful result in simple environments.

This approach was applied to complex environments by Salomon et al. [26]. Firstly, this approach precomputed global roadmap using visibility based probabilistic roadmap (VisPRM). This algorithm was based on PRM algorithm extended by a condition of visibility. Just as PRM, it took a large amount of random samples but these were only taken from the visible area from the avatar. In order to generate collision free path, this model was divided into two processes, which is preprocessing phase and runtime phase. Figure 2.2 illustrates the overview of this two phases.

Preprocessing phase computed the global roadmap which represents any walkable paths for avatar. In order to compute global roadmap, avatar is treated as a point, and many sample points were taken randomly on any walkable surface. These virtual points are treated as nodes, and if avatar could move from one node to another node, these nodes were connected costing the distance. Runtime phase obtained start and goal points from user input, and calculated shortest path between start and goal points. The



(a) (b)

Figure 2.2: Preprocessing phase(a) and Runtime phase(b)

calculation of shortest path employed IDA* search algorithm, which uses iterative deepening depth-first search, to improve the performance. This study also considered the accidental collision. When avatar confronted with an obstacle, the locomotion is simply rearranged to move along the object surface.

This model was investigated in complex environments, and presented promising results. However, it is hard to think that the preliminary computation of global roadmap is an appropriate method, because it took 1.3 hours to compute global roadmap under an environment made up of 10143 polygons and 13.7 hours for 12541083 polygons. This computation may be thought to be time consuming. Furthermore, Salomon et al. mentioned that the collision free paths were not always natural-looking movement.

Furthermore, probabilistic roadmap algorithm has also been applied to multi agent scenes recently [30]. Probabilistic roadmap is a simple way to produce collision free path but it needed a long process time to generate plausible locomotion because of the sampling number, and it also has issues in dynamic scenes.

As research developed for motion planning of single agent, these studies led to crowd simulation of virtual cities. Michael and Chrysanthou [18] investigated agent locomotion in crowded street using density distributions of population which human choose a directions with more available surface to walk. Each agent system has been controlled by affordances and provided natural and smooth trajectories. This resulted in realistic agents' movements with high speed, but the issues were that each agent did not have an explicit target.

Navigation for multi agents who have specific objectives has been developed. Sud et al. [29] proposed the algorithm for multi-agent path planning in dynamic environment. In this algorithm, a new data structure named multi-agent navigation graph (MaNG) was constructed by first-order and second-order Volonoi diagrams to compute global roadmap. Treating all objects and agents as set of points, first and second-order Volonoi diagram were constructed (Figure 2.3 (a) and (b)), then MaNG was created by the integration of the two diagrams preferring the edges of objects region in first order Volonoi diagram (Figure 2.3 (c)). Thus, MaNG presented the path of maximal clearance for locomotive characters, and it

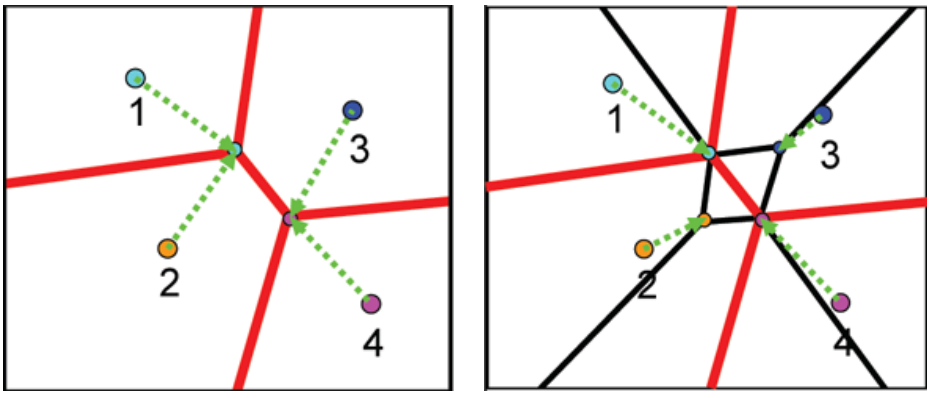
did not need a separate roadmap for each agent. This characteristics was effective for the performance of multi-agent path planning.

(a) (b) (c)

Figure 2.3: Volonoi diagrams and MaNG

After computation of MaNG, agents needed to be connected from agents position and MaNG edges in order to walk on the MaNG map, so virtual target points were settled on the nearest intersection on the path, and interpolation paths were created connecting agent position and each virtual target. After that, shortest path to goal position was calculated.

Instead of using only first-order Volonoi diagram, MaNG method was effective in avoiding collisions with other agents. Employing first order Volonoi diagram only would mean that the arrival time at the first virtual point will be same as neighbors (Figure 2.4 (a)), but MaNG made agents arrive at unique virtual point (Figure 2.4 (b)), hence situations where two agents arrive at the same position at the same time were avoided. Nevertheless, it was possible that two agents passed each other on the same edge. When the agent bumped into another agent, they successfully avoided each other by passing through right side.



(a) (b)

Figure 2.4: Comparison of the first-order Voronoi graph and MaNG

They applied this model to various multi-agent simulations, and discussed the performance of this model. As a result, this model presented the efficiency of finding a more direct and natural looking path. However, they suggested that the limitation of this model was the noisy motions.

Most research for path planning has treated 3D space as 2D space, mapping the objects onto ground surface. However, even if agent movement was prevented by obstacles on 2D coordinates, some situations could be resolved in 3D space without reroute. For example, if a person wished to pass through a low tunnel, the person would walk through with the bent back. Hsu and Li [9] introduced particular research for this humanoid resolution (Figure 2.5). To simplify the problem, they focused the avoidance for upper part of body.

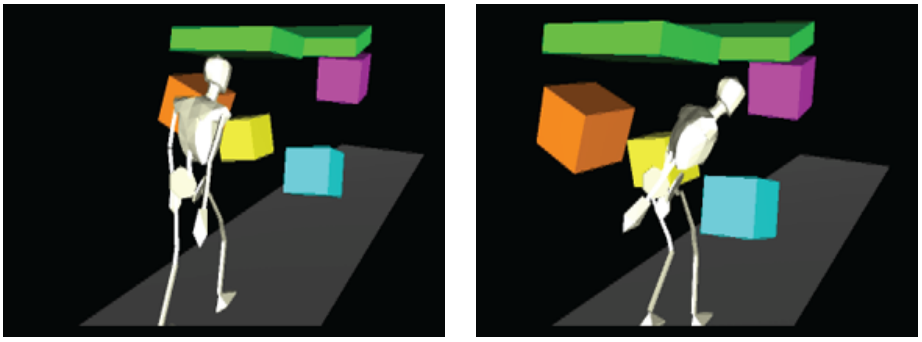


Figure 2.5: Example motion of agent avoiding obstacles

This approach was divided into two processes, which were global path planning and local motion planning. Using the pelvis position as a root of agent, the collision free path of agent was computed by PRM and DFS algorithm in global path planning process. If there were obstructions for the upper body on the collision free path, local motion process attempted to avoid the collision changing the agent pose. In order to generate local motions, many points on the path were taken as sample points, and the collision free positions for the rest of body were computed preserving as key frames. These key frames were called during the locomotion and used to generate avoidance poses.

In the implementation, it produced plausible behavior when the agent bumped into regional and avoidable obstacles in the simple scene. However, if it was complex obstacles, this algorithm could not produce successful result. Increased key frame size also made low performance speed, so this may not be suited for real time implementation.

Main difficulty of navigation is the solution for moving objects, because it needs dynamic process to solve this problem. There is a large volume of published studies describing the solution of moving obstacles [6] [8] [10]. Most of the prior work has treated it as a static object normally, and re-planed route when the objects moved. However, these are usually not suitable for real time and online application. Petti and Fraichard [22] also addressed motion planning with moving obstacles. Partial motion planning

algorithm was adopted for this investigation, but a main drawback of this study is that the algorithm treated uncertain region as unsafe even if it was safe location, so this aspect may reduce the realism of avatar locomotion.

One of the more effective ideas for moving obstacles is Velocity Obstacles algorithm [5] which has also been developed in robotics field. This method allows an agent to predict future location using moving obstacles and the agent's current location and velocity. Agent path can be modified previously with this prediction. When there are few agents in the scene, the studies confronted problems where agents collide with other agents while avoiding other moving obstacles [5] [13]. The reason of this was that every agent was controlled by same algorithm, and these agents tried to avoid each other, but collided as a result.

Berg et al. [1] solved this problem with improved velocity obstacles called reciprocal velocity obstacles (RVO) which sets a agents velocity obstacle at the average with another agent's velocity obstacle (see section 2.2.2). However, another problem named "Reciprocal Dance" occurrs when the robots attempted to avoid collision by changing orientation to same direction [4]. Although this is common phenomenon in a real scene when humans pass each other, it looked like an awkward reaction by the virtual character and robot. Snape et al. [27] developed Hybrid Reciprocal Velocity Obstacles (HRVO) which combined velocity obstacles and RVO concepts to solve Reciprocal Dance giving the condition to ensure that each robot pass through same side of each other, such as right and right, when they across (see section 2.2.3). This algorithm presented fast and smooth performance in multi agent scene, so it may be suited to avoid collision with moving obstacles and another avatar in Second Life scene. The detail of this method will be explained in next section.

2.2 Hybrid Reciprocal Velocity Obstacles

This is the method which combines velocity obstacles and Reciprocal Velocity Obstacles (RVO) concepts. In order to understand HRVO algorithm, firstly we explain the concept of velocity obstacles and RVO briefly.

2.2.1 Velocity Obstacles

Velocity obstacles concept is developed by Fiorini and Shillert [5]. According to the definition, it is assumed that A is a robot and B is a moving obstacle, which are shaped as circle of radius r_A and r_B, respectively, and the current positions are also defined as $\boldsymbol{p}_A$ and $\boldsymbol{p}_B$ (Figure 2.6). When the velocity vectors, which represent the speed and direction of robots A and B, are defined by $\boldsymbol{v}_A$ and $\boldsymbol{v}_B$, the set of all velocities $\boldsymbol{v}_A$ (dark area in Figure 2.7) which will collide with $\boldsymbol{v}_B$ in the near future is defined as velocity obstacle ($VO_{A|B}$) for A induced by B. Assuming that a circle at position $\boldsymbol{p}$ with radius r is $D(\boldsymbol{p}, r)$, $VO_{A|B}$ can be defined as

$$VO_{A|B} = \{\boldsymbol{v} \mid \exists t > 0 :: t(\boldsymbol{v} - \boldsymbol{v}_B) \in D(\boldsymbol{p}_B - \boldsymbol{p})_A, r_A + r_B)\}. \tag{2.1}$$

If a next step's velocity $\boldsymbol{v}_A$ is selected from the area of $VO_{A|B}$, the collision of robots A and B will occur in the future, but if the outside velocity is chosen, the free collision path can be provided. This formulation showed successful results in the scene where there is only single robot, but there was a limitation when there were other robots in same scene, because oscillation of robots occurred. The reason of this oscillation is that the outside of velocity obstacle for one robot can also be outside of velocity obstacle for another robot. It means that their prediction of collision free velocity is not ensured to be safe locomotion. Therefore, the solution of this problem for multi robot navigation was required.

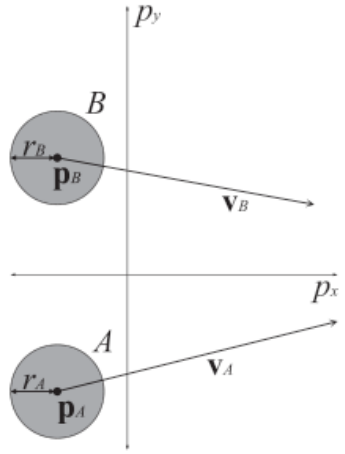


Figure 2.6: Two robots configuration

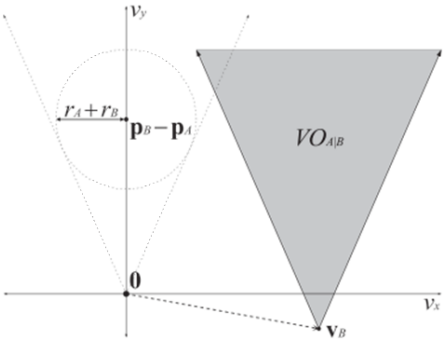


Figure 2.7: Shepe of velocity obstacle $VO_{A|B}$

2.2.2 Reciprocal Velocity Obstacles

Berg et al. [1] focused on this problem, and proposed the reciprocal velocity obstacle algorithm. In this algorithm, when a robot collides with another robot which has same free-collision system, the robot does not need to take whole responsibility to avoid collision. It only takes half responsibility and the remaining half is taken by another robot. In formulation, when robot A chooses new velocity, it takes

average of current $\boldsymbol{v}_A$ and $\boldsymbol{v}_B$ (Figure 2.8). The apex of velocity obstacle is placed between $\boldsymbol{v}_A$ and $\boldsymbol{v}_B$, and this is called reciprocal velocity obstacle$RVO_{A|B}$ for A induced by B, then

$$RVO_{A|B} = \{\boldsymbol{v} \mid 2\boldsymbol{v} - \boldsymbol{v}_A \in VO_{A|B}\}. \tag{2.2}$$

This method allows smooth crossing of two robots without oscillation when this chose to pass same side, such as the right side of one robot and right side of another, and when the robot chose the closest velocity to current velocity. However, this method also confronts problems that robots chose different side of path to cross, such as the right side of one robot and left side of another robot, because they have preferred velocity. This situation often happens in human life when two persons pass each other on the street. Berg et al. [1] mentioned that human can resolve this, but robots are unable to resolve this. Then, the behavior of robots is called "Reciprocal Dance" [4].

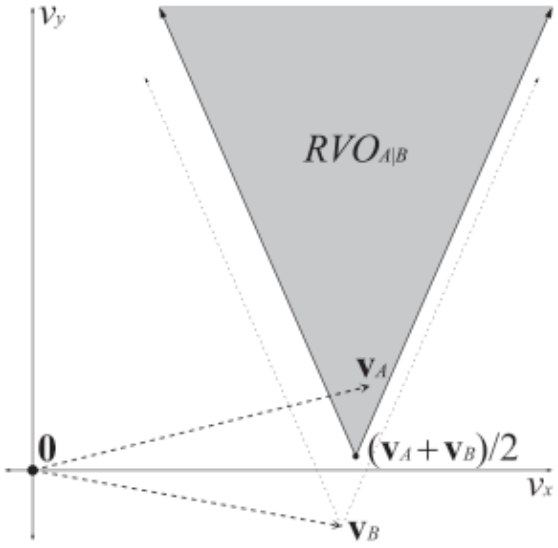


Figure 2.8: Shepe of reciprocal velocity obstacle $RVO_{A|B}$

2.2.3 Hybrid Reciprocal Velocity Obstacles

In order to improve and solve this problem, Snape et al. [27] developed Hybrid Reciprocal Velocity Obstacle (HRVO) algorithm. When $RVO_{A|B}$ is given for A, and $\boldsymbol{v}_A$ is placed at the right of centerline of $RVO_{A|B}$, they force the robot to chose right area of outside $RVO_{A|B}$ for next velocity of A. In practice, the $RVO_{A|B}$ is enlarged to the direction where they do not desire the robot to pass. In order to satisfy this condition, $VO_{A|B}$ is adopted for this enlargement. Using $RVO_{A|B}$ and $VO_{A|B}$, the apex of new velocity obstacle is settled on the intersection of the right side edge of $RVO_{A|B}$ and left side edge of $VO_{A|B}$ (Figure 2.9) in this case, and this is called $HRVO_{A|B}$ for A induced by B. If it is opposite case, such as $\boldsymbol{v}_A$ is on the left side of $RVO_{A|B}$, the apex is changed at the intersection of the left edge of $RVO_{A|B}$ and the right edge of $VO_{A|B}$. The reason why this method is called "hybrid" is that it generates new velocity area combining velocity obstacle and reciprocal velocity obstacle. In addition,

the robot B is given full priority of choice when robot A chose wrong side which influences B because of some influence from third robots. This condition reduced the oscillation problem efficiently.

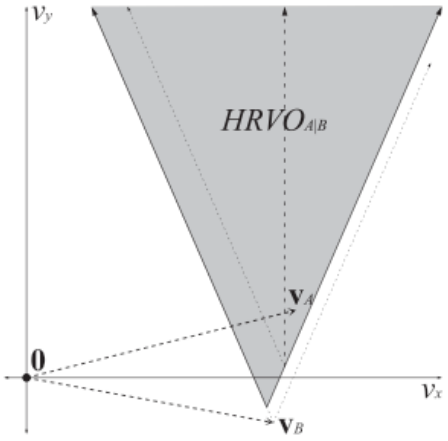


Figure 2.9: Shape of hybrid reciprocal velocity obstacle $VO_{A|B}$

Snape et al. [27] identify several advantages of the case study, such as entrusted collision avoidance with multi agent scene and high speed performance in crowded environments. There are many avatars and moving objects in Second Life scene, so this algorithm may be effective for the autopilot mode, because it can be applied to avatars and moving objects, thus a Second Life avatar can generate collision free path during autopilot mode.

Chapter 3

Navigation Model in Second Life

This section will explain integration strategy of HRVO algorithm within Second Life, starting with the overview of Second Life, because it is a enormous program that requires understanding in order to extend the source code.

3.1 Second Life Overview

Second Life is one of MMOG and holds 11 million users. A user can communicate with another user through its avatar. Since they can use microphone and voice recognition to talk, Second Life's communication is fairly realistic. User can also explore the Second Life world by walking, running or flying freely by controlling the avatar using mouse and keyboard movements. In the world, there are some buildings or areas which exist in real world, so user can play the game by just sightseeing. For example in Figure 3.1, avatar is sightseeing Trafalgar Square in London created in Second Life world. The graphic and animation effects have also improved and it feels more immersive to the user. Furthermore, users can enjoy shopping and entertainments using Linden Dollar which is exchangeable with real worlds money. Therefore, this world can be regarded as a virtual world, which is an imitation to real world.

Figure 3.1: Second Life viewer screen-shot

Various types of objects from huge buildings to tiny accessories to decorate an avatar can be created by

user. In addition, some effects can be applied to these objects, such as movement animation and physical attribute, as a function attached to the objects. The functions written by Linden Script Language (LSL)[5] in the viewer enables duplication of almost all of the effects of real world *e.g.,* vehicles, animals, falling water and smogs. Thus, it is generally agreed that it is possible to create almost all of the idea which human can imagine. However, these effects can be applied to objects mainly. If user wishes to extend avatar's behavior, what user only can do is to add avatar's pose or animation data by uploading of BVH file, because LSL cannot edit or add for avatar's function directly.

It means that avatar's fundamental operation, such as where the avatar is looking at or avatar's locomotion speed, has been edited from original source code. For these cases, Linden Lab also provides Second Life's open source code called Snowglobe which is compatible program with Second Life viewer. Snowglobe viewer can be added and fixed new feature to the official viewer, so new functions are possible to be tested by editing and adding to this code. Attention model was also integrated using this open source code. Navigation model would be new function of avatar so my research has been produced in Snowglobe. However, first difficulty of my research was to understand the structure of program and source code, because there is no documentation which explains Snowglobe and this was an enormous application.

3.2 Autopilot Mode

Prior to presenting the integration of navigation model, it is necessary to give a brief explanation of how autopilot mode works in Second Life. In the LSL, there are functions to generate objects effect, called "GoHere" or "SitHere", which force avatar to move specified position or let avatar sit particular position. For example, this is usually used to create chair object. When user wants avatar to sit on a chair, user specifies the chair and clicks "SitHere" button, and avatar starts moving independently to the chair and sit. Originally, autopilot mode is called such that types of situations. When user put "GoHere" or "SitHere", this function is called and navigates the avatar to the particular position. The flow of this process is that firstly, it decides moving direction by target coordinate data, and move avatar straight to the direction. When avatar hit another object in the scene, the physical attribute prevent the invasion of avatar into the object, but the locomotion is also stopped. Kokkinara [14] entrusted the selection of target to avatar with attention model, and attempted to generate autonomous behavior of virtual character. She called this function autopilot mode.

The basic process of autopilot mode is showed in Algorithm 3.1. Autopilot mode is activated using the keyboard input of "9" to turn on $AutopilotMode$ (i.e. set to true). Firstly, program accesses the object database of attention model which is a store of the objects data in the avatar's surrounding environment. According to the attention model's algorithm, it computes the most salient object in the scene automatically. This selection process is currently implemented as a random choice. After the selection, it sets

[5] http://wiki.secondlife.com/wiki/LSL_Portal (accessed 1 Sep 2011)

the selected object as a target of autopilot mode and moves avatar to the target position in a straight line. When the avatar reaches target, the avatar stays few seconds to observe the object. If avatar reached target object, $AutopilotMode$ is also changed to false value in the function `autoPilot`. After 15 seconds from start of autopilot mode, it chooses new target from the object database again and restarts autopilot. In situations where the avatar is left stranded due to collision, autopilot mode stops and is simply restarted after 15 seconds. The main issue appeared in Kokkinara's research was the stranded case. The locomotion of the avatar during autopilot could not distinguished whether the avatar reached the target object or it's being obstructed by another object. In the evaluation, it reported that there was no difference on the avatar's plausibility between the autopilot mode with Attention model and random model, which chose target object at random. In other words, the Kokkinara's implementation did not improve realism of avatar's locomotion.

Algorithm 3.1 Basic flow of autopilot mode

```
1:  loop
2:    /* idle loop starts */
3:    yaw = 0
4:    if AutopilotMode =true then
5:      /* update yaw */
6:      yaw =autoPilot(yaw)
7:    end if
8:    if Keyboard '9' was pressed then
9:      if global_timer → elapse > 15 sec then
10:       /* Autopilot initialization */
11:       if AutpilotMode ≠ true then
12:         AutopilotMode = true
13:         targetObj = Attention model chooses target object /* Target registration in Second Life
            simlutor. */
14:         startAutoPilotGlobal(targetObj → globalPos)
15:       end if
16:     else
17:       global_timer → elapse = 0
18:     end if
19:   end if
20: end loop
```

In order to solve this problem, navigation model attempts avatar to avoid collision automatically with HRVO algorithm. HRVO open source code is published by University of North Carolina[6]. Having generated the library from the source code, HRVO library is embedded in Second Life project.

Basic idea for integration of HRVO algorithm and autopilot mode is to synchronize Second Life viewer process and HRVO simulator process. When autopilot mode is activated, HRVO simulator is initialized at the same time, acquiring the required data from the Second Life simulator. The indispensable element for agent definition is avatar's position, orientation, speed, radius and goal number. In HRVO algorithm, it is possible to define multiple agents who have different targets to one another, so goal number is also

[6]http://gamma.cs.unc.edu/HRVO/

defined to specify which goal the agent heads to. The goals need to be converted to two dimensional coordinates to work with the HRVO simulator. This can be substituted by global position of target object which is selected by default autopilot mode. Additionally obstacles data are necessary to be defined in HRVO simulator. In HRVO library, an obstacle is configured as a line which is defined by two end points, so it means that four lines are required to set a cubic object for 2D coordinates and infinite number of lines are needed to define cylinder objects for ideal in HRVO simulator. To simplify, the bounding box is adopted, which is originally provided by Second Life program. There are also considerable amount of objects in Second Life scene, so acquisition algorithm of these objects information was considered. Main approach to obtain the obstacle data is to create object database which lists up the visible objects in the viewer and call this database when HRVO simulator is initialized. The detail will be explained in section 3.3. From each object data, the bounding box is acquired and registered in HRVO simulator as an obstacle. Therefore, these processes create same environment as Second Life viewer (aka Second Life simulator) in HRVO simulator. Collision avoidance, thus, can be succeeded by returning the new avatar position and orientation, which was computed by HRVO simulator, for the Second Life simulator.

3.3 Object Database

In order to construct same scene as Second Life viewer in HRVO simulator, avatar data, target position and objects data are necessary. However, the Second Life world is constructed by large amount of objects, so it is impossible to preserve all objects data in HRVO simulator in terms of the performance. Attention model creates own database in the library. It organizes database with using `updateNode` and `removeNode` function to get object from Second Life scene and remove object from database list, respectively. Nevertheless, this is only used for the target selection of attention model and not accessible from outside of the library. HRVO library also does not have the own database, so an object database was created in Second Life code. Same approach with attention model is employed to create the database because it holds same objects with attention model database.

Second Life viewer does not also present all objects in the world. For example, it does not show the objects which are located faraway, backward or another region. Which objects should rendered is judged in the idle function depending whether the object is active or not. Attention model additionally sets distance threshold on the condition, and updates the objects into the database, which is active and inside of distance threshold. When object become inactive, the object is removed from the object database. This database concept of attention model can be applied to HRVO object database to hold same list of attention model.

In order to construct the database, `map` function in the Standard Template Library (STL) of C++ is employed. `map` can manage elements using key and data value. It automatically sorts these elements inside of the class by usually tree structure, so user can access to the element efficiently but user does not need to understand the inside algorithm to create. Only key and data type should be defined by user

to construct the database, so object ID and the position from Second Life scene are stored.

The database can be managed by basically two functions called `insert` and `erase`. After initialization of database by key and data with object ID and position, respectively, `insert` function is called in `processUpdateCore` in `LLViewerObjectList` class where is called every time when update of particular object is processed. On the other hand, if an object disappeared from the scene or should be inactive because of any other reasons, the object is treated as "dead" or "killed" and the processes are conducted in `LLViewerObjectList::killObject` or `LLViewerObjectList::cleanDeadObjects` to remove the object from object list. Therefore, `erase` function for HRVO database is also overwritten in same functions. After production of object database, it is recalled when HRVO simulator is initialized and used to define the obstacle in the simulator. However, an issue appeared in this process. If a new object is activated into the object database after starting of navigation process, it cannot be registered into HRVO simulator during the processing of autopilot locomotion. When the initialization is processed again after the avatar reached target, update database is called and new objects are registered at this time. The situation, which new object suddenly appeared in front of avatar during the locomotion, does not happen usually, but it might be problem when it occurrs.

3.4 HRVO Process

This section explains the detail flow of navigation model. Mainly this process is conducted in the function `autopilotWithNavigation` which is `LLAgent` class and called every frame when the autopilot mode is active. Firstly, it needs to construct same environment as Second Life simulator in HRVO simulator, and this process is named initialization. HRVO process needs global coordinate data to compute the relationship of agents, goals and obstacles, so comparable data should be chosen from Second Life data. Second Life world is divided into large number of regions, since it has enormous area. Thus, there is also region coordinate system (avatar coordinate system) in addition to global coordinate system. This is not the coordinate system which is the origin of avatar position, but this is region coordinate where the avatar exists. Second Life computation often uses this coordinate for small operation instead of global coordinate, because this is smaller than global coordinate data and uses small memory; furthermore, avatar does not across the region to region by walk usually. Therefore, region system position, which is obtained by a function `getPositionAgent`, can be the appropriate data for HRVO global coordinate data.

As the first step of initialization, it needs to register the target object position which was chosen by attention model as a goal position for HRVO simulator (Algorithm 3.2 (line 5)). This target position should be the region coordinate system. Secondly, navigation model obtains current avatar's orientation and position from Second Life simulator and defines them as an agent of HRVO simulator (Algorithm 3.2 (line 7)). It also needs to define some default setting for agent in HRVO simulator, such as initial velocity and

radius. These elements are also defined at this time (Algorithm 3.2 (line 6)). Next step is the installation of obstacles data. The objects data which are currently active is obtained from HRVO object database. From each object, bounding box data is generated and defined as an obstacle. In practice, the four points of the bounding box is used to make four lines and these lines are configured as obstacles in the HRVO simulator (Algorithm 3.2 (line 14)). The volume of bounding box is slightly extended considering the gap between avatar's origin position and physical size. Checks were implemented to ensure that target object or goal position was not inside an obstacle. Hence when an object was selected as the target, navigation model does not register it as an obstacle. After setting the time step of HRVO simulator, initialization step is done.

The position and orientation which avatar should take for next step was computed in each frame, so HRVO simulator's main process is progressed along each frame process of the Second Life simulator. However, avatar's locomotion in Second Life simulator is controlled by only direction and speed. It does not specify avatar's position during the movement. This is same method as actual human, because human only decides direction and speed when it moves. Therefore, it is better to return the only agent orientation taken from HRVO simulator to control the avatar instead of the position, because the avatar moved to the specified position forcibly, it might produce a weird animation or not work. Sufficiently the control of only orientation can allow avatar to avoid the collision if it is iterated. HRVO orientation is represented by radian which lies on 2D coordinate system, so it needs to be convert direction vector to apply to the avatar. This direction vector is derived from $\boldsymbol{dir} = (\cos r, \sin r)$, where r is an agent orientation computed from HRVO simulator and $\boldsymbol{dir}$ is the direction vector for avatar (Algorithm 3.2 (line 25)). However, the possible problem is that the small gap will be generated between avatar position in Second Life and agent position in HRVO simulator after one frame process, because the avatar's moving length at one frame must be slightly different with agent's moving length of one step at HRVO simulator even if it is approximated as much as possible by default speed parameter of agent. These small gaps produced every frame make a big difference of position between the simulators, then the navigation become imprecise. Thus, it is necessary to fix the agent position data in HRVO simulator on every frame in order to produce precise navigation (Algorithm 3.2 (line 21)).

The parameter of HRVO agent which should be fixed at each frame is agent speed represented as moving distance at one step, and position. Firstly, in order to fix the agent position, the avatar's position is returned to HRVO simulator after moving process of the avatar. This adjustment allows HRVO computation to use accurate position data of avatar every frame. Next, the agent speed adjustment also needs to be considered. The moving length of avatar in Second Life simulator is not constant. This length is changed according to frame per second (FPS) which represents process speed of Second Life simulator. If FPS is high, avatar moves short distance at one frame, but if it is low, the distance of avatar movement become long in order to make user to feel constant locomotion. Navigation model solves this problem by prediction of next distance, preserving the last frame position of avatar. Using current fame position

and last frame position, it computes how long the avatar moved from last frame to current frame, and sets this length as preferred speed of HRVO agent. This computation simply uses magnitude of two points vectors where current position and last position are assumed as $\boldsymbol{p}_t$ and $\boldsymbol{p}_{t-1}$, respectively,

$$\boldsymbol{D} = \boldsymbol{p}_t - \boldsymbol{p}_{t-1}, \quad \texttt{then} \quad Speed = \sqrt{\boldsymbol{D}_x^2 + \boldsymbol{D}_y^2}. \tag{3.1}$$

To update the speed of HRVO agent every frame makes the computation more precise (Algorithm 3.2 (line 20)).

Algorithm 3.2 Autopilot with navigation model

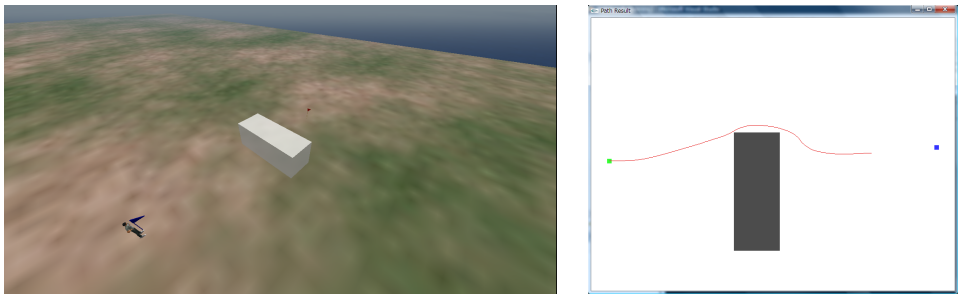
Input: Δyaw
Output: Δyaw

1: **for each** frame **do**
2: **if** Autopilot mode **then**
3: **if** HRVO Simulator is not initialized **then**
4: /* HRVO Initialization States */
5: $HRVO\_Goal \leftarrow$ add $SL\_target\_pos(p_x, p_y)$ /* SL is abbrev. of Second Life */
6: Do `setAgentDefaults()` /* Agent default setting */
7: $HRVO\_Agent \leftarrow$ add $SL\_avatar\_pos(p_x, p_y)$
8: connect $HRVO\_Goal$ and $HRVO\_Agent$
9: /* Obstacle initialization (accessing to HRVO object database) */
10: **for each** obj in $HRVO\_DB$ **do**
11: **if** not $target\_obj$ **then**
12: /* get bounding box */
13: $bbox = obj \rightarrow bounding\_box$
14: $HRVO\_Obstacles \leftarrow$ add $bbox$
15: **end if**
16: **end for**
17: set $Time\_Step$
18: **else**
19: calculate current $agent\_speed$ (see equation 3.1)
20: $AgentPrefSpeed \leftarrow$ set $agent\_speed$
21: $AgentPosition \leftarrow$set $SL\_avatar\_pos(p_x, p_y)$
22: **end if**
23: Do HRVO computation for one frame
24: $orientation \leftarrow$ get $HRVO\_orientation$
25: compute $next\_direction$ from $orientation$ /* change radian to 2D */
26: $yaw \leftarrow$ angle between current $facing\_direction$ and $next\_direction$
27: $turn =$figure out which direction to turn
28: **if** $turn =$positive **then**
29: $\Delta yaw = yaw$
30: **else**
31: $\Delta yaw = -yaw$
32: **end if**
33: **end if**
34: **end for**

3.5 Initial Tests

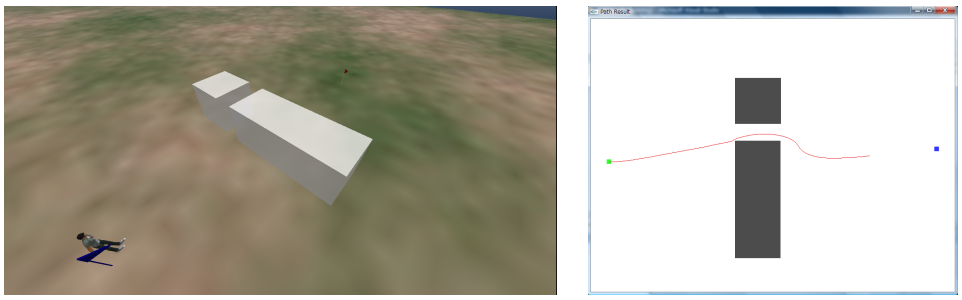
Prior to experiments, the performance of the navigation model is tested in two simple scenes. First scene has one object in front of avatar and target position is placed on the opposite side of the object. Second scene also has objects in front of the avatar and target position is located over these objects, but there is small space between two objects. The first scene examines whether the avatar can avoid obstacle (Figure 3.2 (a)), and the second scene investigates that model chooses appropriate path (Figure 3.3 (a)). The ideal path for second scene is obviously to pass through the space between the two objects.

In both simple tests, navigation model is tested to see if it could lead the avatar to the target. The scene result paths which traced the avatar locomotion are shown in Figure 3.2 (b) and 3.3 (b). The start position and goal position is represented as green and blue points in the result image respectively, and the path is expressed by red line.



(a) (b)

Figure 3.2: First simple scene and result path



(a) (b)

Figure 3.3: Second simple scene and result path

As shown in Figure 3.2 and 3.3, it can be seen that two paths reached goal position avoiding the objects collisions. This test proved that navigation model could make collision free path successfully in simple obstacles scene.

Chapter 4

Implementation and Evaluation

In order to evaluate the effectiveness of navigation model, two experiments were conducted. The first experiment assessed plausibility of path generated by navigation model comparing other paths which were captured from the avatar controlled by human. In this experiment, the participants manipulated an avatar from start position to goal position in three different complex environments while the paths were logged. Navigation model also controlled an avatar in the same situations, logging the paths. The similarity with humans' paths were compared. The realism of the locomotion paths was assessed by using Dynamic Time Warping algorithm to measure the similarity and one-way ANOVA to judge the significance of difference.

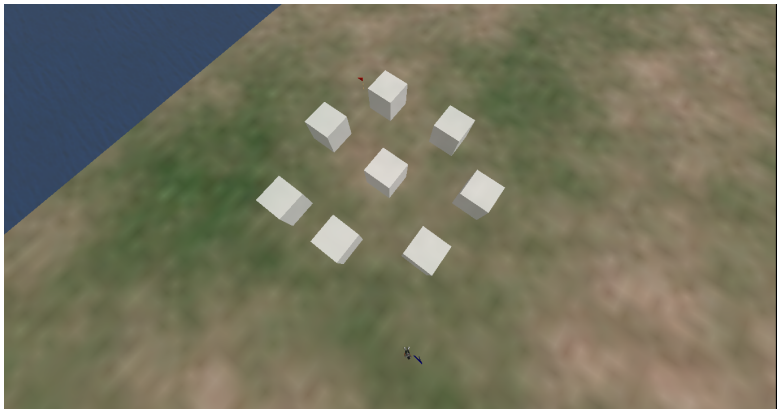
The second experiment was focused on the performance time of navigation model. It was measured using frames per second (FPS) data while varying the number of objects from 10 to 50 in the scene. In order to compare the performance with normal process, the FPS of three type modes, which was normal mode, default autopilot mode and navigated autopilot mode, were taken and also compared each other.

4.1 Experiment 1 - Navigation model path comparison

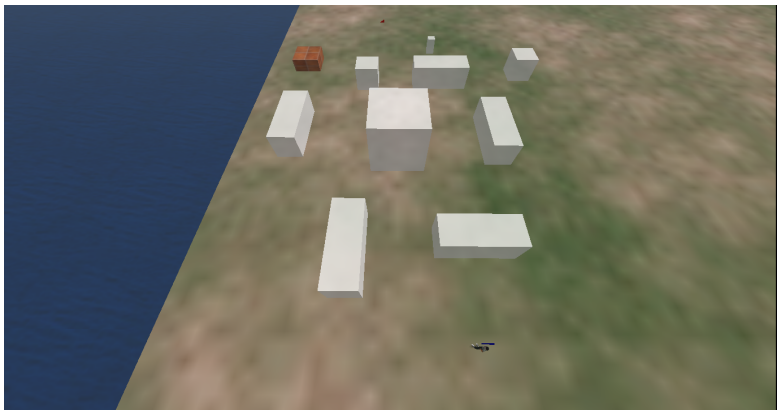
4.1.1 Experiment Design

This experiment compared avatar paths controlled from start to goal point by human or navigation model. Three different environments which had different number of objects were created for this experiment. Figure 4.1 provides the views from top of these scenes. In each scene, blue flag was stood at start point and red flag was stood at goal point. The first environment had 8 objects, and start and goal flag were pointed the bottom and top in the scene, respectively (Figure 4.1 (a)). The second environment where had start and goal position at bottom and top of Figure 4.1 (b), had 10 objects. The third environment which had more complex objects (20 objects) in the scene was also set the start and goal position at bottom and top location, respectively, in Figure 4.1 (c).

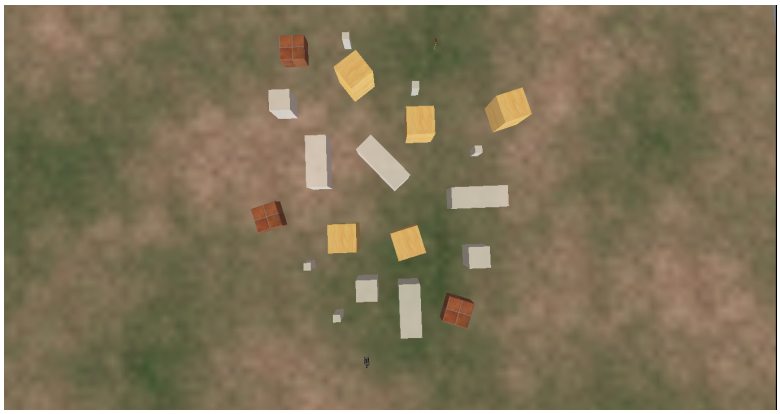
In the experiment, firstly the participant checked the goal position from a bird's eye view of the scene (Figure 4.1) in order to decide a path which they would navigate an avatar to the target most efficiently.



(a) Test 1 - 8 objects scene



(b) Test 2 - 10 objects scene



(c) Test 3 - 20 objects scene

Figure 4.1: Three different experimental scene

Prior to starting the experiment, the participants were given an opportunity to practice the control of avatar as walking around the outside of the environment, and the participant started the experiment to move the avatar from start to goal flag. The motion paths of 10 participants' were captured and logged. After examination of the participants, navigation model also controlled the avatar in the same environments aiming to reach the goal position, and these paths were captured.

4.1.2 Analysis

In order to compare the similarity of paths controlled by human and navigation model, Dynamic Time Warping (DTW) algorithm (or called Dynamic Programming (DP) algorithm) was employed. This algorithm, which was originally invented for genetic pattern matching [20], computes the similarity of two sequences, which could be different lengths. Recently, this has been widely applied to recognition processing in several fields, such as voice recognition and body motion recognition [19] [23] [25].

DTW is simple algorithm, which allows two sequences to be stretched and compressed. Main computation focuses on the generation of DTW grid which restores accumulative distance of two sequences from the start elements. Suppose that there are two time sequences P_1 and P_2, of length N and M respectively, where $P_1 = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots \boldsymbol{p}_i, \ldots, \boldsymbol{p}_N\}$, $P_2 = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_j, \ldots, \boldsymbol{p}_M\}$. In order to compute the similarity cost, DTW grid, which is $M+1 \times N+1$ matrix, is needed to be created. The element $g(i,j)$ of DTW grid is computed by recurrence formula, and the initial entries are defined by

$$
\begin{aligned}
g(0,0) &= 0, && (4.1)\\
g(1,j) &= \texttt{Not a number}, \quad \text{for } (1 \leq j \leq M) && (4.2)\\
g(i,1) &= \texttt{Not a number}, \quad \text{for } (1 \leq i \leq N). && (4.3)
\end{aligned}
$$

The inside of matrix is generated by the distances of each time step of two sequences summing with minimal cost of neighbors. In order to choose neighbors, simple step pattern was adopted, which only takes right, upper and upper right cells, when initial cell is assumed at bottom left corner. Suppose the distance $d(i,j) = |\boldsymbol{p}_i - \boldsymbol{p}_j|$, the recurrence formula is

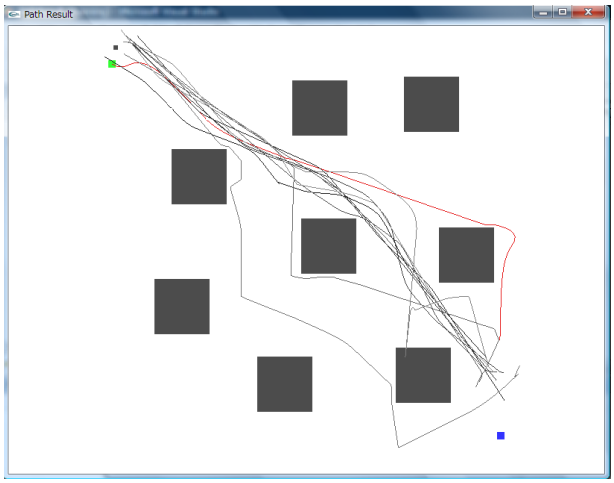
$$
g(i,j) = d(i,j) + \min \begin{cases} g(i-1,j) \\ g(i-1,j-1) \\ g(i,j-1) \end{cases}, \quad \text{for } (2 \leq i \leq N, 2 \leq j \leq M). \tag{4.4}
$$

After this computation of DTW matrix, the value of top right cell will be the cost of similarity between the two sequences.

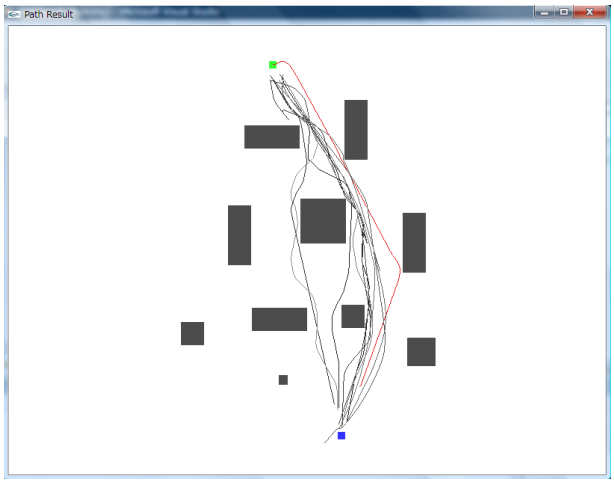
In the analysis, the similarity between navigation model's path and each human's paths were computed. Furthermore, DTW costs of human paths were also computed with one another, since standard costs between each human paths were unknown. Table 4.1 presents these costs of test 1 as an example. Lower costs illustrate that these paths were similar, so the diagonal cells showed zero values, because it compared two paths which were totally same sequences.

The plot of result paths for three environments are shown in Figure 4.2. In the Figure, red line is the path generated by navigation model and other lines are paths of human control. Green point and blue point implies start point and goal point respectively, and other grey squares represent obstacles in the scene.

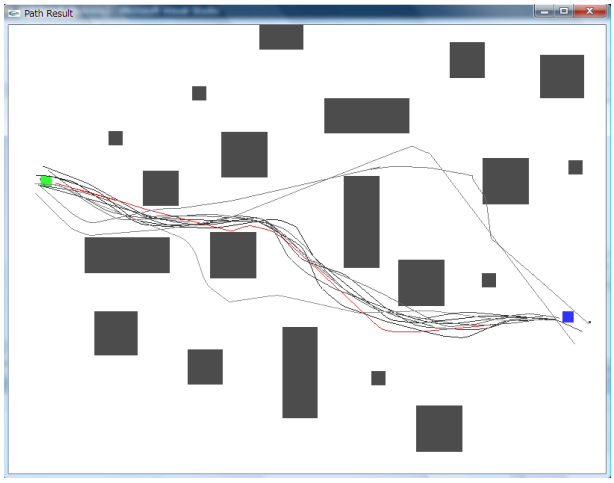
DTW costs of test 1 are presented in Table 4.1 and others are shown in Appendix C and D. In order to evaluate these data certainly, further analysis is necessary. These data were gathered from 10 participants and the navigation model. Hence the objective of the analysis is to identify whether there is significant difference between each path or not. Therefore, one-way ANalysis Of VAriance (ANOVA) was employed, because ANOVA can judge the significance about each group under the assumption of that this data sample is one independent variable data. In addition, Tukey test was chosen as the post hoc comparisons to compare all paths with one another.



(a) Path result - Test 1



(b) Path result - Test 2



(c) Path result - Test 3

Figure 4.2: Experiment 1 - Path result

	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	Navi
H1	0	212.16	177.81	193.65	233.05	228.82	662.97	1686.2	410.92	213.05	508.71
H2	212.16	0	74.324	114.8	136.07	85.502	611.31	1648.9	359	151.22	498.26
H3	177.81	74.324	0	81.896	103.71	77.229	593.32	1602.5	290.61	157.4	495.79
H4	193.65	114.8	81.896	0	145.31	139.31	629.2	1601.7	258.61	150.9	485.21
H5	233.05	136.07	103.71	145.31	0	119.63	591	1606.6	350.94	139.94	623.18
H6	228.82	85.502	77.229	139.31	119.63	0	587.74	1658.3	329.53	131.05	546.94
H7	662.97	611.31	593.32	629.2	591	587.74	0	1416.6	743.57	612.26	1012.4
H8	1686.2	1648.9	1602.5	1601.7	1606.6	1658.3	1416.6	0	1769.1	1765	2263.5
H9	410.92	359	290.61	258.61	350.94	329.53	743.57	1769.1	0	355.73	673.86
H10	213.05	151.22	157.4	150.9	139.94	131.05	612.26	1765	355.73	0	542.94
Navi	508.71	498.26	495.79	485.21	623.18	546.94	1012.4	2263.5	673.86	542.94	0

Table 4.1: Test 1 - DTW result

The results of one-way ANOVA and Tukey tests are shown in Table 4.2 to 4.7. In order to evaluate the significance, the level of significance was chosen as 0.05. Table 4.3 illustrates the relationship between navigation model path and other human paths as Tukey test result of test 1. A one-way ANOVA revealed a significant difference between the paths $F(10, 110) = 3.20$, $p < 0.05$ (Table 4.2). However, when the paths were compared each other using Tukey test, it showed that there were no significant differences between the cost for the navigation model path and each path navigated by the 10 participants tested. Therefore, the path of navigation model can be proved that these path planning were believable as a human navigation.

In addition, the results of test 4 and 5 also showed that the navigation model paths did not have significant differences compared to all of human paths (Table 4.5 and 4.7). Especially, the test 3 result showed non-significant values for almost all paths. One of issues which was appeared was that the path No. 8 in test 1 was only one path which indicated significant difference with navigation model. However, the reason was clear because the path chosen by participant no. 8, who turned right at the first obstacle and also passed through right side of last obstacle resulted in a longer more unusual path as shown in Figure 4.2 (a). These two paths chose totally different route so there should be a significant difference between them.

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	14071475.08	10	1407147.508	6.213	.000
Within Groups	24912178.16	110	226474.347		
Total	38983653.24	120			

Table 4.2: Test 1 - one-way ANOVA

path	1	2	3	4	5	6	7	8	9	10
Sig.	0.95	0.84	0.78	0.82	0.87	0.84	1	0.003	0.99	0.90

Table 4.3: Test 1 - Turkey test results of navigation model

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	15506690.21	10	1550669.021	3.202	0.001
Within Groups	53276582.56	110	484332.569		
Total	68783272.77	120			

Table 4.4: Test 2 - one-way ANOVA results

path	1	2	3	4	5	6	7	8	9	10
Sig.	0.78	1.00	1.00	0.87	0.96	1.00	1.00	0.83	0.85	0.48

Table 4.5: Test 2 - Turkey test results of navigation model

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	28449209.44	10	2844920.944	4.145	.000
Within Groups	75504585.18	110	686405.32		
Total	1.04E+08	120			

Table 4.6: Test 3 - one-way ANOVA results

path	1	2	3	4	5	6	7	8	9	10
Sig.	1	1	1	1	1	1	1	0.043	0.126	0.997

Table 4.7: Test 3 - Turkey test results of navigation model

4.2 Experiment 2 - Performance Speed

4.2.1 Experiment Design

The second experiment focused on the performance speed of navigation model. Five types of environments were created changing the number of objects from 10 to 50 for this experiment. Figure 4.3 and 4.4 presents the configuration of objects in the experimental scenes where had 40 and 50 objects as examples. In order to compare the performance with default performance, the human control mode and default autopilot mode, were also conducted in the same environments. In the human control mode, avatar walked around in each environment being controlled by user in a constant time period of three minutes. Default autopilot mode also moved avatar without collision avoidance during the same constant time. After starting the autopilot mode, it repeated the locomotion within the environment for a period of three minutes. Navigation mode performance time was also acquired using the same procedure. These models' performances were assessed by changing complexity of environment, and checked the smoothness of the program processing.

In order to evaluate the processing speed, frame per second (FPS) data was gathered. This is default information of Second Life, and is printed in the console at once in the few seconds. The FPS data was collected repeating the locomotion for three minutes. After the collection, the mean of FPS data for each simulation was calculated and compared with one another.

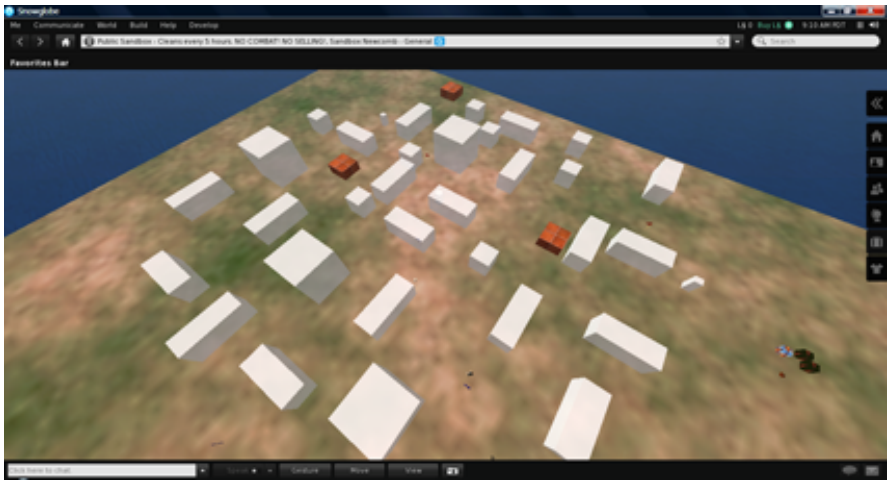


Figure 4.3: Complex environment with 40 objects

Figure 4.4: Complex environment with 50 objects

4.2.2 Analysis

The changes of each model's performance along with the number of objects computed are presented in Table 4.8. The plot of this table was also shown in Figure 4.5.

As can be seen from the Figure 4.5, all models' performances shows a slight decline as the number of objects within the virtual environment increases. In particular, navigation model's performance slightly takes lower values than others from the scene of 20 objects, though this is almost same value with default autopilot mode at the beginning of graph. The computation time of the Navigation model was expected to be lower than the normal or default autopilot mode. More interestingly, it is noted that the computational cost was only about 2 - 3 fps lower than normal or default autopilot mode.

	10	20	30	40	50
Normal	22.671	21.193	21.761	21.29	19.176
Auto Default	20.41	21.31	21.43	20.57	18.883
Auto with Navi	20.489	19.2	20.439	18.627	16.865

Table 4.8: Changes of FPS average along the number of objects

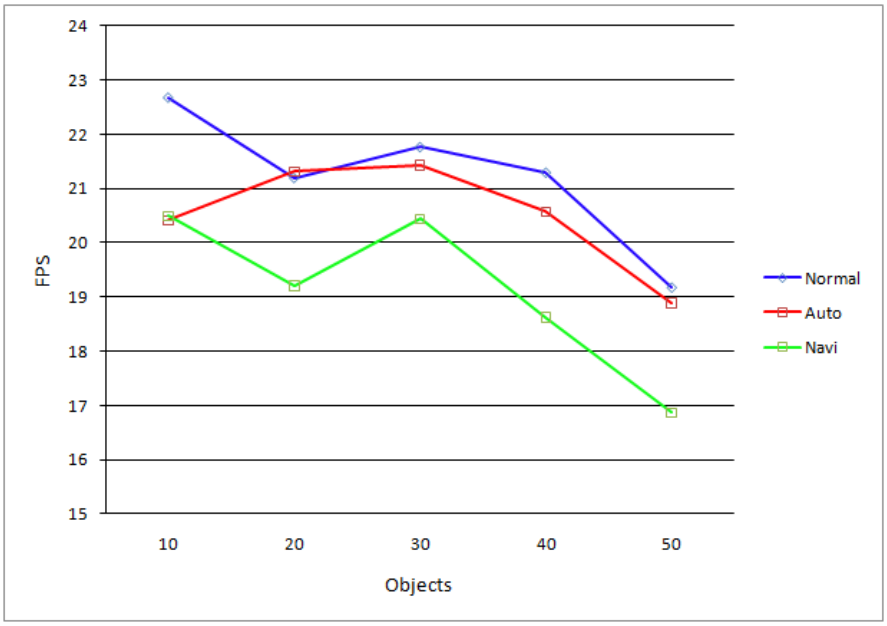


Figure 4.5: Plot of performance changes

Chapter 5

Discussions and Conclusions

5.1 Discussions

This section considers the experimental results, and identifies the benefits and drawbacks of navigation model. The main objective of this project was to integrate a navigation model to improve the autonomous behavior of virtual character. Automatic navigation is necessary to produce humanoid path planning, and smooth animation in any virtual environment. In order to evaluate these aspects, two experiments were conducted. First and second experiments focused on the humanity of path planning of navigation model and the adaptability for various environments, respectively. In this section, these obtained data were analyzed minutely for each experiment and assessed against the objectives of the research.

Firstly, the first experiment compared the path planning of actual human and navigation model. From the plot of paths (Figure 4.2), the route of navigation model for test 1 is possibly thought to be artificial locomotion, because of the visibly different path. These differences were also shown in the comparison of DTW cost. However, one-way ANOVA analysis indicated that there was no differences of significance between the human paths for test 3. There was one different path which was mentioned in section 4.1.2, but this participant highlighted that she did not get accustomed to control avatar enough during the experiment.

On the other hand, choosing a major route, navigation model produced a highly optimum path when compared to human control in test 2 and 3, although the scene was more complex environments. Even in the plot diagram (Figure 4.2), the navigation model path was visibly similar to eight of the 10 captured human paths. It can thus be suggested that navigation model is able to produce believable locomotion.

During the experiments, it was found from observing the participants that navigation model could help the control of unskilled user. Some participants who rarely played video games found it difficult to control an avatar locomotion. They needed more practice to start the experiments. This finding, while preliminary, suggested that the navigation model-controlled avatar could be navigated by only specification of goal position, the complexity of control for these users may be resolved.

The second experiment assessed the navigation model's adaptability to any environments with the measurement of processing speed in various environments. In the simplest environment with 10 objects, navigation model showed almost same performance speed with default autopilot mode. Nevertheless, in environments with greater number of objects, it had lower performance speed than normal mode and default autopilot mode. These results mentioned that there were some issues in the object process of navigation model. At the initialization phase, navigation model inserts all of objects, which were stored in HRVO object database, to HRVO simulator as the obstacles. If the number of object in the environment was increased, processing period for this registration become longer proportionately, so a remedy for this issue may provide better performance speed of navigation model. Nevertheless the computational cost was only about 2 - 3 fps on average.

5.2 Conclusions

This research focused on a collision free path planning for virtual characters. Second Life avatars had been extended by attention model and presented autonomous behavior about its attention, but the path planning for virtual character required more work, as there were issues with the locomotion. This project was undertaken to integrate a navigation model in order to resolve the issues, and evaluated the performance using two approaches.

The main aim of this thesis was to integrate path planning algorithm with Second Life. The HRVO algorithm, which can be applied to dynamic and multi-agent environment, was adapted, and the methodology of integration was proposed. Consequently, the avatar extended by navigation model produced autonomous path planning with collision avoidance.

Two experiments were conducted to evaluate the realism of the autonomous locomotion. One of the more significant findings to emerge from this study was that the paths generated by navigation model were similar to the motion paths by most of the 10 human participants. The Second Life avatar was also able to reach its target position without human control. Additionally it was noted that navigation model could be helpful for unskilled user's control. Bothersome control could be avoided if the model navigated avatar to specified position or suggested paths to naive users.

However, important limitation lies in the fact that the processing speed became lower by 2-3 fps in the complex environments. Object registration at the initialization of HRVO simulator was implied as the reason. In order to resolve this problem, it needs to decrease the number of object which are stored in HRVO simulator. A possible solution for future work is to avoid importing objects which are located at opposite direction of target, or if an object is located on an unlikely avatar path. The performance would be improved considerably in this situation.

Further research regarding the role of extension to 3D space would be great help of virtual character

animation, because the buildings in Second Life is sometimes divided into few floors, and there are also some floating objects. Current investigation was limited to a 2D solution. Hence 3D information was collapsed into 2D data. Thus, present avatar tries to avoid very large objects even though it can passed through without the need for collision avoidance. Furthermore, the study by Hsu and Li [9] is a possibly effective approach. Avatar does not need to be erect when walking. If it was possible to avoid collision by changing the pose, it might produce more believable animation.

Another possibility of improvement is to apply navigation model to moving objects and multiple agents. There are various moving objects and agents in the Second Life world. HRVO is possible to adjust to such that dynamic environments. In order to utilize the ability of HRVO algorithm maximally, it is essential to apply the model to dynamic environment to make more plausible virtual character.

Navigation model generated virtual character who moves automatically, and it was integrated to Second Life virtual world. It means that this model could be applied to various virtual scenes in the near future, and provide any agents presenting humanoid reaction according to the change of environment.

Appendix A

System Manual

This is the instruction for combining navigation model and Second Life viewer which was extended by attention model.

1. Second Life source code is needed to download and compile. There is detailed instruction in Second Life's wiki site[7]. The source code should be built in Visual Studio 2005.

2. After the success of compile, attention model library and some program code should be added in Second Life program in order to use attention model and autopilot mode. The `.dll` and `.lib` files are provided by Oyewole Oyekoya[8]. The detail instruction of attention model is presented in Kokkinara's thesis [14].

3. HRVO source code can be found in University of North Carolina Web site[6]. Files downloaded should be located into `SecondLife/indra/newview/`.

4. Merge the codes written in Appendix E.1.

[7] http://wiki.secondlife.com/wiki/Get_ source_ and _ compile (accessed 7 Sep 2011)
[8] w.oyekoya@cs.ucl.ac.uk
[6] http://gamma.cs.unc.edu/HRVO/

Appendix B

Experiment 1 - Test 1 Results

Table B.1: Test 1 - DWT result

	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	Navi
H1	0	212.16	177.81	193.65	233.05	228.82	662.97	1686.2	410.92	213.05	508.71
H2	212.16	0	74.324	114.8	136.07	85.502	611.31	1648.9	359	151.22	498.26
H3	177.81	74.324	0	81.896	103.71	77.229	593.32	1602.5	290.61	157.4	495.79
H4	193.65	114.8	81.896	0	145.31	139.31	629.2	1601.7	258.61	150.9	485.21
H5	233.05	136.07	103.71	145.31	0	119.63	591	1606.6	350.94	139.94	623.18
H6	228.82	85.502	77.229	139.31	119.63	0	587.74	1658.3	329.53	131.05	546.94
H7	662.97	611.31	593.32	629.2	591	587.74	0	1416.6	743.57	612.26	1012.4
H8	1686.2	1648.9	1602.5	1601.7	1606.6	1658.3	1416.6	0	1769.1	1765	2263.5
H9	410.92	359	290.61	258.61	350.94	329.53	743.57	1769.1	0	355.73	673.86
H10	213.05	151.22	157.4	150.9	139.94	131.05	612.26	1765	355.73	0	542.94
Navi	508.71	498.26	495.79	485.21	623.18	546.94	1012.4	2263.5	673.86	542.94	0

Table B.2: Test 1 - Notes

Notes		
Output Created		06-Sep-2011 11:32:40
Comments		
Input	Active Dataset	DataSet0
	Filter	$< none >$
	Weight	$< none >$
	Split File	$< none >$
	N of Rows in Working Data File	121
Missing Value Handling	Definition of Missing	User-defined missing values are treated as missing.
	Cases Used	Statistics for each analysis are based on cases with no missing data for any variable in the analysis.
Syntax		ONEWAY cost BY path /STATISTICS HOMOGENEITY BROWNFORSYTHE WELCH /MISSING ANALYSIS /POSTHOC=TUKEY GH ALPHA(0.05).
Resources	Processor Time	00 00:00:00.109
	Elapsed Time	00 00:00:00.170

Table B.3: Test 1 - Test of Homogeneity of Variances

Levene Statistic	df1	df2	Sig.
0.101	10	110	1

Table B.4: Test 1 - one-way ANOVA

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	14071475.08	10	1407147.508	6.213	.000
Within Groups	24912178.16	110	226474.347		
Total	38983653.24	120			

Table B.5: Test 1 - Robust Tests of Equality of Means

	Statistica	df1	df2	Sig.
Welch	4.403	10	43.962	.000
Brown-Forsythe	6.213	10	104.415	0

Table B.6: Test 1 - Turkey test results of 1 - 5 paths (path 11 is navigation model)

(I) path	(J) path	Mean Difference (I-J)	Std. Error	Sig.	95 % Confidence Interval	
					Lower Bound	Upper Bound
1	2	57.79945	202.92155	1	-609.3102	724.9091
	3	79.341	202.92155	1	-587.7686	746.4506
	4	66.06855	202.92155	1	-601.0411	733.1782
	5	43.44636	202.92155	1	-623.6632	710.556
	6	56.66264	202.92155	1	-610.447	723.7722
	7	-266.63909	202.92155	0.964	-933.7487	400.4705
	8	-1135.55091	202.92155	0	-1802.6605	-468.4413
	9	-92.23	202.92155	1	-759.3396	574.8796
	10	27.98636	202.92155	1	-639.1232	695.096
	11	-283.95	202.92155	0.946	-951.0596	383.1596
2	1	-57.79945	202.92155	1	-724.9091	609.3102
	3	21.54155	202.92155	1	-645.5681	688.6512
	4	8.26909	202.92155	1	-658.8405	675.3787
	5	-14.35309	202.92155	1	-681.4627	652.7565
	6	-1.13682	202.92155	1	-668.2464	665.9728
	7	-324.43855	202.92155	0.88	-991.5482	342.6711
	8	-1193.35036	202.92155	0	-1860.46	-526.2408
	9	-150.02945	202.92155	1	-817.1391	517.0802
	10	-29.81309	202.92155	1	-696.9227	637.2965
	11	-341.74945	202.92155	0.84	-1008.8591	325.3602
3	1	-79.341	202.92155	1	-746.4506	587.7686
	2	-21.54155	202.92155	1	-688.6512	645.5681
	4	-13.27245	202.92155	1	-680.3821	653.8372
	5	-35.89464	202.92155	1	-703.0042	631.215
	6	-22.67836	202.92155	1	-689.788	644.4312
	7	-345.98009	202.92155	0.83	-1013.0897	321.1295
	8	-1214.89191	202.92155	0	-1882.0015	-547.7823
	9	-171.571	202.92155	0.999	-838.6806	495.5386
	10	-51.35464	202.92155	1	-718.4642	615.755
	11	-363.291	202.92155	0.783	-1030.4006	303.8186
4	1	-66.06855	202.92155	1	-733.1782	601.0411
	2	-8.26909	202.92155	1	-675.3787	658.8405
	3	13.27245	202.92155	1	-653.8372	680.3821
	5	-22.62218	202.92155	1	-689.7318	644.4874
	6	-9.40591	202.92155	1	-676.5155	657.7037
	7	-332.70764	202.92155	0.862	-999.8172	334.402
	8	-1201.61945	202.92155	0	-1868.7291	-534.5098
	9	-158.29855	202.92155	0.999	-825.4082	508.8111
	10	-38.08218	202.92155	1	-705.1918	629.0274
	11	-350.01855	202.92155	0.819	-1017.1282	317.0911
5	1	-43.44636	202.92155	1	-710.556	623.6632
	2	14.35309	202.92155	1	-652.7565	681.4627
	3	35.89464	202.92155	1	-631.215	703.0042
	4	22.62218	202.92155	1	-644.4874	689.7318
	6	13.21627	202.92155	1	-653.8933	680.3259
	7	-310.08545	202.92155	0.907	-977.1951	357.0242
	8	-1178.99727	202.92155	0	-1846.1069	-511.8877
	9	-135.67636	202.92155	1	-802.786	531.4332
	10	-15.46	202.92155	1	-682.5696	651.6496
	11	-327.39636	202.92155	0.873	-994.506	339.7132

Table B.7: Test 1 - Turkey test results of 6 - 11 paths (path 11 is navigation model)

(I) path	(J) path	Mean Difference (I-J)	Std. Error	Sig.	95 % Confidence Interval	
					Lower Bound	Upper Bound
6	1	-56.66264	202.92155	1	-723.7722	610.447
	2	1.13682	202.92155	1	-665.9728	668.2464
	3	22.67836	202.92155	1	-644.4312	689.788
	4	9.40591	202.92155	1	-657.7037	676.5155
	5	-13.21627	202.92155	1	-680.3259	653.8933
	7	-323.30173	202.92155	0.882	-990.4113	343.8079
	8	-1192.21355	202.92155	0	-1859.3232	-525.1039
	9	-148.89264	202.92155	1	-816.0022	518.217
	10	-28.67627	202.92155	1	-695.7859	638.4333
	11	-340.61264	202.92155	0.843	-1007.7222	326.497
7	1	266.63909	202.92155	0.964	-400.4705	933.7487
	2	324.43855	202.92155	0.88	-342.6711	991.5482
	3	345.98009	202.92155	0.83	-321.1295	1013.0897
	4	332.70764	202.92155	0.862	-334.402	999.8172
	5	310.08545	202.92155	0.907	-357.0242	977.1951
	6	323.30173	202.92155	0.882	-343.8079	990.4113
	8	-868.91182	202.92155	0.002	-1536.0214	-201.8022
	9	174.40909	202.92155	0.999	-492.7005	841.5187
	10	294.62545	202.92155	0.932	-372.4842	961.7351
	11	-17.31091	202.92155	1	-684.4205	649.7987
8	1	1135.55091	202.92155	0	468.4413	1802.6605
	2	1193.35036	202.92155	0	526.2408	1860.46
	3	1214.89191	202.92155	0	547.7823	1882.0015
	4	1201.61945	202.92155	0	534.5098	1868.7291
	5	1178.99727	202.92155	0	511.8877	1846.1069
	6	1192.21355	202.92155	0	525.1039	1859.3232
	7	868.91182	202.92155	0.002	201.8022	1536.0214
	9	1043.32091	202.92155	0	376.2113	1710.4305
	10	1163.53727	202.92155	0	496.4277	1830.6469
	11	851.60091	202.92155	0.003	184.4913	1518.7105
9	1	92.23	202.92155	1	-574.8796	759.3396
	2	150.02945	202.92155	1	-517.0802	817.1391
	3	171.571	202.92155	0.999	-495.5386	838.6806
	4	158.29855	202.92155	0.999	-508.8111	825.4082
	5	135.67636	202.92155	1	-531.4332	802.786
	6	148.89264	202.92155	1	-518.217	816.0022
	7	-174.40909	202.92155	0.999	-841.5187	492.7005
	8	-1043.32091	202.92155	0	-1710.4305	-376.2113
	10	120.21636	202.92155	1	-546.8932	787.326
	11	-191.72	202.92155	0.997	-858.8296	475.3896
10	1	-27.98636	202.92155	1	-695.096	639.1232
	2	29.81309	202.92155	1	-637.2965	696.9227
	3	51.35464	202.92155	1	-615.755	718.4642
	4	38.08218	202.92155	1	-629.0274	705.1918
	5	15.46	202.92155	1	-651.6496	682.5696
	6	28.67627	202.92155	1	-638.4333	695.7859
	7	-294.62545	202.92155	0.932	-961.7351	372.4842
	8	-1163.53727	202.92155	0	-1830.6469	-496.4277
	9	-120.21636	202.92155	1	-787.326	546.8932
	11	-311.93636	202.92155	0.904	-979.046	355.1732
11	1	283.95	202.92155	0.946	-383.1596	951.0596
	2	341.74945	202.92155	0.84	-325.3602	1008.8591
	3	363.291	202.92155	0.783	-303.8186	1030.4006
	4	350.01855	202.92155	0.819	-317.0911	1017.1282
	5	327.39636	202.92155	0.873	-339.7132	994.506
	6	340.61264	202.92155	0.843	-326.497	1007.7222
	7	17.31091	202.92155	1	-649.7987	684.4205
	8	-851.60091	202.92155	0.003	-1518.7105	-184.4913
	9	191.72	202.92155	0.997	-475.3896	858.8296
	10	311.93636	202.92155	0.904	-355.1732	979.046

Table B.8: Test 1 - Games-Howell test results of 1 - 5 paths (path 11 is navigation model)

(I) path	(J) path	Mean Difference (I-J)	Std. Error	Sig.	95 % Confidence Interval	
					Lower Bound	Upper Bound
1	2	57.79945	198.14294	1	-657.9558	773.5547
	3	79.341	196.20814	1	-629.3887	788.0707
	4	66.06855	195.04021	1	-638.4503	770.5874
	5	43.44636	195.47418	1	-662.6344	749.5271
	6	56.66264	198.79958	1	-661.4906	774.8159
	7	-266.63909	172.38656	0.885	-895.448	362.1698
	8	-1135.55091	217.11946	0.002	-1922.8811	-348.2207
	9	-92.23	196.91521	1	-803.5201	619.0601
	10	27.98636	203.575	1	-707.7988	763.7715
	11	-283.95	220.84543	0.961	-1085.7815	517.8815
2	1	-57.79945	198.14294	1	-773.5547	657.9558
	3	21.54155	198.33029	1	-694.8841	737.9672
	4	8.26909	197.17493	1	-704.0325	720.5707
	5	-14.35309	197.60422	1	-728.1843	699.4781
	6	-1.13682	200.89436	1	-726.7969	724.5233
	7	-324.43855	174.79816	0.735	-962.8969	314.0198
	8	-1193.35036	219.0391	0.001	-1986.9877	-399.713
	9	-150.02945	199.02982	0.999	-868.9634	568.9045
	10	-29.81309	205.62113	1	-772.7693	713.1431
	11	-341.74945	222.73297	0.89	-1149.6721	466.1732
3	1	-79.341	196.20814	1	-788.0707	629.3887
	2	-21.54155	198.33029	1	-737.9672	694.8841
	4	-13.27245	195.23054	1	-718.4822	691.9373
	5	-35.89464	195.66409	1	-742.6634	670.8741
	6	-22.67836	198.98631	1	-741.4977	696.141
	7	-345.98009	172.60187	0.649	-975.6487	283.6885
	8	-1214.89191	217.29045	0.001	-2002.7805	-427.0033
	9	-171.571	197.10373	0.998	-883.5395	540.3975
	10	-51.35464	203.75736	1	-787.7757	685.0664
	11	-363.291	221.01354	0.845	-1165.6615	439.0795
4	1	-66.06855	195.04021	1	-770.5874	638.4503
	2	-8.26909	197.17493	1	-720.5707	704.0325
	3	13.27245	195.23054	1	-691.9373	718.4822
	5	-22.62218	194.49289	1	-725.1575	679.9131
	6	-9.40591	197.83478	1	-724.1277	705.3159
	7	-332.70764	171.27304	0.686	-957.0761	291.6608
	8	-1201.61945	216.23642	0.001	-1986.0768	-417.1621
	9	-158.29855	195.94113	0.999	-866.0931	549.496
	10	-38.08218	202.63294	1	-770.5926	694.4282
	11	-350.01855	219.97735	0.868	-1149.0773	449.0402
5	1	-43.44636	195.47418	1	-749.5271	662.6344
	2	14.35309	197.60422	1	-699.4781	728.1843
	3	35.89464	195.66409	1	-670.8741	742.6634
	4	22.62218	194.49289	1	-679.9131	725.1575
	6	13.21627	198.26264	1	-703.0252	729.4577
	7	-310.08545	171.76708	0.764	-936.4228	316.2518
	8	-1178.99727	216.62793	0.001	-1964.7261	-393.2684
	9	-135.67636	196.37312	1	-845.0191	573.6664
	10	-15.46	203.05069	1	-749.4204	718.5004
	11	-327.39636	220.36222	0.908	-1127.6821	472.8894

Table B.9: Test 1 - Games-Howell test results of 6 - 11 paths (path 11 is navigation model)

(I) path	(J) path	Mean Difference (I-J)	Std. Error	Sig.	95 % Confidence Interval	
					Lower Bound	Upper Bound
6	1	-56.66264	198.79958	1	-774.8159	661.4906
	2	1.13682	200.89436	1	-724.5233	726.7969
	3	22.67836	198.98631	1	-696.141	741.4977
	4	9.40591	197.83478	1	-705.3159	724.1277
	5	-13.21627	198.26264	1	-729.4577	703.0252
	7	-323.30173	175.54215	0.743	-964.7456	318.1421
	8	-1192.21355	219.63328	0.001	-1987.8197	-396.6074
	9	-148.89264	199.68354	0.999	-870.2044	572.4192
	10	-28.67627	206.25397	1	-773.866	716.5135
	11	-340.61264	223.31732	0.894	-1150.4378	469.2126
7	1	266.63909	172.38656	0.885	-362.1698	895.448
	2	324.43855	174.79816	0.735	-314.0198	962.8969
	3	345.98009	172.60187	0.649	-283.6885	975.6487
	4	332.70764	171.27304	0.686	-291.6608	957.0761
	5	310.08545	171.76708	0.764	-316.2518	936.4228
	6	323.30173	175.54215	0.743	-318.1421	964.7456
	8	-868.91182	196.04753	0.012	-1593.841	-143.9827
	9	174.40909	173.40522	0.993	-458.4704	807.2886
	10	294.62545	180.93246	0.851	-368.557	957.8079
	11	-17.31091	200.16612	1	-759.213	724.5912
8	1	1135.55091	217.11946	0.002	348.2207	1922.8811
	2	1193.35036	219.0391	0.001	399.713	1986.9877
	3	1214.89191	217.29045	0.001	427.0033	2002.7805
	4	1201.61945	216.23642	0.001	417.1621	1986.0768
	5	1178.99727	216.62793	0.001	393.2684	1964.7261
	6	1192.21355	219.63328	0.001	396.6074	1987.8197
	7	868.91182	196.04753	0.012	143.9827	1593.841
	9	1043.32091	217.92913	0.004	253.3407	1833.3012
	10	1163.53727	223.96492	0.002	353.3544	1973.7202
	11	851.60091	239.77116	0.057	-14.5635	1717.7653
9	1	92.23	196.91521	1	-619.0601	803.5201
	2	150.02945	199.02982	0.999	-568.9045	868.9634
	3	171.571	197.10373	0.998	-540.3975	883.5395
	4	158.29855	195.94113	0.999	-549.496	866.0931
	5	135.67636	196.37312	1	-573.6664	845.0191
	6	148.89264	199.68354	0.999	-572.4192	870.2044
	7	-174.40909	173.40522	0.993	-807.2886	458.4704
	8	-1043.32091	217.92913	0.004	-1833.3012	-253.3407
	10	120.21636	204.43832	1	-618.5848	859.0176
	11	-191.72	221.64149	0.998	-996.1101	612.6701
10	1	-27.98636	203.575	1	-763.7715	707.7988
	2	29.81309	205.62113	1	-713.1431	772.7693
	3	51.35464	203.75736	1	-685.0664	787.7757
	4	38.08218	202.63294	1	-694.4282	770.5926
	5	15.46	203.05069	1	-718.5004	749.4204
	6	28.67627	206.25397	1	-716.5135	773.866
	7	-294.62545	180.93246	0.851	-957.8079	368.557
	8	-1163.53727	223.96492	0.002	-1973.7202	-353.3544
	9	-120.21636	204.43832	1	-859.0176	618.5848
	11	-311.93636	227.57885	0.942	-1135.8621	511.9894
11	1	283.95	220.84543	0.961	-517.8815	1085.7815
	2	341.74945	222.73297	0.89	-466.1732	1149.6721
	3	363.291	221.01354	0.845	-439.0795	1165.6615
	4	350.01855	219.97735	0.868	-449.0402	1149.0773
	5	327.39636	220.36222	0.908	-472.8894	1127.6821
	6	340.61264	223.31732	0.894	-469.2126	1150.4378
	7	17.31091	200.16612	1	-724.5912	759.213
	8	-851.60091	239.77116	0.057	-1717.7653	14.5635
	9	191.72	221.64149	0.998	-612.6701	996.1101
	10	311.93636	227.57885	0.942	-511.9894	1135.8621

Table B.10: Test 1 - Homogeneous Subsets cost

path	N	Subset for alpha = 0.05	
		1	2
3	11	332.2354	
4	11	345.5078	
2	11	353.7769	
6	11	354.9137	
5	11	368.13	
10	11	383.59	
1	11	411.5764	
9	11	503.8064	
7	11	678.2155	
11	11	695.5264	
8	11		1547.1273
Sig.		0.783	1

Appendix C

Experiment 1 - Test 2 Results

Table C.1: Test 2 - DWT cost table

	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	Navi
H1	0	660.45	1277.1	252.96	167.6	526.41	543.41	174.69	155.63	1964.9	797.01
H2	660.45	0	986.83	613.5	805.31	1135	1132.9	820.58	604.87	1302.4	1414.4
H3	1277.1	986.83	0	1291	1919	1934.4	2211.2	1318.9	1460.1	290.21	2348.9
H4	252.96	613.5	1291	0	323.56	577.71	570.81	309.57	266.1	1970.9	951.88
H5	167.6	805.31	1919	323.56	0	625.69	658.26	231.56	238.35	2088.9	901.91
H6	526.41	1135	1934.4	577.71	625.69	0	300.64	350.38	561.39	2575.8	793.98
H7	543.41	1132.9	2211.2	570.81	658.26	300.64	0	413.99	574.01	2578.2	670.13
H8	174.69	820.58	1318.9	309.57	231.56	350.38	413.99	0	239.72	2193.7	756.84
H9	155.63	604.87	1460.1	266.1	238.35	561.39	574.01	239.72	0	1915.6	915.11
H10	1964.9	1302.4	290.21	1970.9	2088.9	2575.8	2578.2	2193.7	1915.6	0	2848.1
Navi	797.01	1414.4	2348.9	951.88	901.91	793.98	670.13	756.84	915.11	2848.1	0

Table C.2: Test 2 - one-way ANOVA Notes

Notes		
Output Created		05-Sep-2011 12:08:59
Comments		
Input	Data	test4.sav
	Active Dataset	DataSet0
	Filter	$< none >$
	Weight	$< none >$
	Split File	$< none >$
	N of Rows in Working Data File	121
Missing Value Handling	Definition of Missing	User-defined missing values are treated as missing. Statistics for each analysis are based on cases
	Cases Used	with no missing data for any variable in the analysis.
Syntax		ONEWAY cost BY path /STATISTICS HOMOGENEITY BROWNFORSYTHE WELCH /MISSING ANALYSIS /POSTHOC=TUKEY GH ALPHA(0.05).
Resources	Processor Time	00 00:00:00.125
	Elapsed Time	00 00:00:00.140

Table C.3: Test 2 - Test of Homogeneity of Variances

Levene Statistic	df1	df2	Sig.
0.665	10	110	0.755

Table C.4: Test 2 - one-way ANOVA results

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	15506690.21	10	1550669.021	3.202	0.001
Within Groups	53276582.56	110	484332.569		
Total	68783272.77	120			

Table C.5: Test 2 - Robust Tests of Equality of Means

	Statistica	df1	df2	Sig.
Welch	2.215	10	43.872	0.035
Brown-Forsythe	3.202	10	96.228	0.001

Table C.6: Test 2 - Turkey test results of 1 - 5 paths (path 11 is navigation model)

(I) path	(J) path	Mean Difference (I-J)	Std. Error	Sig.	95 % Confidence Interval	
					Lower Bound	Upper Bound
1	2	-268.73	296.75	1.00	-1244.31	706.84
	3	-774.32	296.75	0.26	-1749.89	201.26
	4	-55.26	296.75	1.00	-1030.83	920.32
	5	-130.91	296.75	1.00	-1106.48	844.67
	6	-260.11	296.75	1.00	-1235.69	715.46
	7	-284.85	296.75	1.00	-1260.43	690.72
	8	-26.34	296.75	1.00	-1001.92	949.23
	9	-37.34	296.75	1.00	-1012.91	938.23
	10	-1200.77727	296.75	0.00	-2176.35	-225.20
	11	-534.37	296.75	0.78	-1509.95	441.20
2	1	268.73	296.75	1.00	-706.84	1244.31
	3	-505.58	296.75	0.83	-1481.15	469.99
	4	213.48	296.75	1.00	-762.10	1189.05
	5	137.83	296.75	1.00	-837.75	1113.40
	6	8.62	296.75	1.00	-966.95	984.19
	7	-16.12	296.75	1.00	-991.69	959.45
	8	242.39	296.75	1.00	-733.18	1217.96
	9	231.40	296.75	1.00	-744.18	1206.97
	10	-932.04	296.75	0.08	-1907.62	43.53
	11	-265.64	296.75	1.00	-1241.21	709.93
3	1	774.32	296.75	0.26	-201.26	1749.89
	2	505.58	296.75	0.83	-469.99	1481.15
	4	719.06	296.75	0.36	-256.51	1694.63
	5	643.41	296.75	0.53	-332.16	1618.98
	6	514.20	296.75	0.82	-461.37	1489.78
	7	489.46	296.75	0.86	-486.11	1465.04
	8	747.97	296.75	0.30	-227.60	1723.55
	9	736.98	296.75	0.33	-238.59	1712.55
	10	-426.46	296.75	0.94	-1402.03	549.11
	11	239.94	296.75	1.00	-735.63	1215.52
4	1	55.26	296.75	1.00	-920.32	1030.83
	2	-213.48	296.75	1.00	-1189.05	762.10
	3	-719.06	296.75	0.36	-1694.63	256.51
	5	-75.65	296.75	1.00	-1051.22	899.92
	6	-204.86	296.75	1.00	-1180.43	770.72
	7	-229.60	296.75	1.00	-1205.17	745.98
	8	28.91	296.75	1.00	-946.66	1004.49
	9	17.92	296.75	1.00	-957.65	993.49
	10	-1145.52000	296.75	0.01	-2121.09	-169.95
	11	-479.12	296.75	0.87	-1454.69	496.46
5	1	130.91	296.75	1.00	-844.67	1106.48
	2	-137.83	296.75	1.00	-1113.40	837.75
	3	-643.41	296.75	0.53	-1618.98	332.16
	4	75.65	296.75	1.00	-899.92	1051.22
	6	-129.21	296.75	1.00	-1104.78	846.37
	7	-153.95	296.75	1.00	-1129.52	821.63
	8	104.56	296.75	1.00	-871.01	1080.14
	9	93.57	296.75	1.00	-882.00	1069.14
	10	-1069.87000	296.75	0.02	-2045.44	-94.30
	11	-403.47	296.75	0.96	-1379.04	572.11

Table C.7: Test 2 - Turkey test results of 6 - 11 paths (path 11 is navigation model)

(I) path	(J) path	Mean Difference (I-J)	Std. Error	Sig.	95 % Confidence Interval	
					Lower Bound	Upper Bound
6	1	260.11	296.75	1.00	-715.46	1235.69
	2	-8.62	296.75	1.00	-984.19	966.95
	3	-514.20	296.75	0.82	-1489.78	461.37
	4	204.86	296.75	1.00	-770.72	1180.43
	5	129.21	296.75	1.00	-846.37	1104.78
	7	-24.74	296.75	1.00	-1000.31	950.83
	8	233.77	296.75	1.00	-741.80	1209.34
	9	222.77	296.75	1.00	-752.80	1198.35
	10	-940.66	296.75	0.07	-1916.24	34.91
	11	-274.26	296.75	1.00	-1249.83	701.31
7	1	284.85	296.75	1.00	-690.72	1260.43
	2	16.12	296.75	1.00	-959.45	991.69
	3	-489.46	296.75	0.86	-1465.04	486.11
	4	229.60	296.75	1.00	-745.98	1205.17
	5	153.95	296.75	1.00	-821.63	1129.52
	6	24.74	296.75	1.00	-950.83	1000.31
	8	258.51	296.75	1.00	-717.06	1234.08
	9	247.52	296.75	1.00	-728.06	1223.09
	10	-915.92	296.75	0.09	-1891.50	59.65
	11	-249.52	296.75	1.00	-1225.09	726.05
8	1	26.34	296.75	1.00	-949.23	1001.92
	2	-242.39	296.75	1.00	-1217.96	733.18
	3	-747.97	296.75	0.30	-1723.55	227.60
	4	-28.91	296.75	1.00	-1004.49	946.66
	5	-104.56	296.75	1.00	-1080.14	871.01
	6	-233.77	296.75	1.00	-1209.34	741.80
	7	-258.51	296.75	1.00	-1234.08	717.06
	9	-11.00	296.75	1.00	-986.57	964.58
	10	-1174.43455	296.75	0.01	-2150.01	-198.86
	11	-508.03	296.75	0.83	-1483.60	467.54
9	1	37.34	296.75	1.00	-938.23	1012.91
	2	-231.40	296.75	1.00	-1206.97	744.18
	3	-736.98	296.75	0.33	-1712.55	238.59
	4	-17.92	296.75	1.00	-993.49	957.65
	5	-93.57	296.75	1.00	-1069.14	882.00
	6	-222.77	296.75	1.00	-1198.35	752.80
	7	-247.52	296.75	1.00	-1223.09	728.06
	8	11.00	296.75	1.00	-964.58	986.57
	10	-1163.43909	296.75	0.01	-2139.01	-187.87
	11	-497.03	296.75	0.85	-1472.61	478.54
10	1	1200.77727	296.75	0.00	225.20	2176.35
	2	932.04	296.75	0.08	-43.53	1907.62
	3	426.46	296.75	0.94	-549.11	1402.03
	4	1145.52000	296.75	0.01	169.95	2121.09
	5	1069.87000	296.75	0.02	94.30	2045.44
	6	940.66	296.75	0.07	-34.91	1916.24
	7	915.92	296.75	0.09	-59.65	1891.50
	8	1174.43455	296.75	0.01	198.86	2150.01
	9	1163.43909	296.75	0.01	187.87	2139.01
	11	666.40	296.75	0.48	-309.17	1641.98
11	1	534.37	296.75	0.78	-441.20	1509.95
	2	265.64	296.75	1.00	-709.93	1241.21
	3	-239.94	296.75	1.00	-1215.52	735.63
	4	479.12	296.75	0.87	-496.46	1454.69
	5	403.47	296.75	0.96	-572.11	1379.04
	6	274.26	296.75	1.00	-701.31	1249.83
	7	249.52	296.75	1.00	-726.05	1225.09
	8	508.03	296.75	0.83	-467.54	1483.60
	9	497.03	296.75	0.85	-478.54	1472.61
	10	-666.40	296.75	0.48	-1641.98	309.17

Table C.8: Test 2 - Games-Howell test results of 1 - 5 paths (path 11 is navigation model)

(I) path	(J) path	Mean Difference (I-J)	Std. Error	Sig.	95 % Confidence Interval	
					Lower Bound	Upper Bound
1	2	-268.73	213.13	0.96	-1050.57	513.10
	3	-774.32	284.74	0.26	-1809.27	260.64
	4	-55.26	245.45	1.00	-941.97	831.45
	5	-130.91	273.59	1.00	-1122.39	860.57
	6	-260.11	289.87	1.00	-1315.29	795.07
	7	-284.85	299.40	1.00	-1378.00	808.29
	8	-26.34	261.66	1.00	-972.42	919.73
	9	-37.34	250.62	1.00	-942.63	867.96
	10	-1200.78	327.50	0.05	-2407.87	6.31
	11	-534.37	300.26	0.78	-1630.98	562.23
2	1	268.73	213.13	0.96	-513.10	1050.57
	3	-505.58	253.70	0.66	-1454.57	443.40
	4	213.48	208.65	0.99	-550.21	977.16
	5	137.83	241.11	1.00	-758.88	1034.54
	6	8.62	259.44	1.00	-964.26	981.51
	7	-16.12	270.05	1.00	-1033.25	1001.01
	8	242.39	227.49	0.99	-598.11	1082.89
	9	231.40	214.70	0.99	-556.82	1019.61
	10	-932.04	300.91	0.16	-2078.17	214.08
	11	-265.64	271.00	0.99	-1286.76	755.48
3	1	774.32	284.74	0.26	-260.64	1809.27
	2	505.58	253.70	0.66	-443.40	1454.57
	4	719.06	281.40	0.34	-305.46	1743.57
	5	643.41	306.25	0.59	-463.36	1750.17
	6	514.20	320.88	0.86	-644.96	1673.37
	7	489.46	329.51	0.91	-701.63	1680.56
	8	747.97	295.64	0.35	-322.47	1818.42
	9	736.98	285.92	0.32	-301.71	1775.66
	10	-426.46	355.24	0.98	-1715.96	863.04
	11	239.94	330.30	1.00	-954.08	1433.97
4	1	55.26	245.45	1.00	-831.45	941.97
	2	-213.48	208.65	0.99	-977.16	550.21
	3	-719.06	281.40	0.34	-1743.57	305.46
	5	-75.65	270.10	1.00	-1055.77	904.47
	6	-204.86	286.59	1.00	-1249.99	840.28
	7	-229.60	296.22	1.00	-1313.38	854.18
	8	28.91	258.02	1.00	-904.71	962.54
	9	17.92	246.82	1.00	-873.81	909.64
	10	-1145.52	324.60	0.07	-2344.93	53.89
	11	-479.12	297.10	0.86	-1566.41	608.18
5	1	130.91	273.59	1.00	-860.57	1122.39
	2	-137.83	241.11	1.00	-1034.54	758.88
	3	-643.41	306.25	0.59	-1750.17	463.36
	4	75.65	270.10	1.00	-904.47	1055.77
	6	-129.21	311.02	1.00	-1253.80	995.39
	7	-153.95	319.92	1.00	-1312.30	1004.41
	8	104.56	284.91	1.00	-925.31	1134.44
	9	93.57	274.81	1.00	-901.97	1089.11
	10	-1069.87	346.36	0.14	-2331.44	191.70
	11	-403.47	320.73	0.97	-1564.91	757.98

Table C.9: Test 2 - Games-Howell test results of 6 - 11 paths (path 11 is navigation model)

(I) path	(J) path	Mean Difference (I-J)	Std. Error	Sig.	95 % Confidence Interval	
					Lower Bound	Upper Bound
6	1	260.11	289.87	1.00	-795.07	1315.29
	2	-8.62	259.44	1.00	-981.51	964.26
	3	-514.20	320.88	0.86	-1673.37	644.96
	4	204.86	286.59	1.00	-840.28	1249.99
	5	129.21	311.02	1.00	-995.39	1253.80
	7	-24.74	333.95	1.00	-1231.38	1181.89
	8	233.77	300.58	1.00	-855.66	1323.20
	9	222.77	291.03	1.00	-836.00	1281.55
	10	-940.66	359.37	0.30	-2243.54	362.21
	11	-274.26	334.73	1.00	-1483.75	935.23
7	1	284.85	299.40	1.00	-808.29	1378.00
	2	16.12	270.05	1.00	-1001.01	1033.25
	3	-489.46	329.51	0.91	-1680.56	701.63
	4	229.60	296.22	1.00	-854.18	1313.38
	5	153.95	319.92	1.00	-1004.41	1312.30
	6	24.74	333.95	1.00	-1181.89	1231.38
	8	258.51	309.79	1.00	-866.71	1383.73
	9	247.52	300.52	1.00	-848.99	1344.02
	10	-915.92	367.10	0.36	-2244.52	412.67
	11	-249.52	343.02	1.00	-1488.54	989.50
8	1	26.34	261.66	1.00	-919.73	972.42
	2	-242.39	227.49	0.99	-1082.89	598.11
	3	-747.97	295.64	0.35	-1818.42	322.47
	4	-28.91	258.02	1.00	-962.54	904.71
	5	-104.56	284.91	1.00	-1134.44	925.31
	6	-233.77	300.58	1.00	-1323.20	855.66
	7	-258.51	309.79	1.00	-1383.73	866.71
	9	-11.00	262.94	1.00	-961.51	939.51
	10	-1174.43	337.03	0.07	-2408.11	59.24
	11	-508.03	310.62	0.85	-1636.52	620.46
9	1	37.34	250.62	1.00	-867.96	942.63
	2	-231.40	214.70	0.99	-1019.61	556.82
	3	-736.98	285.92	0.32	-1775.66	301.71
	4	-17.92	246.82	1.00	-909.64	873.81
	5	-93.57	274.81	1.00	-1089.11	901.97
	6	-222.77	291.03	1.00	-1281.55	836.00
	7	-247.52	300.52	1.00	-1344.02	848.99
	8	11.00	262.94	1.00	-939.51	961.51
	10	-1163.44	328.53	0.07	-2373.29	46.41
	11	-497.03	301.38	0.84	-1596.98	602.91
10	1	1200.78	327.50	0.05	-6.31	2407.87
	2	932.04	300.91	0.16	-214.08	2078.17
	3	426.46	355.24	0.98	-863.04	1715.96
	4	1145.52	324.60	0.07	-53.89	2344.93
	5	1069.87	346.36	0.14	-191.70	2331.44
	6	940.66	359.37	0.30	-362.21	2243.54
	7	915.92	367.10	0.36	-412.67	2244.52
	8	1174.43	337.03	0.07	-59.24	2408.11
	9	1163.44	328.53	0.07	-46.41	2373.29
	11	666.40	367.80	0.76	-664.57	1997.38
11	1	534.37	300.26	0.78	-562.23	1630.98
	2	265.64	271.00	0.99	-755.48	1286.76
	3	-239.94	330.30	1.00	-1433.97	954.08
	4	479.12	297.10	0.86	-608.18	1566.41
	5	403.47	320.73	0.97	-757.98	1564.91
	6	274.26	334.73	1.00	-935.23	1483.75
	7	249.52	343.02	1.00	-989.50	1488.54
	8	508.03	310.62	0.85	-620.46	1636.52
	9	497.03	301.38	0.84	-602.91	1596.98
	10	-666.40	367.80	0.76	-1997.38	664.57

Table C.10: Test 2 - Homogeneous Subsets cost

path	N	Subset for alpha = 0.05	
		1	2
1	11	592.7418	
8	11	619.0845	
9	11	630.08	
4	11	647.9991	
5	11	723.6491	
6	11	852.8545	852.8545
2	11	861.4764	861.4764
7	11	877.5955	877.5955
11	11	1127.1145	1127.1145
3	11	1367.0582	1367.0582
10	11		1793.5191
Sig.		0.257	0.069

Appendix D

Experiment 1 - Test 3 Results

Table D.1: Test 3 - DWT cost table

	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	Navi
H1	0	271.46	247.7	255.74	274.58	262.08	285.78	2251.4	2031.2	843.06	441.65
H2	271.46	0	232.65	301.32	444.03	290.84	373.72	2460.5	2249.4	816.27	280.88
H3	247.7	232.65	0	262.39	364.68	245.73	234.83	2329.7	2144.7	952.08	378.07
H4	255.74	301.32	262.39	0	217.09	283.22	252.98	2323.9	2162.7	763.97	465.54
H5	274.58	444.03	364.68	217.09	0	363.25	306.43	2325.5	2197.3	926.02	711.67
H6	262.08	290.84	245.73	283.22	363.25	0	256.59	2318.9	2159.5	905.27	455.15
H7	285.78	373.72	234.83	252.98	306.43	256.59	0	2154.7	2003.9	924.1	507.79
H8	2251.4	2460.5	2329.7	2323.9	2325.5	2318.9	2154.7	0	468.29	3044.8	2592.2
H9	2031.2	2249.4	2144.7	2162.7	2197.3	2159.5	2003.9	468.29	0	2833	2428
H10	843.06	816.27	952.08	763.97	926.02	905.27	924.1	3044.8	2833	0	1018.2
Navi	441.65	280.88	378.07	465.54	711.67	455.15	507.79	2592.2	2428	1018.2	0

Table D.2: Test 3 - Notes

Output Created		06-Sep-2011 11:35:27
Comments		
Input	Active Dataset	DataSet0
	Filter	$< none >$
	Weight	$< none >$
	Split File	$< none >$
	N of Rows in Working Data File	121
Missing Value Handling	Definition of Missing	User-defined missing values are treated as missing. Statistics for each analysis are based on cases
	Cases Used	with no missing data for any variable in the analysis.
Syntax		ONEWAY cost BY path /STATISTICS HOMOGENEITY BROWNFORSYTHE WELCH /MISSING ANALYSIS /POSTHOC=TUKEY GH ALPHA(0.05).
Resources	Processor Time	00 00:00:00.109
	Elapsed Time	00 00:00:00.101

Table D.3: Test 3 - Test of Homogeneity of Variances

Levene Statistic	df1	df2	Sig.
0.031	10	110	1

Table D.4: Test 3 - one-way ANOVA results

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	28449209.44	10	2844920.944	4.145	.000
Within Groups	75504585.18	110	686405.32		
Total	1.04E+08	120			

Table D.5: Test 3 - Robust Tests of Equality of Means

	Statistica	df1	df2	Sig.
Welch	3.334	10	43.993	0.003
Brown-Forsythe	4.145	10	108.084	.000

Table D.6: Test 3 - Turkey test results of 1 - 5 paths (path 11 is navigation model)

(I) path	(J) path	Mean Difference (I-J)	Std. Error	Sig.	95 % Confidence Interval	
					Lower Bound	Upper Bound
1	2	-50.58364	353.2718	1	-1211.9734	1110.8062
	3	-20.71636	353.2718	1	-1182.1062	1140.6734
	4	-11.29091	353.2718	1	-1172.6807	1150.0989
	5	-87.80909	353.2718	1	-1249.1989	1073.5807
	6	-34.17091	353.2718	1	-1195.5607	1127.2189
	7	-12.37909	353.2718	1	-1173.7689	1149.0107
	8	-1373.20364	353.2718	0.008	-2534.5934	-211.8138
	9	-1228.48545	353.2718	0.029	-2389.8753	-67.0957
	10	-532.92	353.2718	0.914	-1694.3098	628.4698
	11	-192.22727	353.2718	1	-1353.6171	969.1625
2	1	50.58364	353.2718	1	-1110.8062	1211.9734
	3	29.86727	353.2718	1	-1131.5225	1191.2571
	4	39.29273	353.2718	1	-1122.0971	1200.6825
	5	-37.22545	353.2718	1	-1198.6153	1124.1643
	6	16.41273	353.2718	1	-1144.9771	1177.8025
	7	38.20455	353.2718	1	-1123.1853	1199.5943
	8	-1322.62000	353.2718	0.012	-2484.0098	-161.2302
	9	-1177.90182	353.2718	0.044	-2339.2916	-16.512
	10	-482.33636	353.2718	0.954	-1643.7262	679.0534
	11	-141.64364	353.2718	1	-1303.0334	1019.7462
3	1	20.71636	353.2718	1	-1140.6734	1182.1062
	2	-29.86727	353.2718	1	-1191.2571	1131.5225
	4	9.42545	353.2718	1	-1151.9643	1170.8153
	5	-67.09273	353.2718	1	-1228.4825	1094.2971
	6	-13.45455	353.2718	1	-1174.8443	1147.9353
	7	8.33727	353.2718	1	-1153.0525	1169.7271
	8	-1352.48727	353.2718	0.009	-2513.8771	-191.0975
	9	-1207.76909	353.2718	0.034	-2369.1589	-46.3793
	10	-512.20364	353.2718	0.933	-1673.5934	649.1862
	11	-171.51091	353.2718	1	-1332.9007	989.8789
4	1	11.29091	353.2718	1	-1150.0989	1172.6807
	2	-39.29273	353.2718	1	-1200.6825	1122.0971
	3	-9.42545	353.2718	1	-1170.8153	1151.9643
	5	-76.51818	353.2718	1	-1237.908	1084.8716
	6	-22.88	353.2718	1	-1184.2698	1138.5098
	7	-1.08818	353.2718	1	-1162.478	1160.3016
	8	-1361.91273	353.2718	0.009	-2523.3025	-200.5229
	9	-1217.19455	353.2718	0.032	-2378.5843	-55.8047
	10	-521.62909	353.2718	0.924	-1683.0189	639.7607
	11	-180.93636	353.2718	1	-1342.3262	980.4534
5	1	87.80909	353.2718	1	-1073.5807	1249.1989
	2	37.22545	353.2718	1	-1124.1643	1198.6153
	3	67.09273	353.2718	1	-1094.2971	1228.4825
	4	76.51818	353.2718	1	-1084.8716	1237.908
	6	53.63818	353.2718	1	-1107.7516	1215.028
	7	75.43	353.2718	1	-1085.9598	1236.8198
	8	-1285.39455	353.2718	0.017	-2446.7843	-124.0047
	9	-1140.67636	353.2718	0.059	-2302.0662	20.7134
	10	-445.11091	353.2718	0.973	-1606.5007	716.2789
	11	-104.41818	353.2718	1	-1265.808	1056.9716

Table D.7: Test 3 - Turkey test results of 6 - 11 paths (path 11 is navigation model)

(I) path	(J) path	Mean Difference (I-J)	Std. Error	Sig.	95 % Confidence Interval	
					Lower Bound	Upper Bound
6	1	34.17091	353.2718	1	-1127.2189	1195.5607
	2	-16.41273	353.2718	1	-1177.8025	1144.9771
	3	13.45455	353.2718	1	-1147.9353	1174.8443
	4	22.88	353.2718	1	-1138.5098	1184.2698
	5	-53.63818	353.2718	1	-1215.028	1107.7516
	7	21.79182	353.2718	1	-1139.598	1183.1816
	8	-1339.03273	353.2718	0.011	-2500.4225	-177.6429
	9	-1194.31455	353.2718	0.038	-2355.7043	-32.9247
	10	-498.74909	353.2718	0.943	-1660.1389	662.6407
	11	-158.05636	353.2718	1	-1319.4462	1003.3334
7	1	12.37909	353.2718	1	-1149.0107	1173.7689
	2	-38.20455	353.2718	1	-1199.5943	1123.1853
	3	-8.33727	353.2718	1	-1169.7271	1153.0525
	4	1.08818	353.2718	1	-1160.3016	1162.478
	5	-75.43	353.2718	1	-1236.8198	1085.9598
	6	-21.79182	353.2718	1	-1183.1816	1139.598
	8	-1360.82455	353.2718	0.009	-2522.2143	-199.4347
	9	-1216.10636	353.2718	0.032	-2377.4962	-54.7166
	10	-520.54091	353.2718	0.925	-1681.9307	640.8489
	11	-179.84818	353.2718	1	-1341.238	981.5416
8	1	1373.20364	353.2718	0.008	211.8138	2534.5934
	2	1322.62000	353.2718	0.012	161.2302	2484.0098
	3	1352.48727	353.2718	0.009	191.0975	2513.8771
	4	1361.91273	353.2718	0.009	200.5229	2523.3025
	5	1285.39455	353.2718	0.017	124.0047	2446.7843
	6	1339.03273	353.2718	0.011	177.6429	2500.4225
	7	1360.82455	353.2718	0.009	199.4347	2522.2143
	9	144.71818	353.2718	1	-1016.6716	1306.108
	10	840.28364	353.2718	0.39	-321.1062	2001.6734
	11	1180.97636	353.2718	0.043	19.5866	2342.3662
9	1	1228.48545	353.2718	0.029	67.0957	2389.8753
	2	1177.90182	353.2718	0.044	16.512	2339.2916
	3	1207.76909	353.2718	0.034	46.3793	2369.1589
	4	1217.19455	353.2718	0.032	55.8047	2378.5843
	5	1140.67636	353.2718	0.059	-20.7134	2302.0662
	6	1194.31455	353.2718	0.038	32.9247	2355.7043
	7	1216.10636	353.2718	0.032	54.7166	2377.4962
	8	-144.71818	353.2718	1	-1306.108	1016.6716
	10	695.56545	353.2718	0.67	-465.8243	1856.9553
	11	1036.25818	353.2718	0.126	-125.1316	2197.648
10	1	532.92	353.2718	0.914	-628.4698	1694.3098
	2	482.33636	353.2718	0.954	-679.0534	1643.7262
	3	512.20364	353.2718	0.933	-649.1862	1673.5934
	4	521.62909	353.2718	0.924	-639.7607	1683.0189
	5	445.11091	353.2718	0.973	-716.2789	1606.5007
	6	498.74909	353.2718	0.943	-662.6407	1660.1389
	7	520.54091	353.2718	0.925	-640.8489	1681.9307
	8	-840.28364	353.2718	0.39	-2001.6734	321.1062
	9	-695.56545	353.2718	0.67	-1856.9553	465.8243
	11	340.69273	353.2718	0.997	-820.6971	1502.0825
11	1	192.22727	353.2718	1	-969.1625	1353.6171
	2	141.64364	353.2718	1	-1019.7462	1303.0334
	3	171.51091	353.2718	1	-989.8789	1332.9007
	4	180.93636	353.2718	1	-980.4534	1342.3262
	5	104.41818	353.2718	1	-1056.9716	1265.808
	6	158.05636	353.2718	1	-1003.3334	1319.4462
	7	179.84818	353.2718	1	-981.5416	1341.238
	8	-1180.97636	353.2718	0.043	-2342.3662	-19.5866
	9	-1036.25818	353.2718	0.126	-2197.648	125.1316
	10	-340.69273	353.2718	0.997	-1502.0825	820.6971

Table D.8: Test 3 - Games-Howell test results of 1 - 5 paths (path 11 is navigation model)

(I) path	(J) path	Mean Difference (I-J)	Std. Error	Sig.	95 % Confidence Interval	
					Lower Bound	Upper Bound
1	2	-50.58364	342.97568	1	-1290.7132	1189.5459
	3	-20.71636	335.76726	1	-1233.9694	1192.5367
	4	-11.29091	334.72924	1	-1220.7089	1198.1271
	5	-87.80909	332.05947	1	-1287.4069	1111.7887
	6	-34.17091	333.71236	1	-1239.8411	1171.4993
	7	-12.37909	320.29473	1	-1169.5219	1144.7637
	8	-1373.20364	361.33957	0.035	-2683.46	-62.9472
	9	-1228.48545	345.08637	0.056	-2476.5607	19.5898
	10	-532.92	358.84419	0.908	-1833.5252	767.6852
	11	-192.22727	347.57627	1	-1449.7171	1065.2626
2	1	50.58364	342.97568	1	-1189.5459	1290.7132
	3	29.86727	351.88953	1	-1241.4302	1301.1647
	4	39.29273	350.89921	1	-1228.4991	1307.0845
	5	-37.22545	348.35339	1	-1296.053	1221.6021
	6	16.41273	349.92932	1	-1247.9558	1280.7812
	7	38.20455	337.15769	1	-1182.0896	1258.4987
	8	-1322.62	376.36801	0.061	-2683.4046	38.1646
	9	-1177.90182	360.79246	0.099	-2481.1498	125.3461
	10	-482.33636	373.97292	0.96	-1834.1534	869.4807
	11	-141.64364	363.17469	1	-1453.5649	1170.2776
3	1	20.71636	335.76726	1	-1192.5367	1233.9694
	2	-29.86727	351.88953	1	-1301.1647	1241.4302
	4	9.42545	343.85693	1	-1232.6356	1251.4865
	5	-67.09273	341.25858	1	-1299.8268	1165.6414
	6	-13.45455	342.86713	1	-1251.9547	1225.0456
	7	8.33727	329.82213	1	-1184.2079	1200.8825
	8	-1352.48727	369.81105	0.046	-2690.8859	-14.0887
	9	-1207.76909	353.94707	0.075	-2486.6465	71.1083
	10	-512.20364	367.37322	0.936	-1841.3418	816.9345
	11	-171.51091	356.37507	1	-1459.3775	1116.3556
4	1	11.29091	334.72924	1	-1198.1271	1220.7089
	2	-39.29273	350.89921	1	-1307.0845	1228.4991
	3	-9.42545	343.85693	1	-1251.4865	1232.6356
	5	-76.51818	340.23731	1	-1305.5329	1152.4965
	6	-22.88	341.85067	1	-1257.694	1211.934
	7	-1.08818	328.76535	1	-1189.6691	1187.4928
	8	-1361.91273	368.86885	0.043	-2697.1361	-26.6894
	9	-1217.19455	352.96253	0.069	-2492.6061	58.217
	10	-521.62909	366.42475	0.928	-1847.5492	804.291
	11	-180.93636	355.39724	1	-1465.3835	1103.5107
5	1	87.80909	332.05947	1	-1111.7887	1287.4069
	2	37.22545	348.35339	1	-1221.6021	1296.053
	3	67.09273	341.25858	1	-1165.6414	1299.8268
	4	76.51818	340.23731	1	-1152.4965	1305.5329
	6	53.63818	339.23694	1	-1171.7427	1279.0191
	7	75.43	326.04674	1	-1102.9942	1253.8542
	8	-1285.39455	366.44788	0.063	-2612.5097	41.7206
	9	-1140.67636	350.43169	0.101	-2407.227	125.8743
	10	-445.11091	363.98753	0.972	-1762.8121	872.5903
	11	-104.41818	352.88387	1	-1380.1246	1171.2882

Table D.9: Test 3 - Games-Howell test results of 6 - 11 paths (path 11 is navigation model)

(I) path	(J) path	Mean Difference (I-J)	Std. Error	Sig.	95 % Confidence Interval	
					Lower Bound	Upper Bound
6	1	34.17091	333.71236	1	-1171.4993	1239.8411
	2	-16.41273	349.92932	1	-1280.7812	1247.9558
	3	13.45455	342.86713	1	-1225.0456	1251.9547
	4	22.88	341.85067	1	-1211.934	1257.694
	5	-53.63818	339.23694	1	-1279.0191	1171.7427
	7	21.79182	327.72996	1	-1162.9138	1206.4974
	8	-1339.03273	367.94633	0.048	-2671.1578	-6.9077
	9	-1194.31455	351.99832	0.077	-2466.342	77.7129
	10	-498.74909	365.49606	0.944	-1821.5288	824.0306
	11	-158.05636	354.43966	1	-1439.165	1123.0523
7	1	12.37909	320.29473	1	-1144.7637	1169.5219
	2	-38.20455	337.15769	1	-1258.4987	1182.0896
	3	-8.33727	329.82213	1	-1200.8825	1184.2079
	4	1.08818	328.76535	1	-1187.4928	1189.6691
	5	-75.43	326.04674	1	-1253.8542	1102.9942
	6	-21.79182	327.72996	1	-1206.4974	1162.9138
	8	-1360.82455	355.82197	0.034	-2653.2807	-68.3684
	9	-1216.10636	339.30457	0.054	-2444.5937	12.381
	10	-520.54091	353.28762	0.912	-1803.0846	762.0028
	11	-179.84818	341.83658	1	-1418.0373	1058.341
8	1	1373.20364	361.33957	0.035	62.9472	2683.46
	2	1322.62	376.36801	0.061	-38.1646	2683.4046
	3	1352.48727	369.81105	0.046	14.0887	2690.8859
	4	1361.91273	368.86885	0.043	26.6894	2697.1361
	5	1285.39455	366.44788	0.063	-41.7206	2612.5097
	6	1339.03273	367.94633	0.048	6.9077	2671.1578
	7	1360.82455	355.82197	0.034	68.3684	2653.2807
	9	144.71818	378.29243	1	-1222.7278	1512.1641
	10	840.28364	390.88325	0.559	-571.66	2252.2272
	11	1180.97636	380.56514	0.134	-194.3868	2556.3396
9	1	1228.48545	345.08637	0.056	-19.5898	2476.5607
	2	1177.90182	360.79246	0.099	-125.3461	2481.1498
	3	1207.76909	353.94707	0.075	-71.1083	2486.6465
	4	1217.19455	352.96253	0.069	-58.217	2492.6061
	5	1140.67636	350.43169	0.101	-125.8743	2407.227
	6	1194.31455	351.99832	0.077	-77.7129	2466.342
	7	1216.10636	339.30457	0.054	-12.381	2444.5937
	8	-144.71818	378.29243	1	-1512.1641	1222.7278
	10	695.56545	375.90961	0.739	-662.9973	2054.1282
	11	1036.25818	365.16865	0.213	-282.8038	2355.3202
10	1	532.92	358.84419	0.908	-767.6852	1833.5252
	2	482.33636	373.97292	0.96	-869.4807	1834.1534
	3	512.20364	367.37322	0.936	-816.9345	1841.3418
	4	521.62909	366.42475	0.928	-804.291	1847.5492
	5	445.11091	363.98753	0.972	-872.5903	1762.8121
	6	498.74909	365.49606	0.944	-824.0306	1821.5288
	7	520.54091	353.28762	0.912	-762.0028	1803.0846
	8	-840.28364	390.88325	0.559	-2252.2272	571.66
	9	-695.56545	375.90961	0.739	-2054.1282	662.9973
	11	340.69273	378.19663	0.997	-1025.8858	1707.2713
11	1	192.22727	347.57627	1	-1065.2626	1449.7171
	2	141.64364	363.17469	1	-1170.2776	1453.5649
	3	171.51091	356.37507	1	-1116.3556	1459.3775
	4	180.93636	355.39724	1	-1103.5107	1465.3835
	5	104.41818	352.88387	1	-1171.2882	1380.1246
	6	158.05636	354.43966	1	-1123.0523	1439.165
	7	179.84818	341.83658	1	-1058.341	1418.0373
	8	-1180.97636	380.56514	0.134	-2556.3396	194.3868
	9	-1036.25818	365.16865	0.213	-2355.3202	282.8038
	10	-340.69273	378.19663	0.997	-1707.2713	1025.8858

Table D.10: Test 2 - Homogeneous Subsets cost

path	N	Subset for alpha = 0.05		
		1	2	3
1	11	651.3318		
4	11	662.6227		
7	11	663.7109		
3	11	672.0482		
6	11	685.5027		
2	11	701.9155		
5	11	739.1409	739.1409	
11	11	843.5591	843.5591	
10	11	1184.2518	1184.2518	1184.2518
9	11		1879.8173	1879.8173
8	11			2024.5355
Sig.		0.914	0.059	0.39

Appendix E

Coding

E.1 HRVO Code

```
Index: indra/llcommon/llsys.h
===================================================================
--- indra/llcommon/llsys.h (revision 29)
+++ indra/llcommon/llsys.h (revision 32)
@@ -33,7 +33,7 @@
 #ifndef LL_SYS_H
 #define LL_SYS_H

-#define EXE_APP 0 //set to 1 for exe and 0 for dll
+#define EXE_APP 1 //set to 1 for exe and 0 for dll

 //
 // The LLOSInfo, LLCPUInfo, and LLMemoryInfo classes are essentially
Index: indra/newview/llagent.h
===================================================================
--- indra/newview/llagent.h (revision 29)
+++ indra/newview/llagent.h (revision 32)
@@ -1,1100 +1,1139 @@
 const LLQuaternion *target_rotation = NULL,
 void (*finish_callback)(BOOL, void *) = NULL, void *callback_data = NULL,
 F32 stop_distance = 0.f, F32 rotation_threshold = 0.03f);
+ //@@
+ void startAutoPilotWithNavigation(const LLVector3d &target_global,
+ /*HRVO::HRVOSimulator* HRVOInstance1,*/
+ const std::string& behavior_name = std::string(),
+ const LLQuaternion *target_rotation = NULL,
+ void (*finish_callback)(BOOL, void *) = NULL,  void *callback_data = NULL,
+ F32 stop_distance = 0.f, F32 rot_threshold = 0.03f);
 void  startFollowPilot(const LLUUID &leader_id);
 void stopAutoPilot(BOOL user_cancel = FALSE);
 void  setAutoPilotGlobal(const LLVector3d &pos_global);
 void autoPilot(F32 *delta_yaw);  // Autopilot walking action, angles in radians
+ void autoPilotWithNavigation(F32 *delta_yaw);
 void renderAutoPilotTarget();

+ //*************NAVI******************//
+ //@@
+ const float NEIGHBOR_DIST;
+ const int MAX_NEIGHBORS;
```

```
+ const float RADIUS;
+ const float GOAL_RADIUS;
+ const float PREF_SPEED;
+ const float MAX_SPEED;
+ const float UNCERTAINTY_OFFSET;
+ const float MAX_ACCEL;
+ const float VELOCITY_X;
+ const float VELOCITY_Y;
+ const float ORIENTATION;
+ const int CLASS_NO;
+ const int AgentNo;
+ const int GoalNo;
+ const float HRVOTimeStep;
+ BOOL mHRVOInitialized;
+ Vector2 LastAgentPos;
+ float LastAgentOrientation;
+ HRVOSimulator* HRVOInstance1;
+ //****************************************//
+

Index: indra/newview/llviewerobjectlist.h
===================================================================
--- indra/newview/llviewerobjectlist.h (revision 29)
+++ indra/newview/llviewerobjectlist.h (revision 32)
@@ -1,310 +1,320 @@

// Global object list
extern LLViewerObjectList gObjectList;
+extern std::map<std::string, LLVector3> HRVO_objectDB;
+extern std::map<std::string, LLVector3>::size_type NumErased;
// Inlines
/**
 * Note:
 * it will return NULL for offline avatar_id
 */


Index: indra/newview/llappviewer.cpp
===================================================================
--- indra/newview/llappviewer.cpp (revision 29)
+++ indra/newview/llappviewer.cpp (revision 32)
@@ -1,4482 +1,4568 @@
+BOOL NavigationON = TRUE;
+BOOL PRINT_AGENTPOS = FALSE;
//////////////////////////////////////////////////////
// idle()

@@ -1,4482 +1,4568 @@
F32 frame_rate_clamped = 1.f / dt_raw;
frame_rate_clamped = llclamp(frame_rate_clamped, MIN_FRAME_RATE, MAX_FRAME_RATE);
gFrameDTClamped = 1.f / frame_rate_clamped;
+//************NAVI***************//
+ //@@OK
+
+ if(gKeyboard->getCurScanKey() == 56)
+ {
+ //std::cout<<"Navigation Model: ON"<<std::endl;
```

```
+ //NavigationON = TRUE;
+
+ PRINT_AGENTPOS = TRUE;
+
+ //std::cout<<"Agent Position : "<<gAgent.getPositionAgent().mV[VX]<<", "<<gAgent.getPositionAgent().mV[VY]<<std::en
+ }
+ if(gKeyboard->getCurScanKey() == 55)
+ {
+ //std::cout<<"Navigation Model: OFF"<<std::endl;
+ //NavigationON = FALSE;
+ PRINT_AGENTPOS = FALSE;
+ std::cout<<"--------------------------------------------"<<std::endl;
+ }
+
+//*******************************//
// Global frame timer
// Smoothly weight toward current frame

@@ -1,4482 +1,4568 @@
     // Handle automatic walking towards points
     gAgentPilot.updateTarget();
     gAgent.autoPilot(&yaw);
+    /@@OK
+ if(NavigationON){
+ gAgent.autoPilotWithNavigation(&yaw);
+ }
+ else{
+ gAgent.autoPilot(&yaw);
+ }
+
+ if(PRINT_AGENTPOS){
+ std::cout<<"Agent Position : "<<gAgent.getPositionAgent().mV[VX]<<", "<<gAgent.getPositionAgent().mV[VY]<<std::endl
+ }
+
     static LLFrameTimer agent_update_timer;
     static U32  last_control_flags;

@@ -1,4482 +1,4568 @@
{
char objectId[128];
float tx=0,ty=0,tz=0;

+ //@@
+ while(!AppropriateTarget)
+ {
getNavModelTarget(objectId, &tx, &ty,&tz);//Get a randon object from within the fov

selectedObject =gObjectList.findObjectFromStringID(objectId);// detect the object in the SL scene

if(selectedObject)
{
LLVector3 targetPosition = selectedObject->getPosition();

+ //*************NAVI***************//
+ //@@
+ if(abs(targetPosition.mV[VX])>1.0f && abs(targetPosition.mV[VY])>1.0f && abs(targetPosition.mV[VZ])>1.0f){
+
```

```
+ AppropriateTarget = TRUE;
+
+ LLVector3d fakeTarget = LLVector3d(254959, 255030, 26.8464);
+ if(NavigationON)
+ {
+ std::cout<<"AUTOPILOT MODE (Navigation ON)"<<std::endl;
+ gAgent.startAutoPilotWithNavigation(selectedObject->getPositionGlobal());//start navigation avoiding obstacles
+ //gAgent.startAutoPilotWithNavigation(fakeTarget);
+ }
+ else{
+ std::cout<<"AUTOPILOT MODE (Navigation OFF)"<<std::endl;
gAgent.startAutoPilotGlobal(selectedObject->getPositionGlobal());//start navigation toward selected objet
+ //gAgent.startAutoPilotGlobal(fakeTarget);
+ }
+
+ //std::cout<<"Target ID:"<<objectId<<std::endl;
+ //std::cout<<"Target Position Global:"<<selectedObject->getPositionGlobal()<<std::endl;
+ //std::cout<<"Target Position Agent:"<<selectedObject->getPositionAgent()<<std::endl;
+ //std::cout<<"Position:"<<fakeTarget<<std::endl;
+
+ }
+ else{
+ AppropriateTarget = FALSE;
+ }
+
agentNavigation_timer.reset();
}
+ }
}
}
else
{

Index: indra/newview/llviewerdisplay.cpp
===================================================================
--- indra/newview/llviewerdisplay.cpp (revision 29)
+++ indra/newview/llviewerdisplay.cpp (revision 32)
@@ -198,6 +198,7 @@
  {
  F32 fps = gRecentFrameCount / fps_log_freq;
  llinfos << llformat("FPS: %.02f", fps) << llendl;
+ std::cout<<"FPS: "<<fps<<std::endl;
  gRecentFrameCount = 0;
  gRecentFPSTime.reset();
  }
Index: indra/newview/llviewerobjectlist.cpp
===================================================================
--- indra/newview/llviewerobjectlist.cpp (revision 29)
+++ indra/newview/llviewerobjectlist.cpp (revision 32)
@@ -1,1773 +1,1819 @@
extern LLPipeline gPipeline;

+//**********NAVI************//
+//@@
+std::map<std::string, LLVector3> HRVO_objectDB;
+std::map<std::string, LLVector3>::size_type NumErased;
+//*************************//
```

```
// Statics for object lookup tables.
U32 LLViewerObjectList::sSimulatorMachineIndex = 1; // Not zero deliberately, to speed up index check.
LLMap<U64, U32>LLViewerObjectList::sIPAndPortToIndex;
std::map<U64, LLUUID>LLViewerObjectList::sIndexAndLocalIDToUUID;

@@ -1,1773 +1,1819 @@
removeNode( objectId);
//-----------------------------------------------
+       //***************NAVI****************//
+ //@@
+ std::string str;
+ objectp->getID().toString(str);
+ HRVO_objectDB.erase(str);
+ //*********************************//
return;
}

updateActive(objectp);

@@ -1,1773 +1,1819 @@

found=str.find(str2);
if(found ==std::string::npos)
{

updateNode(objectId,position[0],position[2], position[1],ax,az,ay,aw);
+ //****************NAVI*****************//
+ //@@ToDo: consider distance
+ HRVO_objectDB.insert(std::map<std::string, LLVector3>::value_type(str, position));
+ //*********************************//
//std::cout<<"OBJECT POSITION:"<<objectId<<std::endl;
//std::cout<<"distance: "<<distance<<std::endl;
if(objectp->isAvatar())// if the object is an avatar, add more body parts to enhance attention towards it

@@ -1,1773 +1,1819 @@

char objectId[128];
objectp->getID().toString(objectId);
removeNode( objectId);
+ //************NAVI**************//
+ //@@
+ HRVO_objectDB.erase(str);
+ //*******************************//

if(objectp->isAvatar())// if the object is an avatar, delete body parts previously added
{
char tmpid[128];
//remove Left Eyes

@@ -1,1773 +1,1819 @@
if (objectp->onActiveList())
{
//llinfos << "Removing " << objectp->mID << " " << objectp->getPCodeString() << " from active list in cleanupReferenc
objectp->setOnActiveList(FALSE);
mActiveObjects.erase(objectp);
}
+ //**************NAVI*************//
```

```
+ //@@
+ std::string str;
+ objectp->getID().toString(str);
+ HRVO_objectDB.erase(str);
+ //******************************//

if (objectp->isOnMap())
{
mMapObjects.removeObj(objectp);
}
@@ -1,1773 +1,1819 @@
//-----------attentionmodel------------
char objectId[128];
objectp->getID().toString(objectId);

removeNode( objectId);
//-------------------------------------------------
+       //******************NAVI***************//
+ //@@
+ std::string str;
+ objectp->getID().toString(str);
+ HRVO_objectDB.erase(str);
+ //************************************//
if (objectp->isDead())
{
// This object is already dead.  Don't need to do more.
return TRUE;

@@ -1,1773 +1,1819 @@
/*char *objectId = new char[str.size()];
std::copy(str.begin(), str.end(), objectId);*/

removeNode( objectId);
//-------------------------------------------------
+           //******************NAVI******************//
+ //@@
+ std::string str;
+ objectp->getID().toString(str);
+ HRVO_objectDB.erase(str);
+ //****************************************//
mObjects.remove(i);
num_removed++;

if (num_removed == mNumDeadObjects)

@@ -1,1773 +1,1819 @@
//std::copy(str.begin(), str.end(), objectId);

removeNode( objectId);
//-------------------------------------------------
+ //*************NAVI*************//
+ //@@
+ HRVO_objectDB.erase(str);
+ //******************************//
//llinfos << "Removing " << objectp->mID << " " << objectp->getPCodeString() << " from active list." << llendl;
mActiveObjects.erase(objectp);
objectp->setOnActiveList(FALSE);
```

```
}
}
}

Index: indra/newview/llagent.cpp
===================================================================
--- indra/newview/llagent.cpp (revision 29)
+++ indra/newview/llagent.cpp (revision 32)
@@ -80,6 +80,7 @@
 #include "llworld.h"
 #include "llworldmap.h"

+
 using namespace LLVOAvatarDefines;

 extern LLMenuBarGL* gMenuBarView;
@@ -172,6 +173,13 @@
 std::map<std::string, std::string> LLAgent::sTeleportErrorMessages;
 std::map<std::string, std::string> LLAgent::sTeleportProgressMessages;

+//************NAVI***********//
+//@@
+extern std::map<std::string, LLVector3> HRVO_objectDB;
+int ORIENTATION_KEEP_COUNT=0;
+float HRVO_Timer = 0.f;
+//**************************//
+
 class LLAgentFriendObserver : public LLFriendObserver
 {
 public:
@@ -350,7 +358,32 @@
   mFirstLogin(FALSE),
   mGenderChosen(FALSE),

-mAppearanceSerialNum(0)
+ mAppearanceSerialNum(0),
+
+ //**************NAVI**********************//
+ //@@
+ //-------Default Agent Parameters-----------//
+ NEIGHBOR_DIST(3.0f),
+ MAX_NEIGHBORS(6),
+ RADIUS(1.0f),
+ GOAL_RADIUS(1.0f),
+ PREF_SPEED(0.5f),
+ MAX_SPEED(2.0f),
+ UNCERTAINTY_OFFSET(1.0f),
+ MAX_ACCEL(0.5f),
+ VELOCITY_X(0.0f),
+ VELOCITY_Y(0.0f),
+ ORIENTATION(0.0f),
+ CLASS_NO(0),
+ AgentNo(0),
+ GoalNo(0),
+ HRVOTimeStep(1.f),
+ mHRVOInitialized(FALSE),
+ LastAgentPos(0.f,0.f),
```

```
+ LastAgentOrientation(0.f),
+ //@@ Define HRVO Instance and Initialize here
+ HRVOInstance1(HRVO::HRVOSimulator::Instance())
+ //*************************************//
 {
  for (U32 i = 0; i < TOTAL_CONTROLS; i++)
  {
@@ -361,6 +394,7 @@
  mFollowCam.setMaxCameraDistantFromSubject( MAX_CAMERA_DISTANCE_FROM_AGENT );

  mListener.reset(new LLAgentListener(*this));
+
 }

 // Requires gSavedSettings to be initialized.
@@ -408,6 +442,25 @@
  gSavedSettings.getControl("PreferredMaturity")->getValidateSignal()->connect(boost::bind(&LLAgent::validateMaturity, this,
  gSavedSettings.getControl("PreferredMaturity")->getSignal()->connect(boost::bind(&LLAgent::handleMaturity, this, _2));

+ //*******************NAVI********************//
+ //@@ Define HRVO Initialize here
+ //Vector2 InitialVelocity = Vector2(VELOCITY_X,VELOCITY_Y);
+
+ //Consider Agent default params
+ //HRVOInstance1->setAgentDefaults(NEIGHBOR_DIST, MAX_NEIGHBORS, RADIUS, GOAL_RADIUS, PREF_SPEED, MAX_SPEED, UNCERTAINTY_OFF
+
+ //Initialize goal and agent position
+ //Vector2 GoalPos = Vector2(0.0f, 0.0f);
+ //Vector2 AgentPos = Vector2(10.0f,0.0f);
+
+ //HRVOInstance1->addGoal(GoalPos);
+ //HRVOInstance1->addAgent(AgentPos,GoalNo);
+
+ //HRVOInstance1->setTimeStep(0.3f);
+ //HRVOInstance1->initSimulation();
+
+ //*************************************//
+
  mInitialized = TRUE;
 }

@@ -2183,6 +2236,7 @@
 {
  if (!gAgent.getAvatarObject())
  {
+ std::cout<<"No object is chosen."<<std::endl;
  return;
  }

@@ -2192,6 +2246,9 @@
  mAutoPilotBehaviorName = behavior_name;

  LLVector3d delta_pos( target_global );
+ //std::cout<<"target_global:"<< delta_pos<<std::endl;
+ //std::cout<<"agent_global:"<< getPositionGlobal()<<std::endl;
+
  delta_pos -= getPositionGlobal();
```

```
  F64 distance = delta_pos.magVec();
  LLVector3d trace_target = target_global;
@@ -2221,13 +2278,15 @@

  if (distance > 30.0)
  {
-setFlying(TRUE);
+ std::cout<<"I wanna fly because distance is long."<<std::endl;
+ //setFlying(TRUE);
  }

  if ( distance > 1.f && heightDelta > (sqrtf(mAutoPilotStopDistance) + 1.f))
  {
-setFlying(TRUE);
-mAutoPilotFlyOnStop = TRUE;
+ std::cout<<"Arrived Target, but keep flying"<<std::endl;
+ //setFlying(TRUE);
+ //mAutoPilotFlyOnStop = TRUE;
  }

  mAutoPilot = TRUE;
@@ -2262,7 +2321,102 @@
  mAutoPilotNoProgressFrameCount = 0;
 }

+//*************NAVI****************//
+//@@
+//-----------------------------------------------------------------------------
+// startAutoPilotWithNavigation()
+//-----------------------------------------------------------------------------
+void LLAgent::startAutoPilotWithNavigation(const LLVector3d &target_global, const std::string& behavior_name, const
+{
+// std::cout<<"Target Position Global: "<<target_global<<std::endl;

+ if (!gAgent.getAvatarObject())
+ {
+ return;
+ }
+
+ mAutoPilotFinishedCallback = finish_callback;
+ mAutoPilotCallbackData = callback_data;
+ mAutoPilotRotationThreshold = rot_threshold;
+ mAutoPilotBehaviorName = behavior_name;
+
+ LLVector3d delta_pos( target_global );
+
+ delta_pos -= getPositionGlobal();
+ F64 distance = delta_pos.magVec();
+ LLVector3d trace_target = target_global;
+
+ trace_target.mdV[VZ] -= 10.f;
+
+ LLVector3d intersection;
+ LLVector3 normal;
+ LLViewerObject *hit_obj;
+ F32 heightDelta = LLWorld::getInstance()->resolveStepHeightGlobal(NULL, target_global, trace_target, intersection,
+
```

```
+ if (stop_distance > 0.f)
+ {
+ mAutoPilotStopDistance = stop_distance;
+ }
+ else
+ {
+ // Guess at a reasonable stop distance.
+ mAutoPilotStopDistance = fsqrtf( distance );
+ if (mAutoPilotStopDistance < 0.5f)
+ {
+ mAutoPilotStopDistance = 0.5f;
+ }
+ }
+
+ mAutoPilotFlyOnStop = getFlying();
+
+ if (distance > 30.0)
+ {
+ //std::cout<<"I wanna fly because distance is long."<<std::endl;
+ //setFlying(TRUE);
+ }
+
+ if ( distance > 1.f && heightDelta > (sqrtf(mAutoPilotStopDistance) + 1.f))
+ {
+ //std::cout<<"Arrived Target, but keep flying"<<std::endl;
+ //setFlying(TRUE);
+ //mAutoPilotFlyOnStop = TRUE;
+ }
+
+ mAutoPilot = TRUE;
+ mAutoPilotTargetGlobal = target_global;
+
+ // trace ray down to find height of destination from ground
+ LLVector3d traceEndPt = target_global;
+ traceEndPt.mdV[VZ] -= 20.f;
+
+ LLVector3d targetOnGround;
+ LLVector3 groundNorm;
+ LLViewerObject *obj;
+
+ LLWorld::getInstance()->resolveStepHeightGlobal(NULL, target_global, traceEndPt, targetOnGround, groundNorm, &obj);
+ F64 target_height = llmax((F64)gAgent.getAvatarObject()->getPelvisToFoot(), target_global.mdV[VZ] - targetOnGround.mdV[VZ])
+
+ // clamp z value of target to minimum height above ground
+ mAutoPilotTargetGlobal.mdV[VZ] = targetOnGround.mdV[VZ] + target_height;
+ mAutoPilotTargetDist = (F32)dist_vec(gAgent.getPositionGlobal(), mAutoPilotTargetGlobal);
+ if (target_rotation)
+ {
+ mAutoPilotUseRotation = TRUE;
+ mAutoPilotTargetFacing = LLVector3::x_axis * *target_rotation;
+ mAutoPilotTargetFacing.mV[VZ] = 0.f;
+ mAutoPilotTargetFacing.normalize();
+ }
+ else
+ {
+ mAutoPilotUseRotation = FALSE;
+ }
```

```
+
+//-----------------------------------------------------------------------------------------------------------
+ mAutoPilotNoProgressFrameCount = 0;
+}
+//*****************************************//
+
+
 //-----------------------------------------------------------------------------
 // startFollowPilot()
 //-----------------------------------------------------------------------------
@@ -2319,6 +2473,15 @@
  else
  LLNotificationsUtil::add("Cancelled");
  }
+
+ //***********NAVI**************//
+ //@@
+ if(mHRVOInitialized){
+ mHRVOInitialized = FALSE;
+ HRVOInstance1->~HRVOSimulator();
+// std::cout<<"#######HRVO Simulator is deleted.##############"<<std::endl;
+ }
+ //*****************************//
  }
 }

@@ -2349,7 +2512,8 @@

  if (mAvatarObject->mInAir)
  {
-setFlying(TRUE);
+ //std::cout<<"Object is in air."<<std::endl;
+ //setFlying(TRUE);
  }

  LLVector3 at;
@@ -2377,7 +2541,7 @@

  at.normalize();
  F32 xy_distance = direction.normalize();
-
+
  F32 yaw = 0.f;
  if (mAutoPilotTargetDist > mAutoPilotStopDistance)
  {
@@ -2481,7 +2645,364 @@
}


+//*************NAVI***************//
+//
+// Returns necessary agent pitch and yaw changes, radians.
 //-----------------------------------------------------------------------------
+// autoPilotWithNavigation()
+//-----------------------------------------------------------------------------
+//@@OK without HRVO process
+void LLAgent::autoPilotWithNavigation(F32 *delta_yaw)
```

```
+{
+ //**************NAVI**************//
+ //@@ Variables for HRVO Simulation
+ static LLFrameTimer SL_Timer;
+
+ //-----for obstacles-----//
+ static LLViewerObject* HRVOObstacle;
+ LLBBox HRVObbox;
+ int RoadVer[4];
+ int ObstacleCount = 0;
+ LLVector3 CheckGoalObj;
+ F32 BBoxDist;
+ LLVector3 ScaleOffset;
+ LLVector3 BBoxMin_global;
+ LLVector3 BBoxMax_global;
+ int VertexNo[4];
+ std::map<std::string, LLVector3>::iterator DB_itr = HRVO_objectDB.begin();
+
+ //-----for agent-----//
+ LLVector3 agent_nav;
+ Vector2 agent_nav2D;
+ Vector2 InitialVelocity;
+ float orientation_global;
+ F32 yaw2 = 0.f;
+ float agent_speed;
+ float initial_orientation;
+ float init_velocityX;
+ float init_velocityY;
+
+ //-----for target-----//
+ LLVector3 target_nav;
+ Vector2 target_nav2D;
+ LLVector3 direction_nav;
+ //************************************//
+
+
+ if (mAutoPilot)
+ {
+ if (!mLeaderID.isNull())
+ {
+ LLViewerObject* object = gObjectList.findObject(mLeaderID);
+ if (!object)
+ {
+ //std::cout<<"Unable Object"<<std::endl;
+ stopAutoPilot();
+ return;
+ }
+ mAutoPilotTargetGlobal = object->getPositionGlobal();
+ }
+
+ if (mAvatarObject.isNull())
+ {
+ return;
+ }
+
+ if (mAvatarObject->mInAir)
+ {
```

```
+ //std::cout<<"Object is in air."<<std::endl;
+ //setFlying(TRUE);
+ }
+
+ LLVector3 at;
+ at.setVec(mFrameAgent.getAtAxis());
+ LLVector3 target_agent = getPosAgentFromGlobal(mAutoPilotTargetGlobal);
+ LLVector3 direction = target_agent - getPositionAgent();
+
+ F32 target_dist = direction.magVec();
+
+ if (target_dist >= mAutoPilotTargetDist)
+ {
+ mAutoPilotNoProgressFrameCount++;
+ if (mAutoPilotNoProgressFrameCount > AUTOPILOT_MAX_TIME_NO_PROGRESS * gFPSClamped)
+ {
+ //std::cout<<"Time Out: Stop Autopilot."<<std::endl;
+ stopAutoPilot();
+ return;
+ }
+ }
+
+ mAutoPilotTargetDist = target_dist;
+
+ // Make this a two-dimensional solution
+ at.mV[VZ] = 0.f;
+ direction.mV[VZ] = 0.f;
+
+ //@@
+ F32 targetDist2D = direction.magVec();//<--length until target on xy coordinate
+ //mAutoPilotTargetDist = targetDist2D;//<--update the length
+
+ at.normalize();
+ F32 xy_distance = direction.normalize();
+
+ F32 yaw = 0.f;
+
+ ////*************NAVI****************//
+ //-----get agent position from SL and change to 2D vector-----//
+ agent_nav = getPositionAgent();
+ agent_nav2D = Vector2(agent_nav.mV[VX],agent_nav.mV[VY]);
+// std::cout<<"Agent Position : "<<getPositionAgent().mV[VX]<<", "<<getPositionAgent().mV[VY]<<std::endl;
+
+ //-----HRVO Initialization-----//
+ if(!mHRVOInitialized){
+
+ mHRVOInitialized = TRUE;
+ HRVOInstance1 = HRVOSimulator::Instance();
+ HRVO_Timer = 0.f;
+
+ //-----goal setting-----//
+ target_nav = getPosAgentFromGlobal(mAutoPilotTargetGlobal); //<--change LLVector3d to LLVector3.
+ target_nav2D = Vector2(target_nav.mV[VX], target_nav.mV[VY]);
+ HRVOInstance1->addGoal(target_nav2D);
+
+ //-----initial agent setting-----//
+ ORIENTATION_KEEP_COUNT=0;
```

```
+ initial_orientation  = angle_between(at, LLVector3(1.0f, 0.0f, 0.0f));
+ LastAgentOrientation = initial_orientation;
+ init_velocityX       = cos(initial_orientation);
+ init_velocityY      = sin(initial_orientation);
+ InitialVelocity      = Vector2(init_velocityX, init_velocityY)*PREF_SPEED;
+
+ HRVOInstance1->setAgentDefaults(NEIGHBOR_DIST, MAX_NEIGHBORS, RADIUS, GOAL_RADIUS, PREF_SPEED, MAX_SPEED,
+ UNCERTAINTY_OFFSET, MAX_ACCEL, InitialVelocity, initial_orientation, CLASS_NO);
+
+ HRVOInstance1->addAgent(agent_nav2D, GoalNo);
+ HRVOInstance1->setAgentGoal(AgentNo,GoalNo);
+// std::cout<<"Initial Agent Pos in HRVO: "<<HRVOInstance1->getAgentPosition(AgentNo)<<std::endl;
+// std::cout<<"Initial Goal  Pos in HRVO: "<<HRVOInstance1->getGoalPosition(GoalNo)<<std::endl;
+
+ //-----obstacle setting-----//
+ while(DB_itr != HRVO_objectDB.end())
+ {
+ HRVOObstacle = gObjectList.findObjectFromStringID((*DB_itr).first);
+ CheckGoalObj = LLVector3(HRVOObstacle->getPositionAgent() - target_nav);
+ CheckGoalObj.mV[VZ]=0.f;
+
+ if(abs(CheckGoalObj.magVec())>0.01f && !HRVOObstacle->isDead())
+ {
+ if(abs(HRVOObstacle->getPosition().mV[VX])>1.0f && abs(HRVOObstacle->getPositionAgent().mV[VY])>1.0f && abs(HRVOObstacle->g
+ {
+ if(!HRVOObstacle->isAvatar())
+ {
+ HRVObbox = HRVOObstacle->getBoundingBoxAgent();
+ BBoxDist = dist_vec(HRVObbox.getMinLocal(), HRVObbox.getMaxLocal());
+ ScaleOffset = LLVector3(0.1f,0.1f,0.1f); //LLVector3(BBoxDist*0.3f, BBoxDist*0.3f, BBoxDist*0.3f);
+ //LLVector3 BBoxScale = BBoxDist.magVec();
+ //std::cout<<"ID: "<<(*DB_itr).first<<", Pos: "<<HRVOObstacle->getPosition()<<", avatar: "<<agent_nav<<std::endl;
+ BBoxMin_global = HRVOObstacle->getPositionAgent() + HRVObbox.getMinLocal() - ScaleOffset;
+ BBoxMax_global = HRVOObstacle->getPositionAgent() + HRVObbox.getMaxLocal() + ScaleOffset;
+ Vector2 ObstacleVertexSet[] = {Vector2(BBoxMin_global.mV[VX], BBoxMin_global.mV[VY]),
+    Vector2(BBoxMin_global.mV[VX], BBoxMax_global.mV[VY]),
+    Vector2(BBoxMax_global.mV[VX], BBoxMax_global.mV[VY]),
+        Vector2(BBoxMax_global.mV[VX], BBoxMin_global.mV[VY])};
+
+ //std::cout<<"Obstacle "<<ObstacleCount<<std::endl;
+ for(int j=0;j<4;j++)//Create Obstacle Vertecies as Roadmap Vertecies
+ {
+ RoadVer[j] = HRVOInstance1->addRoadmapVertex(ObstacleVertexSet[j]);
+// std::cout<<"Obstacle"<<ObstacleCount<<", "<<HRVOInstance1->getRoadmapVertexPosition(RoadVer[j]).x()<<", "<<HRVOInstance1-
+ }
+
+
+ for(int j=0;j<4;j++)
+ {
+ int J = fmod((double)j+1, 4.0);
+ HRVOInstance1->addRoadmapEdge(RoadVer[j],RoadVer[J]);
+ VertexNo[j]=HRVOInstance1->addObstacle(ObstacleVertexSet[j],ObstacleVertexSet[J]);
+ }
+
+ ObstacleCount++;
+ }
+ }
```

```
+ }
+ else{
+ if(HRVOObstacle->isDead())
+ {
+ //std::cout<<"This is dead obstacle."<<std::endl;
+ }
+ else{
+ //std::cout<<"Found goal object. Avoid to put in HRVO obstacle list. "<<CheckGoalObj.magVec()<<std::endl;
+ //std::cout<<"Target Object Position : "<<HRVOObstacle->getPositionGlobal()<<" : "<<HRVOObstacle->getPositionAgent
+ }
+ }
+
+ ++DB_itr;
+ }//end while
+
+ HRVOInstance1->setTimeStep(HRVOTimeStep);
+ HRVO_Timer = 0.f;
+ HRVOInstance1->initSimulation();
+
+ //std::cout<<"default orientation: "<<initial_orientation * 180.0f / 3.141592f<<", initial velocity: "<<InitialVelo
+// std::cout<<"/////////HRVO Initialize is done!/////////"<<std::endl;
+ }
+ else{
+ // idle roop --- fix agent position in the simulator//
+ agent_speed = abs(agent_nav2D - LastAgentPos);
+ HRVOInstance1->setAgentPosition(AgentNo,agent_nav2D);
+ HRVOInstance1->setAgentPrefSpeed(AgentNo, agent_speed);
+ HRVO_Timer += HRVOInstance1->getTimeStep();
+ }
+
+ HRVOInstance1->doStep();//<--update process of HRVO simulation for one step
+
+ //@@
+ orientation_global = HRVOInstance1->getAgentOrientation(AgentNo);
+
+ //-----remmove when strange value-----//
+ if(abs(orientation_global - LastAgentOrientation) > 0.785f && ORIENTATION_KEEP_COUNT<2)
+ {
+ //std::cout<<"keep last orientation. curr: "<<orientation_global * 180.0f / 3.141592f<<", last: "<<LastAgentOrienta
+ orientation_global = LastAgentOrientation;
+ ORIENTATION_KEEP_COUNT++;
+ }
+ else{
+ ORIENTATION_KEEP_COUNT=0;
+ }
+
+ //-----remove when strange value-----//
+ if(HRVO_Timer<10.0f){
+ if(orientation_global == 0.0f || orientation_global == 180.0f){
+ orientation_global = LastAgentOrientation;
+ }
+ }
+
+
+ direction_nav = LLVector3(cos(orientation_global), sin(orientation_global),0.f);
+ direction_nav.normalize();
+
```

```
+ if (mAutoPilotTargetDist > mAutoPilotStopDistance)
+ {
+ //@@
+ yaw2 = angle_between(mFrameAgent.getAtAxis(), direction_nav);
+ yaw = angle_between(mFrameAgent.getAtAxis(), direction);//<-origibnal code
+ }
+ else if (mAutoPilotUseRotation)
+ {
+ // we're close now just aim at target facing
+ yaw = angle_between(at, mAutoPilotTargetFacing);
+ yaw2 = angle_between(at, mAutoPilotTargetFacing);
+ direction = mAutoPilotTargetFacing;
+ direction_nav = mAutoPilotTargetFacing;
+ }
+
+ yaw = 4.f * yaw / gFPSClamped;
+ //@@
+ yaw2 = 4.f * yaw2 / gFPSClamped;
+
+ // figure out which direction to turn
+ //LLVector3 scratch(at % direction);
+ LLVector3 scratch(at % direction_nav);
+ //std::cout<<"HRVOTime: "<<HRVO_Timer<<", orientation_global: "<< orientation_global / 3.141592f * 180.0f<<", agentPos: "<<
+
+ if (scratch.mV[VZ] > 0.f)
+ {
+ setControlFlags(AGENT_CONTROL_YAW_POS);
+ }
+ else
+ {
+ //@@
+ yaw2 = -yaw2;
+ yaw = -yaw;
+ setControlFlags(AGENT_CONTROL_YAW_NEG);
+ }
+
+
+ //*delta_yaw = yaw;
+ *delta_yaw = yaw2;
+ //-----preserve position and orientation to check error value-----//
+ LastAgentPos = agent_nav2D;
+ LastAgentOrientation = orientation_global;
+
+ // Compute when to start slowing down and when to stop
+ F32 stop_distance = mAutoPilotStopDistance;
+ F32 slow_distance;
+ if (getFlying())
+ {
+ slow_distance = llmax(6.f, mAutoPilotStopDistance + 5.f);
+ stop_distance = llmax(2.f, mAutoPilotStopDistance);
+ }
+ else
+ {
+ slow_distance = llmax(3.f, mAutoPilotStopDistance + 2.f);
+ }
+
+ // If we're flying, handle autopilot points above or below you.
```

```
+ if (getFlying() && xy_distance < AUTOPILOT_HEIGHT_ADJUST_DISTANCE)
+ {
+ if (mAvatarObject.notNull())
+ {
+ F64 current_height = mAvatarObject->getPositionGlobal().mdV[VZ];
+ F32 delta_z = (F32)(mAutoPilotTargetGlobal.mdV[VZ] - current_height);
+ F32 slope = delta_z / xy_distance;
+ if (slope > 0.45f && delta_z > 6.f)
+ {
+ setControlFlags(AGENT_CONTROL_FAST_UP | AGENT_CONTROL_UP_POS);
+ }
+ else if (slope > 0.002f && delta_z > 0.5f)
+ {
+ setControlFlags(AGENT_CONTROL_UP_POS);
+ }
+ else if (slope < -0.45f && delta_z < -6.f && current_height > AUTOPILOT_MIN_TARGET_HEIGHT_OFF_GROUND)
+ {
+ setControlFlags(AGENT_CONTROL_FAST_UP | AGENT_CONTROL_UP_NEG);
+ }
+ else if (slope < -0.002f && delta_z < -0.5f && current_height > AUTOPILOT_MIN_TARGET_HEIGHT_OFF_GROUND)
+ {
+ setControlFlags(AGENT_CONTROL_UP_NEG);
+ }
+ }
+ }
+
+ //  calculate delta rotation to target heading
+ F32 delta_target_heading = angle_between(mFrameAgent.getAtAxis(), mAutoPilotTargetFacing);
+
+
+ if (xy_distance > slow_distance && /*yaw*/ yaw2 < (F_PI / 10.f))
+ {
+ // walking/flying fast
+ setControlFlags(AGENT_CONTROL_FAST_AT | AGENT_CONTROL_AT_POS);
+
+ }
+ else if (mAutoPilotTargetDist > mAutoPilotStopDistance)
+ {
+
+ if (at * direction_nav > 0.9f)
+ {
+ setControlFlags(AGENT_CONTROL_AT_POS);
+ }
+ else if (at * direction_nav < -0.9f)
+ {
+ setControlFlags(AGENT_CONTROL_AT_NEG);
+ }
+ }
+
+ // check to see if we need to keep rotating to target orientation
+ if (mAutoPilotTargetDist < mAutoPilotStopDistance)
+ {
+ setControlFlags(AGENT_CONTROL_STOP);
+ if(!mAutoPilotUseRotation || (delta_target_heading < mAutoPilotRotationThreshold))
+ {
+// std::cout<<"Agent reaches to target."<<std::endl;
+ stopAutoPilot();
```

```
+ }
+ }
+ }
+}
+
+//*****************************************************************
+
+//-------------------------------------------------------------------------
 // propagate()
 //-------------------------------------------------------------------------
 void LLAgent::propagate(const F32 dt)
```

E.2 DTW Code

```
%Evaluation function
%input : 11 sequences of paths
%output: Cost table of DTW cost
%        Average, standard diviation of each path's cost
function [ Ctable, average, stdiv] = Evaluation(S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11)

Ctable=zeros(11,11);

Paths={S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11};
for i=1:11;
    for j=1:11;
        [D,Ctable(i,j)]=DP_Algorithm(Paths{i},Paths{j});
    end
end

average=mean(Ctable,1);
stdiv=std(Ctable,0,1);
end



%Use dynamic programming to find a min-cost path through matrix M.
%Return state sequence in p,q
%2003-03-15 dpwe@ee.columbia.edu
% Copyright (c) 2003 Dan Ellis <dpwe@ee.columbia.edu>
% released under GPL - see file COPYRIGHT

%Edited by Takatoshi Ono

function [D,cost] = DP_Algorithm(Path1,Path2)
%param: 2 path sequences (should be 2D vectors)
%Compute distance map and cost of similarity

[p1X,p1Y]=size(Path1);
[p2X,p2Y]=size(Path2);

M=zeros(p1X,p2X);

for i=1:p1X;
    for j=1:p2X;
        M(i,j)=sqrt(abs(Path1(i,1)-Path2(j,1)).^2+abs(Path1(i,2)-Path2(j,2)).^2);
```

```
    end
end

[r,c] = size(M);

% costs
D = zeros(r+1, c+1);
D(1,:) = NaN;
D(:,1) = NaN;
D(1,1) = 0;
D(2:(r+1), 2:(c+1)) = M;

% traceback
phi = zeros(r,c);

for i = 1:r;
  for j = 1:c;
    [dmax, tb] = min([D(i, j), D(i, j+1), D(i+1, j)]);
    D(i+1,j+1) = D(i+1,j+1)+dmax;
    phi(i,j) = tb;
  end
end

% Traceback from top left
i = r;
j = c;
p = i;
q = j;
while i > 1 && j > 1
  tb = phi(i,j);
  if (tb == 1)
    i = i-1;
    j = j-1;
  elseif (tb == 2)
    i = i-1;
  elseif (tb == 3)
    j = j-1;
  else
    error;
  end
  p = [i,p];
  q = [j,q];
end

% Strip off the edges of the D matrix before returning
D = D(2:(r+1),2:(c+1));
cost=D(size(D,1),size(D,2));

end
```

Bibliography

[1] J. Berg, M. Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. *IEEE International Conference on Robotics and Automation*, pages 1928–1935, 2008.

[2] J. Canny. *The complexity of robot motion planning*. The MIT Press, 1988.

[3] M. Cavazza, S. Bandi, and I. Palmer. "situated ai" in video games: Integrating nlp, path planning and 3d animation. *Proceedings of AAAI Spring Symposium on Arti.cial Intelligence and Computer Games, Stanford University, Stanford, CA*, 1999.

[4] F. Feutey. Simulating the collision avoidance behavior of pedestrians. *Master's Thesis, University Tokyo*, 2000.

[5] P. Fiorini and Z. Shillert. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7), 1998.

[6] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4, 1997.

[7] J.-S. Gutmann, M. Fukuchi, and M. Fujita. Real-time path planning for humanoid robot navigation. *The 19th International Joint Conference on Artificial Intelligence*, 2005.

[8] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3), 2002.

[9] S.-W. Hsu and T.-Y. Li. Third-person interactive control of humanoid with real-time motion planning algorithm. *IEEE International Conference on Intelligent Robots and Systems*, pages 4845–4850, 2006.

[10] L. Jaillet and T. Sim?on. A prm-based motion planner for dynamically changing environments. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[11] Y. Kanayama and B. I. Hartman. Smooth local path planning for autonomous vehicles. *IEEE International Conference on Robotics and Automation*, 3:1265 – 1270, 1989.

[12] L. Kavraki and J. C. Latombe. Randomized preprocessing of configuration space for fast path planning. *IEEE International Conference on Robotics and Automation*, 1994.

[13] B. Kluge and E. Prassler. Reflective navigation: individual behaviors and group behaviors. *IEEE International Conference on Robotics and Automation*, 2007.

[14] E. Kokkinara. Modelling attention in autonomous virtual characters. *University College London Masters Thesis*, 2010.

[15] M. C. Lee and M. G. Park. Artificial potential field based path planning for mobile robots using a virtual obstacle concept. *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 2, 2003.

[16] W.-Y. Lee, A.-D. Yang, T.-C. Hung, and J.-S. Guo. Path planning strategy design under the experience of mobile robot navigation. *Computational Intelligence in Robotics and Automation*, pages 316–321, 2009.

[17] T.-Y. Li and H.-K. Ting. An intelligent user interface with motion planning for 3d navigation. *IEEE Virtual Reality 2000 Conference*, 2000.

[18] D. Michael and Y. Chrysanthou. Automatic high level avatar guidance based on affordance of movement. *Eurographics 2003*, 2003.

[19] M. Müller and T. Röder. Motion templates for automatic classification and retrieval of motion capture data. *Eurographics / ACM SIGGRAPH Symposium on Computer Animation*, 48(3), 2006.

[20] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3), 1970.

[21] M. H. Overmars. Path planning for games. *International Game Design and Technology Workshop*, 2005.

[22] S. Petti and T. Fraichard. Safe motion planning in dynamic environments. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[23] L. Rabiner and B. H. Juang. *Fundamentals of Speech Recognition*. Signal processing. Prentice Hall, 1993.

[24] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura. The intelligent asimo: system overview and integration. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3, 2002.

[25] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING*, 26(1), 1978.

[26] B. Salomon, M. Garber, M. C. Lin, and D. Manocha. Interactive navigation in complex environments using path planning. *Proceedings of the 2003 Symposium on Interactive 3D Graphics*, pages 41–51, 2003.

[27] J. Snape, J. V. D. Berg, S. J. Guy, and D. Manocha. Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[28] K. E. Stevens, S. Kruck, J. Hawkins, and S. C. Baker. Second life as a tool for engaging students across the curriculum. In *Design and Implementation of Educational Games: Theoretical and Practical Perspectives*. Information Science Reference, 2010.

[29] A. Sud, E. Andersen, S. Curtis, M. C. Lin, and D. Manocha. Real-time path planning in dynamic virtual environments using multiagent navigation graphs. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):526–538, 2008.

[30] M. Sung, L. Kovar, and M. Gleicher. Fast and accurate goal-directed motion synthesis for crowds. *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, 2005.

[31] C. J. Taylor and D. J. Kriegman. Vision-based motion planning and exploration algorithms for mobile robots. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, 14(3), 1998.

[32] J. Wiecha, R. Heyden, E. Sternthal, and M. Merialdi. Learning in a virtual world: Experience with using second life for medical education. *Journal of Medical Internet Research*, 12(1), 2010.