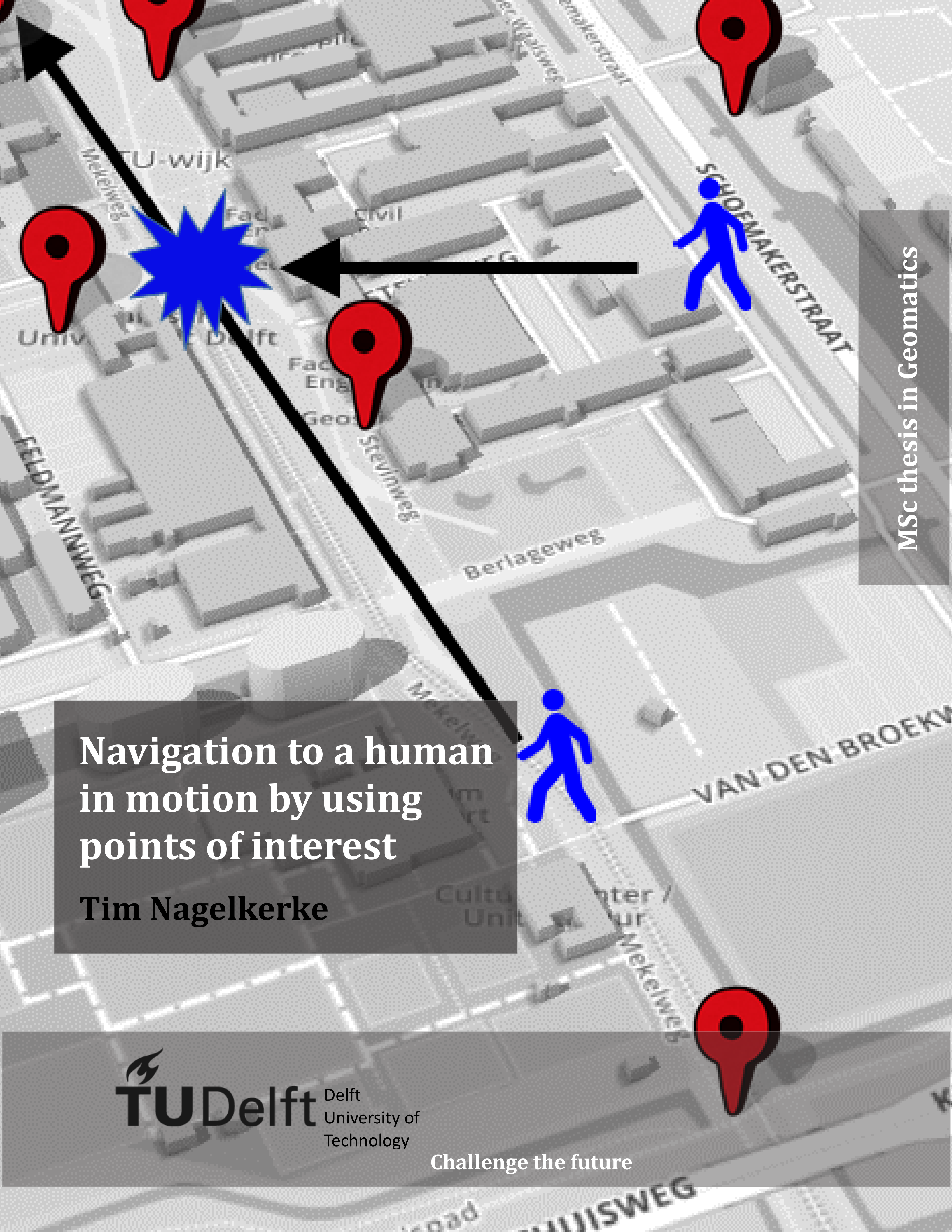

MSc thesis in Geomatics

Navigation to a human in motion by using points of interest

Tim Nagelkerke

Delft University of Technology

Challenge the future

On cover:
Abstract result of the SEA* method implemented in the outdoor implementation using real GPS data, scenario 2, start position 3. The map used for visualization is obtained from osmbuildings.org.

Navigation to a human in motion by using points of interest

Master of Science Thesis

By

Tim Nagelkerke

Student number: 4083792

E-mail: t.nagelkerke@student.tudelft.nl

June, 2016

In partial fulfillment of the requirements for the degree of

Master of Science

in Geomatics

at the Delft University of Technology

to be defended publicly on Monday June 27, 2016 at 10:45 AM.

First mentor:	*Zlatanova, Dr.ing. S.*
Second mentor:	*Diakite, A.A.*
Co-reader:	*Quak, Drs.C.W.*
Delegate of the board of examiners:	*Burg, Ir. L.P.J. van den*

Faculty of Architecture and the Built Environment - Delft University of Technology

Delft University of Technology

Abstract

Navigating to a human target is underexposed in the current literature. However, there are cases where it is necessary to get to the moving target as fast as possible. This thesis supports navigation of a person to another person in motion that they lost or need. In this way, children, elderly, family, coworkers and friends could be found more quickly.

This research thesis proposes the Semantically Enriched A * (SEA*) method to use semantics, in the form of points of interest, to determine the prediction of the target and uses this prediction to approach the target. Overall the SEA* method uses the positive components of the iterative A* algorithm, semantics and the direction of the target to predict where the target is going to, to successfully reach the target. Points of interest, landmarks, are critical points to check where a person is moving towards. These static locations are promising for navigating to a person in motion. Estimating the predicting location of the target is recommended by first limiting down the points of interest by the approaching points of interest by the target and then using the point of interest that is the closest to the target. This process gives a good prediction of the target in both implementations.

The proposed SEA* method is tested both in an indoor environment, as in an outdoor environment. The SEA* method shows promising results in both the simulated indoor environment, represented by a 2D square regular grid, as in the outdoor environment, represented by a road network, using real GPS data. In this thesis the SEA* method is compared to the iterative A* approach and shows promising improvements.

This thesis provides a framework that could be implemented to always find a person in motion or find them faster by using shortcuts to get to this person. A variety of different spatial models could implement the SEA* algorithm as long as the spatial model supports the A* algorithm, is able to translate the positions of the user and the target to the spatial model and supports adding points of interest to the model.

Finally this research discusses further implementations of this method. Future research must provide answers to the questions if this method also works in a real-time case or if it works using a 3D spatial model to support 3D indoor navigation.

Table of Contents

List of figures

List of tables

List of algorithms

List of abbreviations

GPS:	Global Positioning System
MTS:	Moving Target Search
TIN:	Triangular Irregular Network
DT:	Delaunay Triangulation
CDT:	Constrained Delaunay Triangulation
SEA*:	Semantically Enriched A*
POI:	Point of Interest
2D:	Two Dimensional
3D:	Three Dimensional
API:	Application Programming Interface
FRA*:	Fringe-Retrieving A*

1. Introduction

1.1 Motivation

In the last decade the navigation market has been highly developed and continues to develop. There is a huge boost in the number of navigation devices and applications used worldwide in the last few years (European GNSS Agency, 2013). The navigation market exists out of several applications that support pathfinding and guidance, from car navigation systems (MarketsandMarkets, 2011, TomTom, 2016 & Garmin, 2016), which navigate a user to a preferred destination, to indoor applications (MarketsandMarkets, 2014, Infsoft, 2016 & Fallah et. al., 2013), which calculates a route to find the right room for a meeting. Due to the development of mobile phones and the ability to use internet services, many mobile applications have been developed to ease the user.

The current applications that support navigation all focus on the navigation from the user to a static destination. In this way pathfinding, the process of finding a path to the desired end point, calculates a path from the current position of the user to the position of the static destination. More advanced applications also update the current position of the user and recalculate the path to the static destination, so that if the user makes an error, the user still knows how to continue their way to the destination. Navigation to a static destination is fine if the user wants to find a specific building or room, which are all static places that always have the same geographical position. Although, there is an emerging need for applications that provide a path towards a dynamic destination (Zlatanova et. al., 2013), no applications are found to navigate a human to a dynamic destination. This dynamic destination could for example be:

- A child that is lost and has to be found
- Finding your partner in a big city or big building
- A family member with Alzheimer that needs help to find the way back home
- A co-worker during an emergency in a crowded place
- A scarce tool in a hospital that is expensive and is used frequently in different operating rooms

A dynamic destination could therefore be defined as: a target, which is someone or something, in motion that needs to be found or is needed.

By navigation to a static destination the path is calculated once and then the user can continue this path until it reaches the end point. In this way, the path can be either the shortest path, the fastest path or a user preferred path. By navigation to a dynamic destination it is not possible to determine the shortest, fastest or user preferred path, because the target is moving and the destination will change over time. Navigating to a previous location of a dynamic end point has no guarantees to eventually meet the dynamic target. Figure 1.1 shows the difference between navigation to a static end point and navigating to a dynamic end point. On the left, a user (A) is navigating to for example a coffee machine (B), where the shortest and fastest path is visualized by the arrow. The coffee machine is static and therefore the user will reach its destination. On the right, a user (A) is navigating to a person in motion (B). The user navigates to the person, but because the person is moving the user might not meet this person. In this case, the user and the person never meet each other when there are no multiple measurements of the position of the target.

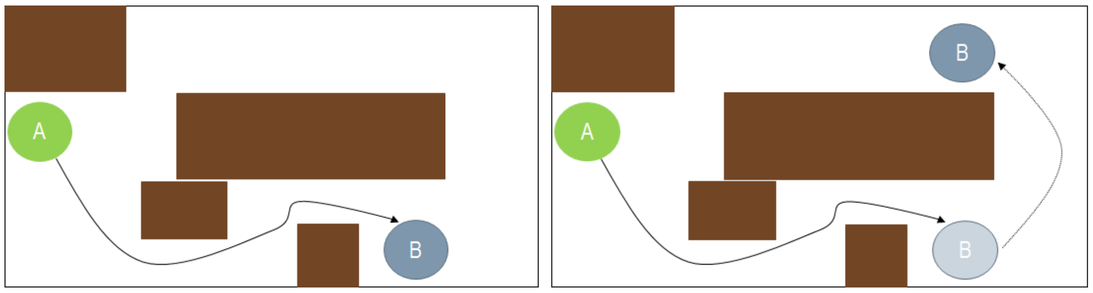

Figure 1.1, left: navigation to a static end point, right: navigation to a dynamic end point

Current buildings and city centers have signs to support navigation to a location. Current signs support navigation to static end points, but cannot solve navigation to dynamic targets unless you promised the target to meet at a specific place. However, verbal communication is not always possible between two people. Therefore, there is a lack of possibilities to navigate to a person in motion. Especially for a user that wants to navigate to a person that cannot verbally contact the user, like small children or people with dementia.

Current navigation systems consist out of a few techniques:

- Positioning of the user
- Positioning of the target
- Spatial model of a building or a city
- Algorithm to calculate the path

Both the position of the user and the target, the starting point and the end point, has to be known to calculate a route. Based on the methods to obtain an accurate position of the user, there is made a distinction between outdoor and indoor navigation. Outdoor navigation is based on the Global Positioning System (GPS) that gives a high accurate position of the user. However, GPS does not have accurate measurements to obtain the position in an indoor environment, due to the weak signal strength. Therefore other positioning systems like RFID, Bluetooth, WLAN, UWB, Cellular-Based, IR and Ultrasonic (Adalja, 2013) are used to determine the accurate positioning of the user within an indoor environment. Nowadays systems lets the user enter an end destination that is static, but this could also be a person in motion that is being tracked by one of these systems.

There also has to be a spatial model that represents the terrain that can be used for navigating purposes. Algfoor et al. (2015) give a summary of possible spatial models used in the fields of gaming and robotics. They make a distinction between regular grids, irregular grids and hierarchical techniques to represent the terrain into a spatial model.

When there is an accurate location of both the target and the user and there is a spatial model, there must be a way to calculate the path from the starting point to the end point. In case of navigating to a static destination the A* algorithm is widely used and examined in the navigation domain (Fallah et. al., 2013). This algorithm uses a heuristic value to speed up the process to find the fastest path possible from the starting point to the end point. The A* algorithm always find a path, when there is a path available between two points. Also the A* algorithm is faster than other path finding algorithms, like the Dijkstra algorithm. Therefore this research focusses on using the A* algorithm. A more in depth overview of the A* algorithm is described in chapter 2.

In other domains, like robotics and the gaming industry, navigating to a moving target is widely explored and implemented under the term Moving Target Search (MTS). MTS is introduced by Ishida and Korf (1991) and is a dynamic path planning problem where the user is trying to catch a moving target with minimum movement cost. This problem is applicable to computer games, where characters have to follow or attack each other, or to robotics where robots has to steer to a moving target (Nussbaum and Ÿořukču, 2015). Sun, Yeoh and Koenig (2010) offer two approaches to deal with MTS, either an offline or an online approach.

The offline approach uses all possible parameters to find the best strategy for the user, but because it calculates all the possibilities, it is not efficient for a large environment. Also the position of the moving target is not known in advance, wherefore this approach is not suitable for navigation to a moving person.

The online approach only uses the information that is available at the time and uses this to navigate to the moving target. While the user gets more information if the target is moving, it will update the information that is available and calculate a new path. Two algorithms that use the online approach are real-time search algorithms (Ishida and Korf, 1991) and incremental search algorithms (Sun, Yeoh and Koenig, 2009).

The real-time search algorithms only deal with a small amount of time before the target is moved and the calculation has to be adjusted. The disadvantage is that when the user moves with only the available information, the user might move in a suboptimal way. Undeger and Polat (2007) developed a real-time edge follow alternative reduction method, which uses perceptual information. Undeger and Polat their method has improved in contrast to previous real-time search algorithms like real-time A* and real-time A* with n-look-ahead depth (Korf, 1990). Sun, Yeoh and Koenig (2010) mention that real-time searches are really memory intensive and therefore do not scale well with larger environments. Therefore, these algorithms are not applicable for large buildings or outdoor environments.

Incremental search algorithms on the other hand first calculates a path and then starts to move towards the moving target (Sun, Yeoh and Koenig, 2010). Incremental search algorithms are optimal to calculate a path in a large environment and are all based on the iterative use of the A* algorithm, which calculates the path every ‘n’ seconds and therefore updates the path to let the user reach the target. The incremental A* algorithm is the leading algorithm for navigating a user towards a moving target, in games and the robotics domain. Therefore this research will examine these kind of algorithms that could be used to navigate a person towards a moving target.

However, these concepts could be used to navigate a person within a building to a moving target, navigating persons differs from navigating robots or game characters. Persons do behave differently, while robots and game characters have to be programmed. Therefore, the end position of a gaming character can be known in advance. When there is a person walking around in a building, the end destination of this person is unknown. The incremental A* algorithm has the limitation that the user might follow the target, instead of finding a faster way towards the target by using a shortcut. Also when a target is still moving, the target will be followed and never reached.

Also different from robots is that people can avoid obstacles in real-time. Therefore, the map representation has not to be very precise and small or dynamic obstacles have not to be modelled. Humans also differ from game characters in the way that they walk toward something within a building, for example a door, a room, the coffee machine or a staircase.

These differences can be solved by using the semantics of a building. Semantics are a way to give meaning to a model, so that the model knows where the important features for a certain person are. A person in motion is always walking toward an interesting place for him or her. These places can be defined as points of interest. Semantics of the building could be used to check where a dynamic target is heading to and to navigate towards its prediction. The points of interest for a specific dynamic target are then used as prediction points of this dynamic target. In this way, semantics can solve the issue that the user can take shortcuts to get to a target faster and that it will always be able to reach the target.

1.2 Research objective

Incremental A* algorithms are widely used in the gaming and robotics domains, but are never implemented in applications to navigate a person towards another person in motion. This is because of the differences in these domains, where humans interact differently than robots or gaming characters. Adding semantics might be the factor to bridge the different domains and also make applications for persons to navigate to a moving target. Semantics could be used to determine where the person is moving to, which is the prediction of the target. Then the user can navigate towards this prediction. The focus of this research is investigating possible improvements for using the iterative A* algorithm for human navigation with using semantics. The general aim of this research is to develop a conceptual framework that supports navigation to a human in motion based on the iterative A* algorithm and semantics. To support these findings the proposed method will be implemented and tested. The main research question for this thesis is as follows:

Which defined objects could be used to estimate the predicting location of a moving person to support navigation to a person in motion?

To answer this main research question, the research question is divided into four sub questions:

1. What are the current limitations of the A* algorithm for navigating to a dynamic target?
2. What semantics are important for navigation to a moving target?

3. What are the improvements and limitations of the proposed method over using the iterative A* algorithm?

1.3 Research scope

The core of this research is to check whether the iterative A* algorithm supports navigation from a person to another person in motion and improve this algorithm by using semantics. This research will be tested by implementing this method and test possible scenarios. Due to the fact that there is no indoor positioning system available for this research, it is not possible to create an application for indoor navigation. However, there will be built an application, based on real building data, where the behavior of the target is simulated. Also this research will be tested within an outdoor environment, using real GPS data.

The scope of this research focusses on:

- One person to one person navigation, there will be no examination of navigating multiple people.
- A static indoor and outdoor environment, where small and dynamic obstacles are not mapped. Only the static obstacles are mapped. This means that people automatically avoid small and dynamic obstacles to get to their destination.
- Only 2D navigation is examined, 3D is out of the scope of this research. Extending the framework to a 3D environment will be discussed in chapter 6.
- No indoor positioning techniques are used. The positioning of the user and the target is simulated. In theory RFID-tags or a Wi-Fi system can be used to get an accurate position of the target and the user within the indoor environment. The proposed method, will address how this component could be implemented.

1.4 Research contributions

This thesis research has a research contribution that can be divided in a scientific contribution and in a societal contribution.

1.4.1 Scientific contribution

This thesis studies the possibility to use and improve the iterative A* algorithm for human to human navigation, instead of navigation a human to a static end point. To bridge the gap between the domains where the iterative A* algorithm is implemented and to implement this algorithm in human to human navigation, semantics will be added. To my findings there is no such application that navigates a human to another human in motion. Also existing systems, like signs in buildings or on the streets are not sufficient to deal with the problem to navigate to a moving target without having to verbally contact the target. Therefore the developed method will give more insight in navigating to a moving person. This research will investigate the limitation of the iterative use of the A* algorithm.

Determine possible points as the prediction of the target and navigate not directly to the target, but to the prediction of this target is a new concept. The indoor application is based on real simulated positioning data of the user and the target. For the outdoor application real GPS data is used to test the conceptual framework. Therefore these implementations give good insight in how the conceptual framework works. Where the effectiveness of the navigation could be determined.

Eventually the data that is necessary to navigate to a moving target is examined and used to support both applications. The results of this thesis research cannot only be used for navigating one person to one person, but eventually adjusted to navigate multiple people at the same time or navigating to robots or animals.

1.4.2 Societal contribution

The main societal contribution of this thesis is to support navigation of a person to another person that they lost or need. In this way children, elderly, family, coworkers and friends can be found more quickly. Especially navigating to persons that cannot verbally communicate with the user, or animals is a problem that can be solved by using this conceptual framework. These persons or animals need an RFID-tag or some kind of positioning system to locate them so that the user could find them quickly and spend no unnecessary time by following the target until it stops moving.

Examples are that parents could lose their child in a playground or in a big warehouse like IKEA, where the parents have a smartphone and the child has a RFID-tag in his/her jacket. The parents could then always find their child as fast as possible. Another example is that animals in a zoo can escape and these have to be found very quickly to ensure there is no danger to the visitors. However, defining points of interest is harder than by human navigation.

The emergency services could use this technique to navigate as quickly as possible to each other in a dangerous situation. This could be for example a big fire for firefighters or a terroristic attack for policemen. Also ambulance personnel can quickly navigate when they have to apply first aid to someone that is leaving a building in a dangerous situation. Points of interest could then for example be emergency exits.

1.5 Thesis outline

This thesis consists out of six chapters, which are structured as follows:

Chapter 1	introduces the problem statement, the research objectives, the scope of the research and the research contributions.
Chapter 2	presents the literature findings concerning the iterative/incremental A* algorithm, existing spatial models and the use of semantics for navigation. This chapter gives answers to the first two sub questions of this research, and theoretical background information.
Chapter 3	gives a detailed description of the developed conceptual framework method. This framework is using characteristics of the incremental A* algorithm and semantics to navigate from one human to another human in motion.
Chapter 4	presents the results of the implementation within an indoor environment of the conceptual framework explained in chapter three. This implementation uses simulation for the positioning of the user and the target.
Chapter 5	shows the results of the outdoor implementation of the conceptual framework explained in chapter three. This implementation uses real GPS data to test the framework.
Chapter 6	concludes this thesis research, answers the main research question and addresses future research topics.

The code used for both the indoor and outdoor implementation of the SEA method is made public on the 27th of June and is available at: https://github.com/TimNagelkerke/thesis-implementation.*

2. Theoretical background

This chapter discusses the theoretical background that is used for this thesis to answer the research question. This chapter is mainly a literature study that discusses approaches to solve navigation from one person to another person. This chapter is subdivided in three different subsections. Mainly pathfinding consists out of a way to represent the terrain and a pathfinding algorithm. Section 2.1 describes the possible spatial models that are used to represent the terrain to obtain a navigable map. Section 2.2 discusses the use of a semantic model for navigation purposes in the context of navigating to a human in motion. Finally section 2.3 explains the working and the limitations of the A* algorithm and the incremental A* algorithms for dynamic human navigation.

2.1 Spatial models

To navigate a user towards a target there must be knowledge about the terrain. The terrain where the user wants to navigate on has to be represented by a graph. A graph is a structure that is used to represent relations between different objects. These objects are called nodes, which could be defined as "a certain place with a fixed geographical position". A node for example could be a room in a building or an intersection in a road network. The geometric model represents the location of the nodes. The relations between these nodes are called edges. These edges are defined as "a connection from one node to another node". An edge for example could be a hallway in a building or a street from the first intersection to the next intersection. The topological model represents the relations between these nodes, with each a specific movement cost to travel from one node to the other node. A graph is therefore a collection of nodes and edges (Trudeau, 1994 and Chartrand, 1984). A graph could therefore be defined as:

$$G = (N, E)$$

where N is the set of nodes and E is the set of edges.

Most spatial models do not only consist out of a geometrical component and a topological component, but they also support a semantic component. This semantic model gives meaning to the nodes and edges. For example, what specific room is meant by a specific node or what does that edge in a road network represent, a highway or a normal road. Section 2.2 goes into more detail.

In a recent study Algfoor et. al. (2015) created an overview of the current spatial models. They divide the spatial models into two categories: Grids and Hierarchical techniques. The grids are subdivided to regular grids and irregular grids. A further decomposition is shown in table 2.1.

Table 2.1, decomposition of spatial models (Algfoor et. al., 2015).

Grids		**Hierarchical techniques**
Regular grids	Irregular grids	
2D square grid	Visibility graphs	Probabilistic road maps
2D hexagonal grid	Meshes navigation	Quadtrees
2D triangular grid	Waypoints	Rapidly exploring random trees
3D cubic grid		

According to Ma et. al. (2011) a grid is a composition of vertices/nodes that are connected by the edges that represent these vertices/nodes. Navigation on a grid is possible by the properties of the grid, the neighboring relations of the grid and the distance from one point to another point. There are different kinds of grids, which are regular grids and irregular grids. The following sections will elaborate on the different kind of spatial models.

2.1.1 Regular grids

Regular grids are the most used ones to create a graph representation (Alfgoor, 2015).There are a lot of regular grids used for navigation purposes (Girard et. al. 2011, Lee, 2004, Kuffner, 1998, Bandi & Thalmann, 1998 & Li et. al. 2010). These regular grids consist only out of one shape that repeats itself. Examples of these regular grids are 2D square, hexagonal and triangular grids or a 3D cubic grid. A 2D square grid is a grid consisting out of squares, where the other regular grids consist out of hexagons, triangles and cubes. Each shape is representing a node that is used for navigation. It is important that grids represent the terrain in a correct way. Therefore obstacles in the grid have to be representative. A grid is never able to give an exact representation of the real situation. There is always some sort of overrepresentation of the obstacles in a grid. This overrepresentation depends on the used shape, see figures 2.1, 2.2 and 2.3. In these figures it is visible that a 2D square grid has a bigger overrepresentation (the blue area) than the 2D hexagonal and the 2D triangular grids. Although the 2D square grid is represented by less shapes and therefore the computation on such a grid is faster. To lower the overrepresentation of the obstacles the size of the shapes could be adjusted, for example the size of the square could be smaller and therefore there would be less overrepresentation of the obstacles. The size of the shapes in the grid is an important factor to represent the terrain.

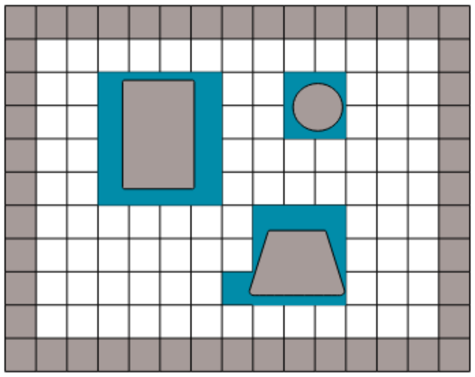
Figure 2.1, 2D square grid overrepresentation of three obstacles (Algfoor, 2015).

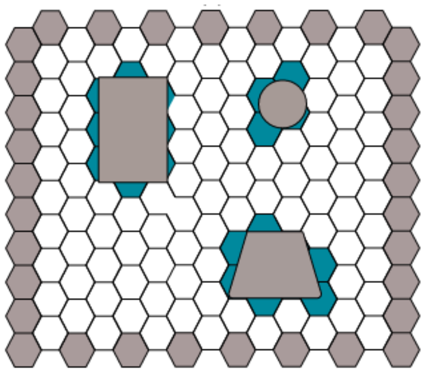
Figure 2.2, 2D hexagonal grid overrepresentation of three obstacles (Algfoor, 2015).

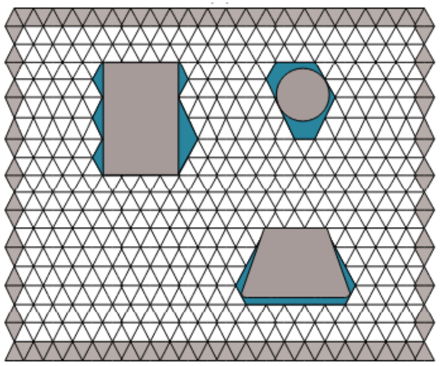
Figure 2.3, 2D triangular grid overrepresentation of three obstacles (Algfoor, 2015).

The size of the shapes in the grid is determined by several factors. The first factor is the scale of the map, the grid size has to represent the obstacles and the free space in a way that the grid size does not block a possible passage, or makes an overrepresentation of the walls in a way a room has less doors than in the real situation. The second factor is the natural movement of the person. A person that is standing still is occupying between the 1 m² and 0.5 m² of free space. The last factor is the speed on which a person moves, which is determined by the movement of the users of the navigation system.

Obstacles in a grid could be represented in a few ways (Patel, 2006). This could be done by a Boolean grid, where the value 0 stands for navigable and the value 1 stands for obstacle. Only the spaces that have a value of 0 will be used for the navigation. Another way to represent obstacles is to remove the obstacles from the grid, where the grid only represents the navigable space. Or the obstacles could have an infinite movement cost to get to a particular obstacle. In this way these obstacles are never used in the navigation, but the path planning algorithm that is used has to calculate all these possibilities and therefore will be less efficient.

To create a grid, a map of the actual situation has to be crossed by a grid. When there is an obstacle in the map the whole shape will be marked as an obstacle and it is not possible to cross this shape during the navigation, see figures 2.1, 2.2 and 2.3. The process to convert a vector map to a raster map is called rasterization. The outcome of this process is a graph with the set of nodes and set of edges. The grid size is the most important parameter for this process.

Now that the geometrical component of the grid, the set of nodes, is discussed it is possible to explain the relations between these shapes, the topological component. There are different possibilities of adjacency within a grid. First for the regular square grid a four adjacency (figure 2.4) or an eight adjacency (figure 2.5) could be used to. Where B are the adjacent squares of the square A.

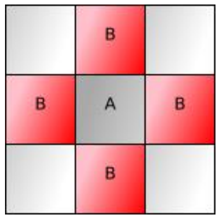

Figure 2.4, four adjacency within a 2D square grid.

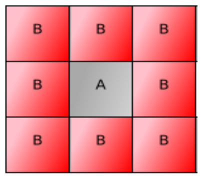

Figure 2.5, eight adjacency within a 2D square grid.

This is important to check what possibilities the user has to navigate in a certain direction. A four adjacency square grid only let people navigate in the horizontal and vertical direction, while with an eight adjacency square grid people are also allowed to move diagonal (Patel, 2006).

If in figure 2.4 the position of the tile in the middle is defined as (X, Y), then the adjacent neighbors are the following:

X + 1, Y	X, Y + 1
X – 1, Y	X, Y – 1

The natural movement of a person also makes it possible to move diagonally in an open space. In the case of an eight adjacency (figure 2.5) the neighbors are defined as follows:

X + 1, Y	X + 1, Y + 1
X – 1, Y	X + 1, Y – 1
X, Y + 1	X – 1, Y + 1
X, Y – 1	X – 1, Y – 1

Moving on a hexagonal grid has another adjacency function. In this case the neighbors of the center hexagon are all the six neighbors, see figure 2.6. The adjacent neighbors are defined as follows, where (X, Y) is the centering hexagon:

X – 1, Y + 1 (North West)	X, Y + 1 (North East)
X – 1, Y (West)	X + 1, Y (East)
X – 1, Y – 1 (South West)	X, Y – 1 (South East)

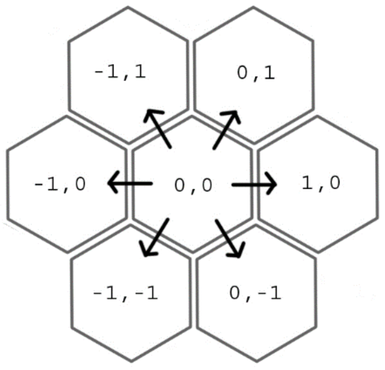

Figure 2.6, adjacency in a 2D hexagonal grid.

The adjacency on a triangular grid is more complicated due to the fact that there are twelve neighboring triangles when movement in every direction is allowed (figure 2.7). A normal three adjacency is insufficient for representing the behavior of navigating in any direction. The triangles with number one represent triangles that share the same edge, triangles with number two represent triangles that share an edge with number one and a corner with the centering triangle and triangles with number three only share an edge with the centering triangle (Nagy, 2003).

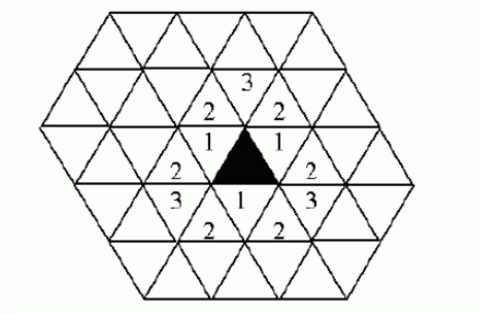

Figure 2.7, twelve adjacency in a 2D triangular grid (Nagy, 2003).

Nagy (2003) discusses the twelve adjacency in a triangular grid, using three axes, x, y and z. According to these rules each neighboring triangle could be expressed using these axes, where the x-axis is the first term, the y-axis is the second term and the z-axis is the third term. This is visualized in figure 2.8. When the triangle is in the opposite direction of the center triangle (0, 0, 0) respectively to the direction of the axis then the value will decrease by one in that specific direction. When the triangle is in the same direction as the center triangle respectively to the direction of the axis, then the value will increase by one in that specific direction. In this way it is possible to determine all the twelve adjacent triangles of the center triangle.

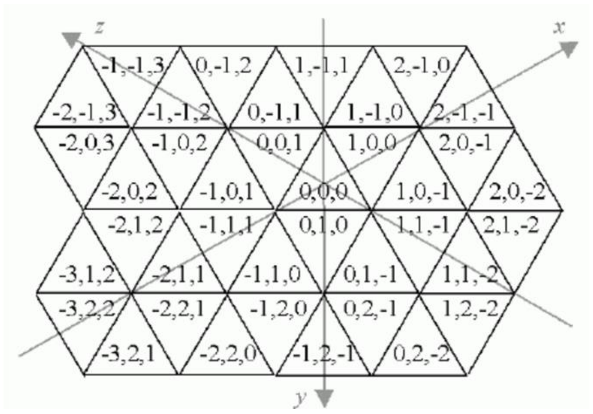

Figure 2.8, determining the adjacent triangles in a 2D triangular grid (Nagy, 2003).

A 3D cubic grid is using voxels to represent the 3D environment. Although this research focusses on the 2D terrain, a cubic grid could represent the indoor environment in another way than the 2D grids. The space is filled with cubes and whenever a cube intersects with an obstacle, this space is non-navigable. However, it is harder to determine the exact topological model. Research is needed to check in what situations a human could navigate to each of the neighboring cubes.

To navigate on a regular grid, the grid must be translated to a navigation graph. This graph is constructed by using the center points of the shapes (square, hexagon, and triangle) of the grid as nodes and the edges are the connections between those nodes, if two shapes are its neighbors.

2.1.2 Irregular grids

Also irregular grids, a network representation is used to subdivide the terrain for human navigation (Brown et. al., 2013, Boguslawski & Gold, 2009, Liu & Zlatanova, 2012 & Lorenz et. al. 2006). Not the whole terrain is subdivided in equal shapes, but in irregular shapes, like polygons. Each polygon is then presented by a node, where the adjacency between the nodes is retrieved from the neighboring relationship.

A visibility graph (figure 2.9) is a set of nodes, where each node represents the corners of the obstacles, and the edges represent a visible connection between the nodes. For each set of two nodes is checked whether the direct line between the two nodes intersects an obstacle. If the direct line between the nodes intersects an obstacle, these nodes have no visibility edge. Else, they have a visibility edge. This visibility graph always finds the shortest path, if the distances of each edge are known, using a path finding algorithm. Using a visibility graph could give insight in the situation that the user and the target could see each other. However, each time the position of the user or the target changes the visibility graph has to be recomputed by adding the positions of the persons (Berg et. al. 2008). These nodes and edges could be directly used for navigation.

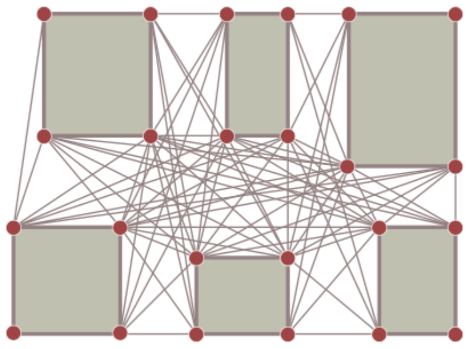

Figure 2.9, the visibility graph (Patel, 2006).

Mesh navigation (figure 2.10) does not represent the obstacles as polygons, as with the visibility graph, but represents the navigable space as polygons. Patel (2006) mentions four possible approaches to navigate on a mesh. First of all (figure 2.10, top left) it is possible to use the center of each irregular shape as a node, second (figure 2.10, top middle) it is possible to use the middle of each edge as the navigable nodes, the third approach (figure 2.10, top right) is using the corners of the irregular shapes as navigable nodes and the last approach (figure 2.10, bottom) is combining the second and third approach. The hybrid approach is the most promising, but makes the model also more complex. The second approach is therefore the best alternative. All approaches however give a sub-optimal path (figure 2.10, yellow paths) compared to using a visibility graph (figure 2.10, pink path).

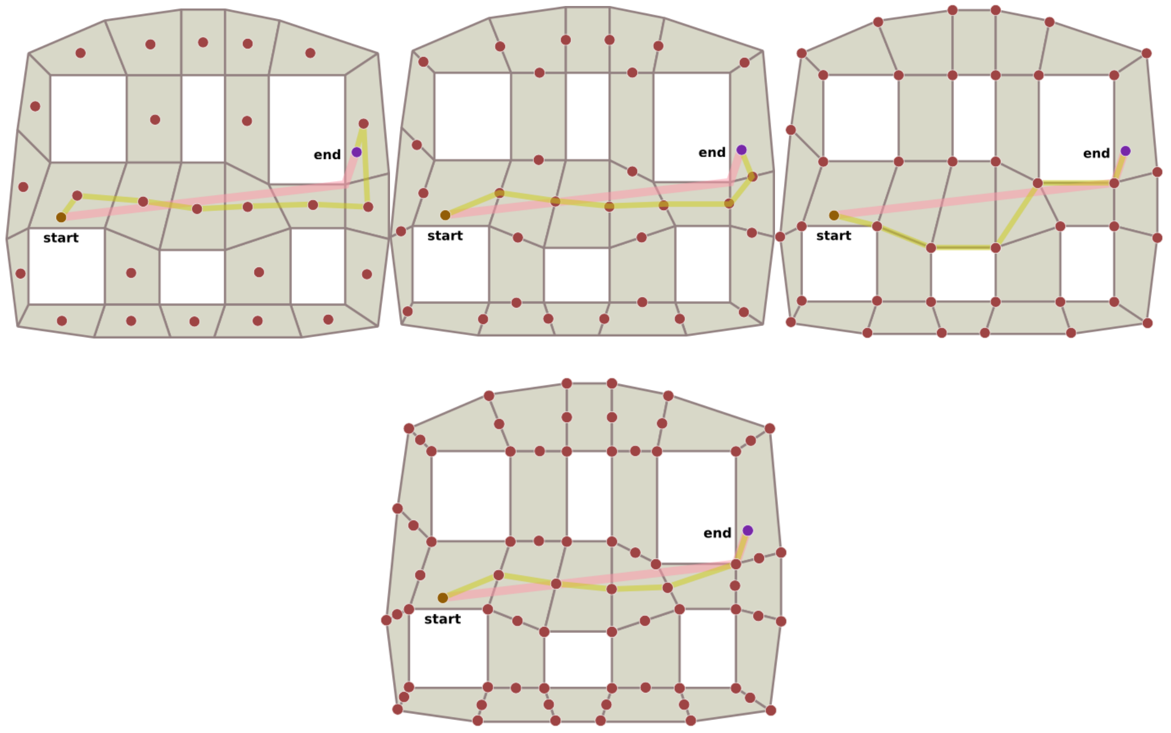

Figure 2.10, four approaches for mesh navigation (Patel, 2006)

Not only polygon shapes, but also triangular shapes could be used to represent a terrain. The Triangular Irregular Network (TIN) is a vector-based topological data model that represents the terrain. A TIN subdivides the terrain in irregular spaced triangles. A TIN is the result of the Delaunay Triangulation (DT) of the Voronoi diagram representing the surface, figure 2.11. However, when navigating in an environment with obstacles a DT is not sufficient, due to the fact that this approach does not take into account the obstacles. Therefore a Constrained Delaunay Triangulation (CDT) is developed which a result is depicted in figure 2.12 (Fleischmann, 1999). Here the original edges of an obstacle are used as input to obtain a navigable graph. Therefore the nodes and edges of a CDT could be used for navigation purposes. The Euclidean distances between these nodes will refer to the movement costs in the path planning algorithm.

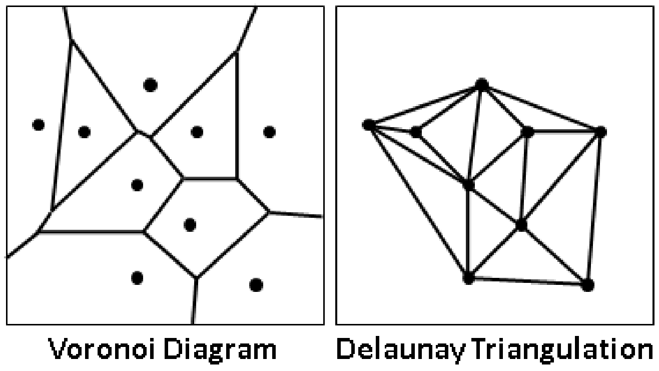

Figure 2.11, relation Voronoi Diagram and Delaunay Triangulation, retrieved from http://www.csie.ntnu.edu.tw/~u91029/Triangulation.html.

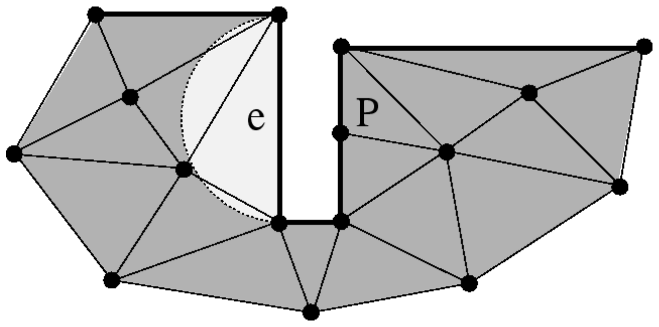

Figure 2.12, Constrained Delaunay Triangulation (Fleischmann, 1999).

Waypoint navigation is a spatial model where people navigate based on waypoints, where waypoints are physical locations. These waypoints often refer to landmarks, which are critical points in human navigation. These waypoints could be used to indirectly navigate to a certain end destination. The user visits first the specific points, the waypoints, and then continues its path to the end destination. In this way it is possible to navigate with a certain strategy. The waypoints are representing the nodes, where the relation between the landmarks are representing the edges to obtain a navigable graph. The distances between the nodes have to be calculated to get the edge cost for the path finding algorithm. Figure 2.13 shows how waypoints could be used to navigate from the start point, to the end point. When the model contains more waypoints the path towards the goal will be approaching the fastest path. Figure 2.13 shows the difference between using waypoints and a navigation mesh. Both approaches obtain a sub-optimal path compared to a visibility graph.

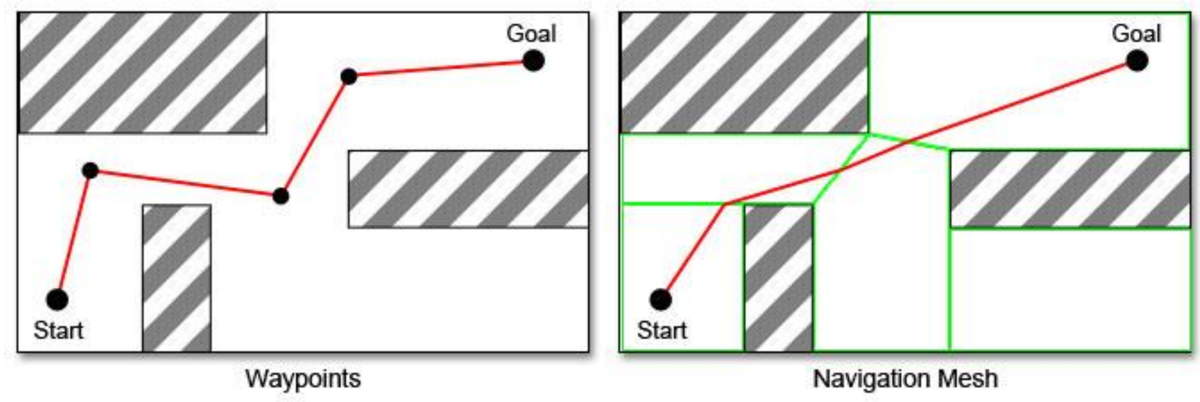

Figure 2.13, waypoints and navigation mesh, retrieved from https://udn.epicgames.com/Three/AIAndNavigationHome.html.

Irregular grids include less nodes as the regular grids and therefore the path planning algorithm is faster. Although, the positioning of both the user and the target cannot be described as detailed as the regular grids. There are two methods to user the positioning of the user and the location. The first approach is that the location of the user and the target are snapped to the closest node in the model. This gives a bad result, when there are not that many nodes present in the model. When there are more nodes present in the model, the positions of the user and the target are approached more exact. The second approach is to use the positions of the user and the target as nodes in the model. Then these nodes should be added to the irregular grid, where the edges must be retrieved dealing with the criteria of the model. This is inefficient, because the edges in the model must be recalculated, each time the position of the target or user has changed. Therefore the first approach is recommended. Points of interest could be added, by adding nodes to each model. This could be done when the spatial model is created. Both regular and irregular grids require considerable memory space within large environments (Algfoor, 2015).

2.1.3 Hierarchical techniques

Grids use a lot of memory and are therefore not always efficient. Hierarchical techniques deal with this problem and divide the terrain into multiple levels. Therefore it is possible to have a higher level of abstraction, which represents the terrain in less detail, combined with lower levels of abstraction, which represent the terrain into more detail. Algfoor (2015) defines three possible hierarchical techniques: Probabilistic Road Maps, Quadtrees and Rapidly Exploring Random Trees.

Probabilistic Road Maps and Rapidly Exploring Random Trees are developed to use in autonomous robotic path planning. Therefore the robot takes samples and uses these to initialize its search. Humans behave totally different from robots, because they can find a path with the help of navigation instructions. Therefore the model used for navigation has not to be very precise.

For human navigation the Quadtree could be used, which is a homogenous hierarchy with an arbitrary number of levels. The Quadtree data structure is a hierarchical technique that subdivides the space in four equally grid cells until each grid cell is either an obstacle or an empty space (Samet, 1988). The Octree is the 3D equivalent of the Quadtree (Samet, 1988). In case there is a memory overload and a grid representation is not suitable, the Quadtree is a good solution. The Quadtree data structure is suitable to represent static environments. The Octree is suitable to identify the navigable space within a 3D environment. This navigable space could be used for navigation, where the center of each square represents a node and the edges are retrieved by the neighboring squares.

2.1.4 Discussion

There are several ways to represent a 2D flat terrain, which are regular grids, irregular grids or using a Quadtree. Each of these implementations has its own applications and complexity. Algfoor (2015) & Ma (2011) both note that a regular grid gives the best representation for a plane surface, while irregular grids are better to represent three dimensional surfaces. Also a regular grid gives very detailed locations of both the user and the target, while irregular grids do not exactly describe the location, but give a more abstract location. In the case that there must be an accurate description of the positions of the user and the target a 2D square grid is recommended. This spatial model is thoroughly tested and supports path planning algorithms and semantic modelling. The geometrical and topological model is easier to construct than by the other spatial models. When efficiency is more important than the precise location, an irregular grid is recommended. A visibility graph returns the shortest path and is therefore the most optimal representation. When a grid becomes inefficient, a Quadtree must be used to represent the terrain.

2.2 Semantic modelling

People that navigate in a building mainly have a clear end destination. For example a person that goes to his work, walks to his room, gets a coffee in the breaks at the coffee machine and eventually leaves the office at one of the exits. Adding semantics to a model is actually a technique to give meaning to a model. In this way it is for example possible to check where the room of the target is that the user needs, or where the coffee machines are within a building. In this way it is possible to make a distinction between different objects in a model. The characteristics of the target could support the user to navigate to a moving target.

Worboys (2011) notes that semantic models represent the entities in an indoor space with their properties and mutual relationships. Also Worboys makes an important difference between different kinds of semantics, namely the difference between the static and the dynamic environment. The static environment is everything that cannot be moved, for example the doors, walls, rooms, windows and the floors (figure 2.14). The dynamic environment is everything that can move on its own or be moved, like furniture, people and equipment. Landmarks play a really important role in navigation applications and location-based services (Worboys, 2011). According to the Cambridge dictionary the definition of a landmark is: "*a building or place that is easily recognized, especially one that you can use to judge where you are".* So landmarks could be defined as critical points in a building where a person makes decisions where to navigate to or to where a person wants to navigate to.

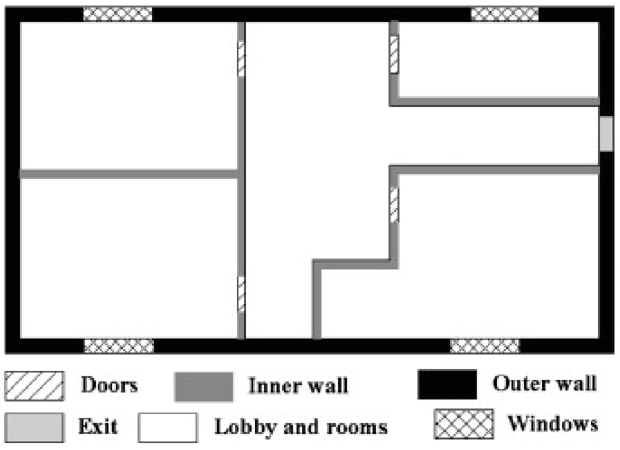

Figure 2.14, a building using semantics (Li et. al., 2010)

It is important that semantics are supported by the used spatial model. A regular grid is suitable for using semantics. Each shape on the grid represents a certain location. The geometrical model remains the same, only there is extra information about what each shape represents. If for example a coffee machine is on this specific location, this shape will get a value that represents that there is a coffee machine. In this case it is possible to store all the coffee machines in a building and retrieve their locations. Adding semantics to an irregular grid is more difficult. Most landmarks and important points are modelled as nodes, however this does not always have to be the case. Especially using a visibility graph, it is hard to add extra points to the model. In a real-time application the visibility graph has to be recomputed each time there is a new node involved. Also by using waypoints or a mesh navigation this is the case, but these representations mostly include the landmarks. Then for each node semantics could be added.

The end destinations where a person moves to are defined as points of interest. Goetz and Zipf (2011) also use points of interest to represent important fixed locations for a specific person. These points of interest each have its own fixed position, are user specific and can change over time. Each person has other points of interest within a building, for example a visitor of a company has other points of interest than the personnel working for that company. Also it is important to note that some points of interest are more likely to be visited at a specific time than at another time. For example the canteen is more likely to be visited around noon than at other times. Also when a building is closing, the exits of the building are becoming points of interest for the people that are inside the building.

The static semantics according to Worboys (2011) always have a fixed location and do not change over time, assuming that there are no renovations. The indoor location of a target can be inaccurate, however indoor position techniques are improving. Navigating to a static end point is more accurate, because this is a precise location. Therefore the static entities are important future destinations for the moving target. These could be used to navigate to the prediction of the target.

2.3 Pathfinding algorithms

Now that the spatial models are discussed, it is important to know what approaches there are to navigate to a moving target. There are a lot of algorithms developed that solve the calculation of a path from the beginning point to the end point, which are called path planning algorithms. These algorithms each fulfill the user needs on its own way. Examples of these algorithms are algorithms that calculate the shortest, the fastest or a least risks path for the user to the destination. Most current applications use the A* algorithm to find the fastest route from the beginning point to the end point (Fallah et. al., 2013). The A* algorithm is the fastest algorithm that always finds a path when there is a path available between two points. Also it finds always the fastest path between two points, given the edge costs in the model. Therefore the A* algorithm is recommended over the Dijkstra algorithm or a greedy Best First Search algorithm. The A* algorithm is suitable to navigate to a static end point, but is not sufficient to navigate to a moving target. Therefore the incremental A* algorithms are developed, which is based on the iterative use of the A* algorithm. This algorithm is used a lot in robotics and the gaming industry to solve the continuous calculation of a path to a moving target. As discussed in the introduction of this thesis research the focus lies on the incremental A* algorithms. To understand this algorithm first the basic A* algorithm will be explained. Second, the incremental A* algorithms that are available will be examined and the limitations of these algorithms will be explained.

2.2.1 The A* algorithm

The A* algorithm is first described by Hart et. al. (1968) which is an extension of the algorithm of Dijkstra (Dijkstra, 1959). The A* algorithm is a best-first search algorithm that searches all possibilities to find the shortest path from the beginning point to the end point. To speed up the search of all possibilities a heuristic function is used. This function determines which next step will be the most promising and chooses this step as the next iteration. This process continues until the end destination is reached.

Before the algorithm can be explained it is important to explain the movement cost function that is used. The movement cost function sums the cost to get to the node and the prediction of the cost it will take to get to the end point, which is the heuristic function. The movement cost function is as follows (Hart et. al., 1968):

$$F(n) = g(n) + h(n)$$

where g(n) = the movement cost/distance from the starting node to the current node

where h(n) = the cost/distance of the current node until the end node, the heuristic function

where F(n) = total costs of the current node

The heuristic function (h(n)) is the important estimate to control the A* search behavior. There are several cases where the A* algorithm behaves optimal or far from optimal (Patel, 2006 & Hart et. al., 1968):

- If h(n) is zero, this algorithm will act as the Dijkstra algorithm.
- If h(n) is lower than the cost of moving from the node to the goal, the algorithm will find a shortest path more quickly than the Dijkstra algorithm.

- If h(n) is equal the cost of moving from the node to the goal, only the best path will expand and therefore the search is very quick. This however is nearly impossible to achieve.
- If h (n) is higher than the cost of moving from the node to the goal, the algorithm is fast but uncertain to find the destination.
- If h (n) is very high in comparison with g(n), only the heuristic function determines the movement costs. The algorithm will act as a greedy Best First Search.

These cases are really different from each other and therefore it is important to use a correct heuristic function. The heuristic function determines the speed and the accuracy of the path planning algorithm. Because in real-time applications the speed of the system must satisfy the user needs, the algorithm must be fast. However the user also needs to navigate to the end destination, and therefore must always find the end destination. In this case the best heuristic that should be used is that h(n) is lower than the cost of moving from the node to the goal. In this way the algorithm is faster than the Dijkstra algorithm and always finds the path from the starting point to the end point, see figure 2.15.

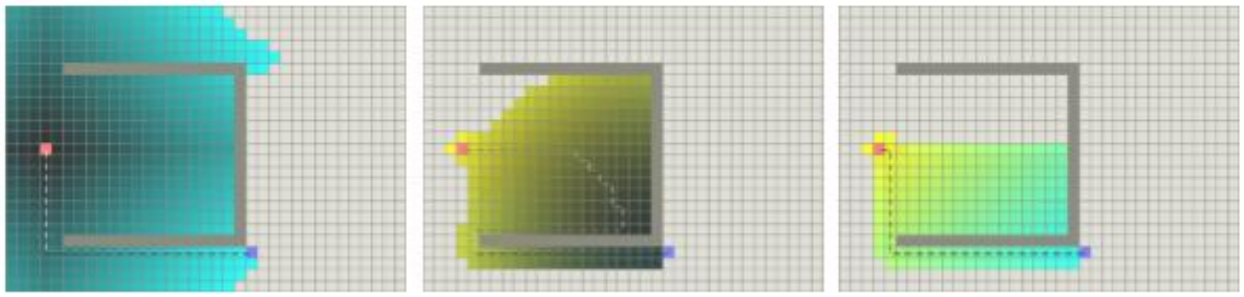

Figure 2.15, the search space of the Dijkstra algorithm (left), a greedy Best First Search (middle) and the A algorithm (right)*

Now that the behavior of the heuristic function is discussed, several different heuristic functions will be examined. The focus of this thesis lies on a regular square grid, and therefore only grid based heuristic functions will be discussed.

The heuristic function that has to be chosen depends on the movement that is allowed. Patel (2006) discusses that the Manhattan distance is best for 4 directional movement on a grid and that the Chebyshev or the octile distance is best for 8 directional movement on a grid. Also it is possible to use the Euclidean distance as the heuristic.

Manhattan distance

The Manhattan distance is according to Black (2006): "*The distance between two points measured along the axes at right angles".* The Manhattan distance in a plane is calculated by the following formula:

$$D_{\text{Manhattan}} = |x1 - x2| + |y1 - y2|$$

Where the starting point is x1, y1

Where the ending point is x2, y2

The Manhattan heuristic is not taking into account diagonal movement and therefore not suitable for human navigation on a grid. The heuristic is an overestimate of the actual distance (Rabin and Sturtevant, 2013). This heuristic is suitable for navigating when only horizontal and vertical movement is allowed (Patel, 2006).

Chebyshev distance

The Chebyshev distance is the distance that is known as the chessboard distance (Kresse and Danko, 2012 & Math.NET, 2016). The Chebyshev distance in a plane is calculated by the following formula:

$$D_{\text{Chebyshev}} = \max(|x1 - x2|, |y1 - y2|)$$

Where the starting point is x1, y1

Where the ending point is x2, y2

The Chebyshev heuristic takes diagonal movement into account. In this case the diagonal movement cost is the same as the vertical or horizontal movement cost. The Chebyshev distance does not align with the human movement, due to the fact that the movement cost in each direction is equal.

Octile distance

The Octile distance is the distance that allows diagonal movement as a movement cost of $\sqrt{2}$ (Rabin and Sturtevant, 2013). The Octile distance in a plane is calculated by the following formula:

$$D_{\text{Octile}} = \max(|x1 - x2|, |y1 - y2|) + (\sqrt{2} - 1) * \min(|x1 - x2|, |y1 - y2|)$$

Where the starting point is x1, y1

Where the ending point is x2, y2

The Octile heuristic is according to Rabin and Sturtevant (2013) closest to the natural movement of a person. Therefore this heuristic is the most accurate to use on a grid. Also Patel (2006) argues that the Octile distance is the best heuristic for movement in eight directions.

Euclidean distance

The Euclidean distance is according to Black (2004): "*The straight line distance between two points*". The Euclidean distance in a plane is calculated by the following formula:

$$D_{\text{Euclidean}} = \sqrt{}((x1 - x2)^2 + (y1 - y2)^2)$$

Where the starting point is x1, y1

Where the ending point is x2, y2

According to Patel (2006) and Rabin and Sturtevant (2013) the Euclidean distance is an underestimate of the actual distance on a grid. This heuristic assumes that movement in any direction is allowed, which is not possible on a grid. The Euclidean distance is best for applications where movement in any direction is allowed. Therefore the Euclidean distance is the best heuristic to use with an irregular grid.

Now that the cost function and the heuristic functions are discussed, the A* algorithm will be explained. First of all the algorithm needs to initialize an open list, a closed list and a list that keeps track of what the parent of which node is. The closed list is the list of the nodes that already are evaluated. The open list is the set of nodes that are the current search space. The starting node is initially the starting point from where the search begins and this node is added to the open list. Then when the open list is not empty the algorithm sorts the open list and takes the node with the least movement cost (F(n)). This node then has to be removed from the open list. If this node is the goal then the path is the reverse of the list that keeps track of what the parent of which node is that starts with the current node. If the current node is not the end goal then the current node is added to the closed list. Then the neighboring nodes are evaluated for potential candidates. Then for each neighboring node that is not already visited and is not in the open list the neighboring node is added to this open list. If the neighboring node is not in the closed list then the movement cost (F(n)) is recalculated to be sure that this is not a more promising path. When the movement cost is more promising this node's movement cost has to be updated. If the current neighboring node is the most promising, this node is added to the list that keeps track of what the parent of which node is. During this process the total movement cost of each node is stored and this value is used to aim the search of the A* algorithm. The pseudocode of the A* algorithm is algorithm 2.1: The A* algorithm (based on Eranki, 2002, Patel, 2006 & Hart et. al., 1968).

Algorithm 2.1: the A* algorithm

Input:

- **Starting point**
- **End point**

Output:

- **Path from the starting point to the end point**

```
1.  Function A*(start, end)
2.  initialize
3.  Closed list = {}
4.  Open list = {start} where the f(n) = 0
5.  cameFrom = empty map
6.
7.  While open list is not empty:
8.     Sort the open list based on the lowest f(n)
9.     Current node = first of the open list
10.
11.    If current = end:
12.       Return path(cameFrom, end)
13.
14.    Remove the current node from the open list
15.    Add the current node to the closed list
16.    Determine all the neighbors of the current node
17.    For each neighbor:
18.       If neighbor in closed list:
19.          Continue
20.       G(n) neighbor = current G(n) + distance current and neighbor
21.       If neighbor not in  open list:
22.          Add neighbor to open list
23.       Else if G(n) >= G(n) neighbor
24.          Continue
25.
26.       Add current to cameFrom[neighbor]
27.       F(n) neighbor = G(n) neighbor + H(n)
28.  Return no possible path
```

In the case that the open list is empty before the end node is reached there is no path from the starting point to the end point. In the case the goal is reached, another algorithm is used that actually reconstructs the path. This algorithm reverses the list that keeps track of what the parent of which node is and gives the path from the beginning node to the end node. This is algorithm 2.2: Path creation.

Algorithm 2.2: Path creation

Input:

cameFrom mapping
End point

Output:

Path from the starting point to the end point

```
1. Function Path(cameFrom, end)
2. initialize
3. Path = {end}
4.
5. While end in cameFrom:
6.    end = parent of the end node
7.    Add end to the Path
8. Return Path
```

2.1.2 The incremental A* algorithm

Now that the A* algorithm is explained it is possible to describe the incremental A* algorithms. In general incremental search algorithms combine the heuristic search of the A* algorithm and calculate the path for each unit of time (Koenig et. al., 2004), therefore these algorithms give the same results as the iterative A* algorithm. Incremental search algorithms are mostly based on the A* algorithm. It is possible to calculate the path with the A* algorithm every time when the end destination changes, but because A* can be slow for large environments, this might be insufficient and the user of the system has to wait every time the system recalculates a path. To solve this issue there are a lot of different variations on the A* algorithm. Mainly incremental search uses the information of a previous search to speed up the calculation for the current search, which makes the path finding algorithm faster than calculating the A* from scratch (Deo and Pang, 1984). Incremental search algorithms are especially made to deal with incomplete information, which means that the map of the environment is unknown to the user, or with information that changes over time, which focusses on pathfinding to a moving target (Koenig et. al., 2004). Sun, Yeoh and Koenig (2010) classify two different families of incremental searches.

The first kind uses information from the previous search and updates the heuristic values to focus for the next search. Examples of this first kind are MT-Adaptive A* (Koenig, Likhachev and Sun, 2007) and Generalized Adaptive A* (Sun, Koenig and Yeoh, 2008).

The second type of incremental searches uses the previous search tree as a starting point for the current search tree. This has the advantage over incremental A* that it does not start from scratch. Examples of this second kind are Differential A* (Trovato and Dorst, 2002), D* (Stentz 1994; 1995), D* Lite (Koenig and Likhachev, 2005), Fringe-Retrieving A* (FRA*) (Sun, Koenig and Yeoh, 2009), Generalized FRA* (Sun, Koenig and Yeoh, 2010) and Moving Target D* Lite (Sun, Koenig and Yeoh, 2010). These kind of incremental searches were designed for static environments. Therefore this type of algorithms is best for solving the pathfinding problem in a static indoor or outdoor environment.

Sun, Koenig and Yeoh (2010) proved that FRA* has the smallest runtime in a static environment when navigating to a moving target. Nussbaum and Ÿořukču (2015) invented a new incremental search algorithm called Moving Target Search with Sub-Goal Graphs which has a faster runtime than the previous named incremental search algorithms. Nussbaum and Ÿořukču use the subgraph algorithm of Uras, Koenig and Hernandez (2013). The idea is to simplify the spatial model to a higher level in a preprocessing step, to speed up the search algorithm. This is only possible if the environment is static and no new obstacles appear. Therefore the preprocessing, by making the subgoal graphs, is ideal for a static environment like a building. Although when the algorithm scales to other buildings, preprocessing will be time consuming. Also when buildings are adjusted, because of renovations, all the preprocessing has to be renewed. Another possibility is to use compressed path databases (CPDs) in the field of moving target search as executed by Baier et. al. (2015), which also gives good results but has a lot of preprocessing steps to cope with the path planning problem.

Incremental search algorithms based on the A* algorithm are used to calculate a path as fast as possible to deal with an autonomous robot in real-time or with game characters that all have to follow each other and have to react quick. It is important to note that the A* algorithm in general is sufficient to calculate a path for a user to a static destination. Using the A* algorithm incrementally could be sufficient to calculate the path each time the target is moving and therefore the user has to be redirected to the target. Especially when the path has not to be calculated every time the target is moving, but only if the moving target is making an important decision. Using the A* algorithm incrementally would be sufficient for this research. This makes the implementation far less complex and there are less steps involved in preprocessing the data to compute very fast paths.

Incremental search algorithms have a few drawbacks in finding a path from the user to the target. Because the incremental search algorithms are based on the A* algorithm it is always used to calculate the shortest path from the start point to the end point. In this way the algorithm does not use any information about the target. For example, the movement vector of the target, the trajectory of the target or other characteristics are not used to navigate to the target. Because only the path is calculated from the user to the target, no possible future shortcuts are taken into account. The behavior of the incremental search will lead to following the target, instead of making smart decisions based on the characteristics of the target. This behavior of following is depicted in figure 2.16. In the top left, the user A navigates to the target B while B is moving north. Then the path will be calculated from A to B. The user A will follow this path, while B moves north. The result of the movement according to this path is shown in the top right. Now the path from A to B has to be recalculated, because the user did not reach the target. This next movement step is showed in the bottom left and the result of this next step is shown in the bottom right. In this case the user A ‘follows’ the target B. In an open space, incremental search algorithms work fine, because it will always use the shortest/fastest path. However, when there are obstacles these algorithms can work sub optimal.

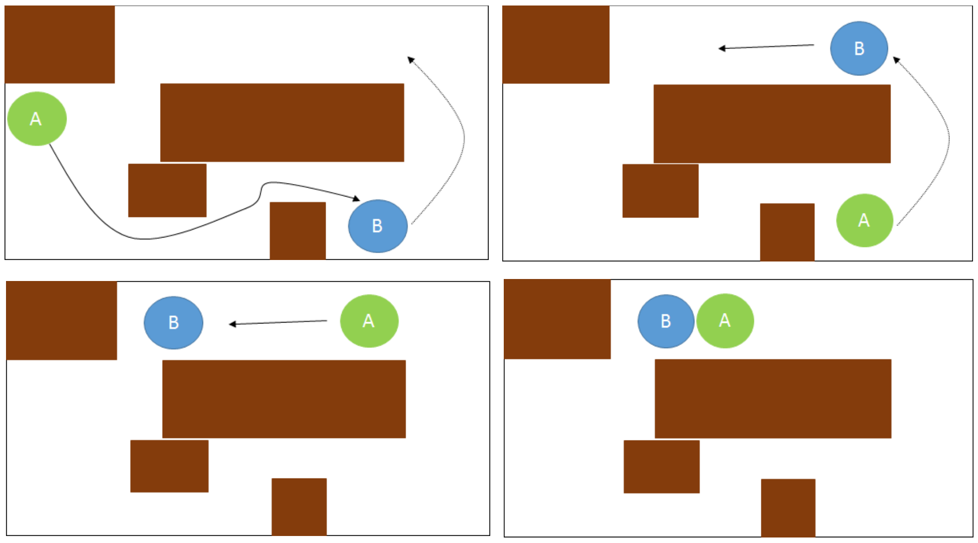

Figure 2.16, Incremental A search results in following the target*

Another drawback is that if the target is moving all the time and the target is moving away from the user, the target will never be reached. This is especially the case when the target is moving faster than the user. If the target is moving slower than the user then the user will be able to finally navigate to the target.

In order to meet the target and to reach the moving target faster it is desired to make use of the characteristics of the target. Therefore not the sub-optimal behavior of following the target as shown in figure 2.16 is preferred, but taking into account the movement of the target (figure 2.17).

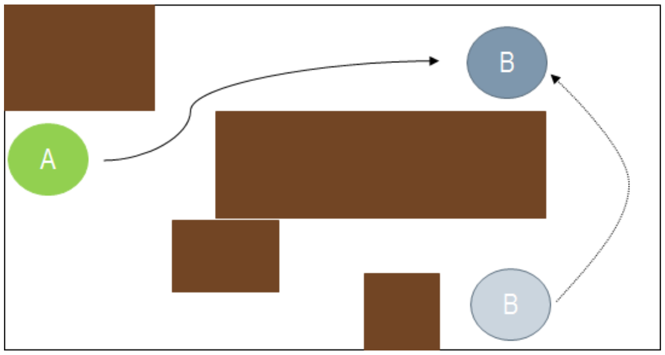

Figure 2.17, desired behavior for the incremental A search*

The movement of the target could be described by a vector that changes over time. A vector is an element that exists of a magnitude and a direction. The notation of a vector is as follows, where a1 is the movement in the horizontal direction and a2 is the movement in the vertical direction:

$$\vec{a} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$$

The movement in the horizontal direction can be translated to the movement on the x-axis and the movement in the vertical direction can be translated to the movement on the y-axis. When the vector is three dimensional, instead of two dimensional, another parameter is added (a3) that represents the movement in the upwards direction, translated to the z-axis.

3. Conceptual framework

Semantically Enriched A (SEA*)*

The introduction and the theoretical framework clarify that path finding to a moving person is underexposed in the current literature. Developed algorithms are suitable for navigation to a static destination, a dynamic robot or game character, but are insufficient to obtain a 'smart' path from the user to the person he needs. In this context the term 'smart' refers to the idea of making use of possible shortcuts in contrast to only following the target, which is used in the current state of incremental search algorithms. Using not only the current position of the target, but also the other characteristics of the target could support this way of smart pathfinding. Humans in motion are more likely to visit points of interest. Given this notion the user could identify the points of interest for the person he wants to navigate to and navigate to these destinations instead of navigating directly to the person.

The aim of this chapter is to provide a conceptual framework to answer the main research question of this thesis research. This chapter is divided into two parts to provide this conceptual framework.

The first part gives an overview of the design of the conceptual framework. The design will discuss the requirements of the framework, gives the overall system architecture and discusses the used parameters. The requirements are used to specify the behavior of the provided conceptual framework given specific criteria and assumptions. The system architecture discusses the overall methodology at a higher level, discusses the input and output data and the relationships between the different components. The system parameters give the definitions of the used concepts.

The second part discussed each component of the overall methodology in more depth. Each component has its own input and output data and the pseudocodes are provided. The framework consists out of five main components. These components are the spatial model, the points of interest, localization of the user and the target, the prediction of the target and the path planning algorithm from the user to the prediction of the target. This section ends with a detailed system flow.

3.1 Conceptual design

This section gives the overall scheme of the conceptual framework and discusses the design at a higher level.

3.1.1 System requirements

The system requirements frame the concept and are divided into criteria, which are conditions that have to be achieved by the method, and assumptions, which are conditions that are taken for granted or accepted without having the actual proof.

Criteria are used to frame the developed method so that the method is user-friendly and could be implemented in real world cases. Therefore five criteria are defined, which should be met by the conceptual framework. These criteria are then used for validation and discussion of the method. The following criteria are defined:

- *Spatially logical*: The data that is used must represent the terrain correctly. This has not to be very detailed, but it must be possible to navigate in ways it is possible to navigate. In this research the focus lies on navigating people, therefore paths must be obtained that can navigate real persons from the start point to the moving target. This could be defined as the space where a person could fit through. The implementation of this method should therefore take this into account.
- *Scalable*: The implementation of the conceptual framework must be able to use multiple spatial models and multiple datasets. In this way this process is not optimal for only one building in 2D, but can also be used for other buildings in 2D. Also outdoor situations must be covered by this framework. When the conceptual framework is scalable, more people could use the method.
- *User dependent*: The system must be user dependent, because each person has different points of interest. Therefore the user of the system must be able to use its own knowledge to improve the system, by providing a list of points of interest per target. In this way the solution could be optimized for a specific target group.
- *Effective*: The user must always be able to find the target and it must be quicker or at the same speed as a normal iterative A* search. In this way it is always possible to find the person. The method should improve an iterative A* search, because otherwise this method has no added value.

- *Efficient*: The user must also be able to get directions to the target as quick as possible and has not to wait longer than possible. In this way the user should not wait more than a few seconds to get a response for its next move. Otherwise the user navigates to a previous measurement of the target and the method will act as the iterative A* search. Also the user wants to find the target, if the user has to wait a long time, it will not use this framework.

The most important criteria are that the methodology should be spatially logical, effective and efficient. These criteria will be used for the validation process that determines whether the methodology is successful. The other two criteria, that the solution must be scalable and user dependent, are requirements that are preferred but won't be used for the validation. These two criteria will be used in the discussion of this research.

There are a few assumptions that will support this research, which are determined by the theoretical framework in the previous chapter. The following assumptions for the conceptual framework are used:

- A person that is standing still is occupying between the 1.2 m^2 and 0.45 m^2 of free space, which is defined by Hall (1969) as the personal distance. The height where a person might fit through is determined by the lowest standardized door height, which is 201.5 cm (Skantrae, 2016). Although this research focusses only on a two dimensional surface, this surface must fulfill the criteria of spatially logical. Therefore only heights of 201.5 cm or higher and widths of 0.45 meter or more are suitable for human navigation.
- The focus does not lie on indoor positioning. Therefore the assumption is that for an indoor environment the position is measured every few seconds and that the accuracy of these positions are high enough to calculate a movement vector.
- The third dimension is not examined and therefore three dimensional and 2.5 dimensional are used as a two dimensional representation. Stairs to another floor are represented as 'exits'.
- The data of the building is up-to-date. This means that the building has no reconstructions in the time between the creation of the spatial model and the time that the actual path planning is used.
- The persons are walking at a constant speed. Important is that the users both use the same mode of transport, otherwise the time it takes for the user and the target to get to a prediction point are different.

These assumptions are necessary to make decisions in the methodology. These areas require more in depth research and will be discussed in the final chapter of this thesis.

3.1.2 The system architecture: Semantically Enriched A* (SEA*)

Navigating to a moving target has more complexity than navigating to a static destination. The incremental A* algorithm lacks in navigating to a moving target, because the algorithm does no prediction at all about the movement of the target. Because persons behave logical it is possible to predict where they are going and not directly navigate to the moving target but navigate towards this prediction. Therefore the aim of this concept is to provide a path to a user, where the user can navigate to another person by predicting the movement of this other person.

The system is divided into five components: The spatial model, the points of interest, localization, prediction of the target and path planning. Figure 3.1 gives an overview of these five components and the relation between these components. These components together should result in a path where the user should navigate to at each moment in time. Important is the factor that the methodology is used in real time, where the localization must be up-to-date and each time the locations change a new path should be calculated. The localization, the prediction of the target and the path planning are acquired, processed and analyzed in real time. The spatial model is not up-to-date and are therefore acquired and pre-processed before the actual path planning from the user towards the moving target. The next sections will describe each component in more detail.

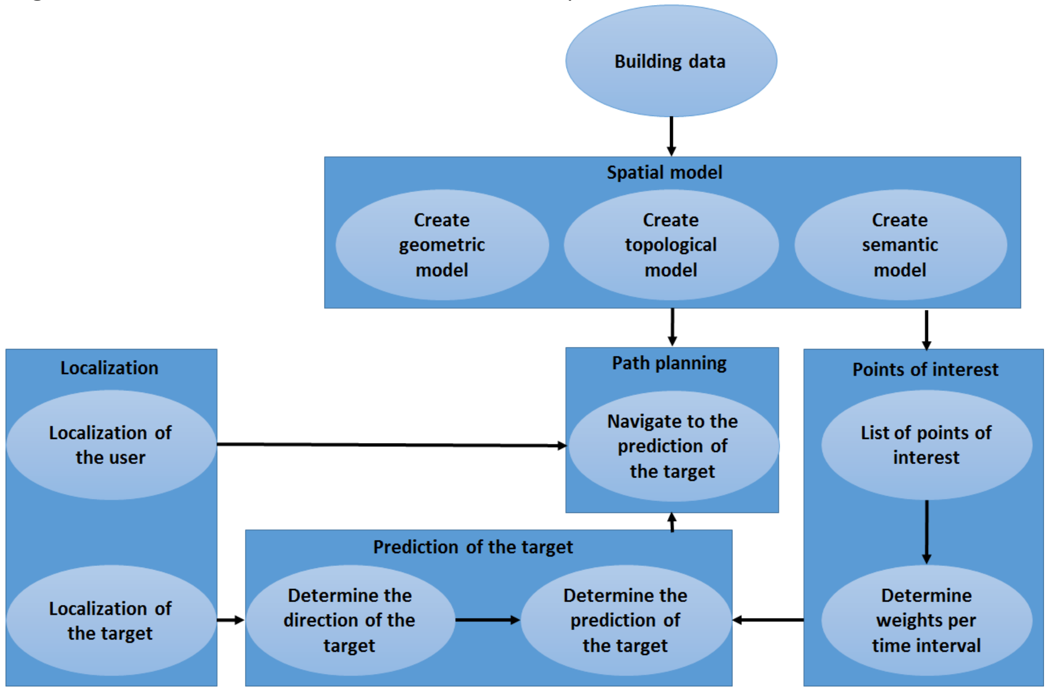
Building data
Spatial model
Create geometric model
Create topological model
Create semantic model
Localization
Localization of the user
Localization of the target
Path planning
Navigate to the prediction of the target
Points of interest
List of points of interest
Determine weights per time interval
Prediction of the target
Determine the direction of the target
Determine the prediction of the target

Figure 3.1, global overview of the conceptual framework, the SEA method*

The conceptual framework is called the Semantically Enriched A* method, or SEA*. The method uses semantics of the building, in the form of points of interest, to predict where the target is moving to and navigates the user to this prediction.

First there must be a spatial model that represents the terrain. Map data is used to represent the actual situation. This map data must be converted to a grid in a way where the requirement spatially logical is achieved. The translation of the map data to the spatial model is described in section 3.2.1. This section explains the usage of a regular grid and the usage of an irregular grid.

The points of interest component uses semantic data to enrich the incremental A* search by providing possible predictions of the target, translated as points of interest. The user of the system has to decide by itself which points of interest are important to take into account, because this person knows which person he or she is following. This step is executed before the actual path finding and is an important pre-processing step, because the whole navigation depends on a good definition of the points of interest for a specific person. Also it must be possible to integrate the semantics into the spatial model, in a way that it is possible to navigate to these points of interest. The semantic model and its detailed implementation is explained in section 3.2.2.

The localization component is very important, but out of scope of this thesis research. Without an accurate localization of the target and the user it is not possible to calculate a path from the user to the target. In this research the possibilities, input and output data is briefly discussed. The implementation however uses a logical simulation of the user and the target to test the methodology in an indoor environment. The framework is also implemented within the outdoor environment using real GPS data. The localization of the user and the target should be up-to-date and the more accurate measurements, the more accurate the path finding will be. This component is briefly described in section 3.2.3.

The prediction of the target component determines to which point of interest the target is most likely to move. It compares the semantic input that the user has given, based on the map data and knowledge about the target, with the actual movement of the target. In this way the most likely point of interest is the prediction of the target. Because there could be multiple points of interest the target is heading to, two approaches are defined. First of all probabilities over time will be used to determine the most likely point of interest the target is navigating to. These probabilities are pre-processed in the points of interest component by the user. The second approach is using the closest point of interest the target is heading to as the prediction of the target. The determination of the prediction of the target is fully explained in section 3.2.4.

The final component is the path planning component. This component uses the position of the user and the prediction of the target as input for the path planning algorithm. The A* algorithm is used iterative to calculate this path every time when the prediction of the target changes. In this way not every movement of the target will acquire an intensive A* search. The spatial model that is created is used as the navigable map. The detailed description of this component is stated in section 3.2.5.

3.1.3 Input data

This section gives an overview of all the data inputs that are necessary for the navigation to a moving target by using semantics. The input data consist out of real-time data and no real-time data, which are listed in table 3.1. These data input each has its own value and the source where the data is derived from is included.

Table 3.1, data input parameters for the SEA method*

Real-time	**Data**	**Value**	**Derived from**
Yes	Up to date position of the user	X,Y Every 'n' second(s)	Positioning system
Yes	Up to date position of the target	X,Y Every 'n' second(s)	Positioning system
No	Spatial model	2D square Boolean grid	Map
Yes	Date Time	Date Hour: minutes	Exact time in the current time zone
No	Points of interest	Place: X,Y Weight (from 0 to 1) per time interval	User defined, statistics

First of all the up to date position of the user is necessary to know, where the user's position is, so this can be used as the starting point for the navigation. This position is the (X, Y) position of the user at the current moment in time. Because it is important to keep the navigation path up-to-date this position of the user must be obtained every 'n' seconds. This user's up-to-date location has to be derived by a positioning system.

Second, the up to date position of the target must be known. This position also has the (X, Y) position of the target at that moment of time. To achieve that the navigation to this person is as accurate as possible, also this position must be in real-time and obtained every ‘n’ seconds. The target’s up-to-date location has to be derived by a positioning system.

The third input data source is the spatial model. The navigable graph is obtained by using a map of the current situation. This map could be a floor plan for a building, or a correct map of an outdoor environment. Both an analog and a digital map could be used, but this has consequences for the pre-processing time. The translation from a map to a useful spatial model is explained in section 3.2.1.

The time and the date are used to determine in what time interval the system is to see which weights are applicable for the points of interest. This time parameter is derived from the current time in the specific time zone. The date is defined as the current date.

The points of interest are determined by the user of the system. The user defines the locations of the points of interest that are interested for the target. In this way the input is a list of points of interest. The user then has to define for each point of interest a weight per time interval. In this way the most important point of interest could be derived from using the current time and the weight at that specific time. Then these locations are used as predictions of the target. The system is highly dependent on the users input data, so the user has to reason every point of interest carefully.

3.1.4 Output data

The output data is the real time path that the user has to take every time the prediction of the target changes. This path should be effective and efficient to guide the user towards the target. This up-to-date path should be correctly visualized or directions towards the target should be given. Otherwise the user has no clue where to go.

3.1.5 System parameters, terms and concepts

This section gives a summary of all the parameters, terms and concepts that are used to make the following sections more clear. Each of these system parameters will be briefly explained.

User: This is the person that uses the system and wants to navigate to the target.

Target: The person that is needed by the user.

Spatial model: *A model containing geometrical, topological and semantic features.*

2D regular square grid: A spatial model where the space is subdivided in tiles, in the horizontal and in the vertical direction, of each the same size.

***Tile**: One square with a minX, maxX, minY and maxY coordinate that represents the reality. The center point of the tile is used as navigation node.*

***Tile size**: The size of each tile in reality. A grid size of 1 meter means that each tile is 1 by 1 meter, 1 m².*

Movement cost: The cost that is used to travel to a certain node.

***Distance**:* One of the in the previous chapter defined distances. The Manhattan, the Chebyshev, the Octile or the Euclidean distance.

A* algorithm: The algorithm presented in the previous chapter, described in Algorithm 1.

Point of interest: A place of interest for the target, defined as a node, with a physical location.

Weight: The weight given as a chance between 0 and 1.

Prediction of the target: The most likely place the target will navigate to based on the points of interest, the weights given by the user and the direction of the target.

Door: A navigable space, where a person fits through that connects two spaces.

Obstacle: A space that in not navigable, or where a person does not fit through.

3.2 Conceptual components

This section discussed each component of the overall methodology in more depth. Each component has its own input and output data. The framework consists out of five main components. These components are the spatial model (section 3.2.1), the points of interest (section 3.2.2), localization of the user and the target (section 3.2.3), the prediction of the target (section 3.2.4) and the path planning algorithm from the user to the predication of the target (section 3.2.5).

3.2.1 The spatial model

The aim of the spatial model component is to provide a navigable map for the user to use to find a path to the target. Map data of an indoor environment or an outdoor environment is used to translate the terrain into a model. Map data that is suitable for representing an indoor environment is a floor plan of the actual situation. A floor plan can both be analog or digital. The difference between these is the time that it takes to translate the map to a navigable model. The obtained model must both support the geometrical features and the topological features. The model created has a significant influence on the other components of the system. The spatial model must support adding semantics and the path finding algorithm must be able to deal with the specific spatial model. Both regular and irregular grids could be used as the spatial model. This component explains how map data must be translated to a spatial model. The following input and output data are acquired:

Input:

- Map data
- Shape size

Output:

- Spatial model

<u>Regular grids:</u>

The process to convert a vector floor plan to a regular grid is called rasterization. In this process there is information loss, because a regular grid cannot precisely define obstacles. A vector map however defines the precise locations of the obstacles. This rasterization process is depicted in figure 3.2.

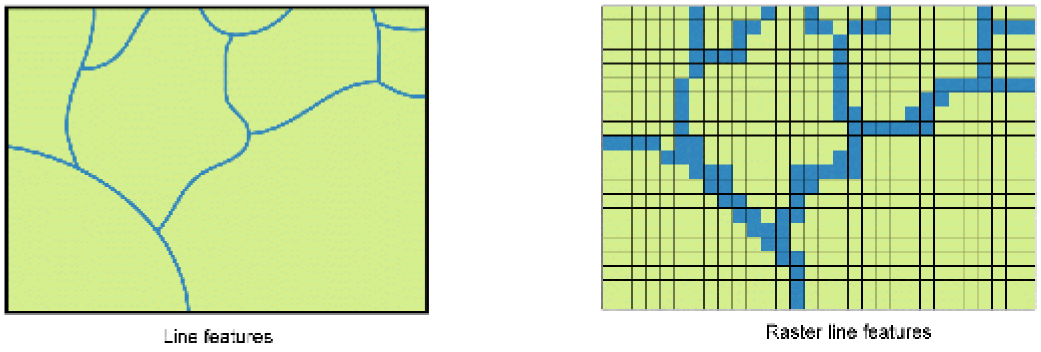

Figure 3.2, rasterization of vector data (ArcGIS, 2016). Left: Vector data & right: raster data.

The floor plan is crossed by a raster and for every obstacle in the floor plan the tile will be marked as obstacle. In figure 3.3 this process is visible and the algorithm is given in algorithm 3.1. The shape size is an important factor to take into account, because this determines the level of detail that is kept in the model. The tile size could be adjusted to the specific needs of the developer of the system. Logically this tile size is for human navigation between the 0.45 and 1.2 meter. Not only squares have to be used, also hexagons or triangles could be used to subdivide the terrain in regular shapes. The same process applies to these shapes.

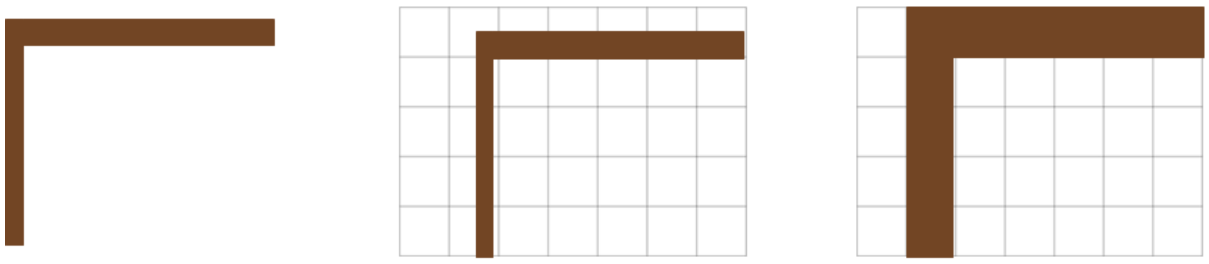

Figure 3.3, rasterization, from vector to a square grid.

Left: floor plan, middle: floor plan crossed by a raster and right: 2D square grid

Algorithm 3.1: Rasterization

```
1.  Procedure Rasterization(map, grid)
2.    Squaregrid
3.    For tile in grid:
4.      If tile is empty:
5.        SquareGrid[tile] = 0 //This space is navigable
6.      Else:
7.        SquareGrid[tile] = 1 //This is an obstacle
8.    Return SquareGrid
```

It is important that the regular grid that is obtained by the rasterization process represents the reality so that people can navigate in a way they also could navigate in the real world. There are two important features that have to be taken into account:

- *Overrepresentation of a passage*: When there is a door in reality, there must be an opening in the model (figure 3.4).
- *Navigable in the model, but not in reality*: If it is not possible to go from one room to another room, then there must be an obstacle between those two rooms (figure 3.5). Here the diagonal movement, that as best represents the natural movement of a person, must be taken into account.

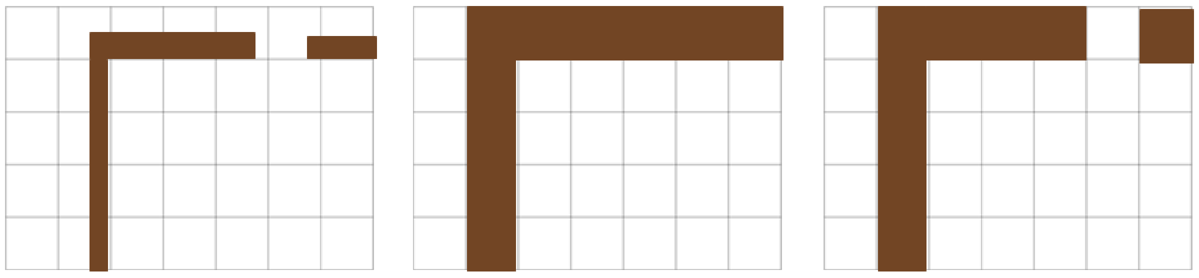

Figure 3.4, Overrepresentation of a passage.

Left: floor plan, middle: incorrect rasterization and right: desired rasterization

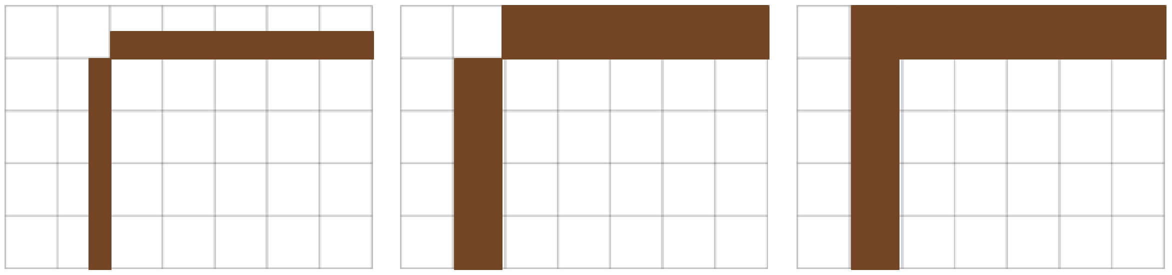

Figure 3.5, navigable in the model, but not in reality.

Left: floor plan, middle: incorrect rasterization and right: desired rasterization

There has to be dealt with these errors, to use a regular grid. Reducing the tile size solves the overrepresentation of passages in the model. However, it gives an error in the other problem, namely navigation in the model, when it is not possible to navigate in reality.

Now that the floor plan is rasterized to a regular grid, it is important to note the topological relations of the tiles in the grid. For the A* search algorithm it is important to detect the neighboring relationship between the tiles. Because diagonal movement is allowed, the neighbors of each tile are determined by the Algorithm 3.2, which uses an eight adjacency. This algorithm uses the relationships of the neighbor from the previous chapter to calculate all possible neighbors. Then the neighbor is only added to the list of neighbors when the neighbor exists and when the neighbor is not an obstacle. In this way the neighbors of each tile can be determined. For other shapes, triangles or hexagons, the adjacency as described in chapter 2 has to be applied instead of the square grid implementation. The process however is the same. The movement cost of each step is defined by the Octile distance in a square grid and a hexagonal grid. For a triangular grid, the Euclidean distance is better.

Algorithm 3.2: Neighbor function

```
1.  Function Neighbor(tile, SquareGrid)
2.    Neighbors
3.    NeighborRelationship = [(1,0), (0, 1), (-1, 0), (0, -1), (-1,-1), (1,1), (-1,1), (1,-1)]
4.    For relationship in NeighborRelationship:
5.      Neighbor = (tile[x] + relationship[0], tile[y] + relationship[1]
6.      If SquareGrid[Neighbor] = 0: //Check if the current tile exists and is not an obstacle
7.        Neighbors ← Neighbor
8.    Return Neighbors
```

Irregular grids:

Not only regular grids could be used as a spatial model, also irregular grids could be used. From the map data a graph must be obtained. A visibility graph, a mesh navigation or waypoints could be used to model a 2D or 3D environment. The nodes of these graphs represent the important locations a person could navigate to. A visibility graph only has an edge when two nodes could 'see' each other. The other irregular grids have edges when the nodes are linked to each other. Each node then has a set of neighboring nodes, which concept is also used by the regular grids. The cost to move from one node to another node must be derived from the distance between the two nodes. For irregular grids this distance is the Euclidean distance between two nodes. A more detailed implementation of using waypoints is given in chapter 5.

Now the spatial model is suitable for navigating purposes. Obstacles and passages in reality are represented in the model and knowledge about possible navigation is obtained.

3.2.2 The points of interest

The points of interest are used to identify possible predictions of the target. From the map data different points of interest could be derived. These points of interest have to be identified before the system is ready. First a list of all possible points of interest of a building must be obtained. Then the user gives according to a specific target weights per time interval for each of these points of interest. The output of this component is a list of locations of the points of interests, where each point of interest has a weight per time interval. The location of a point of interest is represented by a node in the spatial model.

Input:

- Map data

Output:

- Collection of nodes representing the points of interest with weights per time interval

It is impossible to obtain all points of interest automatically from the map data. Therefore the developer of the system should identify the points of interest by itself. A point of interest is specific for each person and therefore it is impossible to give a complete list of points of interest within a building. Information about this target, like its time table, preferences of the target, meetings and attended events could be used to determine the points of interest for a target. Possible points of interest are mainly landmarks and could for example be:

- Rooms (Work space, cafeteria, bathroom, etc.)
- Exits of a building
- Doors within the building
- Stairs/ Elevator
- Coffee Machines
- Information desk

Each point of interest has a static location where it should be able to navigate to. To use these locations the points of interest must have a link to the spatial model. These positions have to be retrieved from the input map data. Each point of interest therefore is represented as a node. Therefore a point of interest must always be defined in the navigable space. The user has to determine the location of each point of interest.

For both a regular grid as an irregular grid, the location of the point of interest is defined by the node representing this landmark. In a regular grid this is the tile representing the object. Each point of interest only could be represented by one node. Therefore if a room contains one door, the point of interest is represented by the node of the location of that door. A room with multiple doors is represented by the center tile in the room. Other objects that consist out of multiple tiles must be represented by the center tile where the person interacts with the object. In this way each point of interest could be translated to one shape in a regular grid:

$$POI_{location} = Tile[0]$$

In an irregular grid the points of interest are mostly defined as the nodes. If there is a point of interest which is not represented by a node, a node should be added to the spatial model. Another alternative is to snap the location of the point of interest to the closest node in the spatial model. In this way it is possible to use the same spatial model while using different sets of points of interest for specific people. Also no adjustments have to be made concerning the spatial model.

When all the points of interest are defined, the user can give target specific weights per time interval to each point of interest. The weights determine the chance that a point of interest is used as prediction of the target in contrast to the other weights of the points of interest. For example, a target is moving towards an elevator and a staircase, but the user of the system knows that this target prefers taking the stairs. In this case the stairs get a higher probability then the elevator. The user determines all these weights for the target. However, the probability of navigating to a point of interest could differ over time. Most points of interest only have a high probability during a certain time interval. For example the coffee machine can be visited all day long by a user and then the weight of the coffee machine is the same during the whole day. But a cafeteria has certain opening hours, which determine the likelihood of the target navigating to this place. During the opening hours the weights should be higher than when the cafeteria is closed.

To define this concept, each point of interest has a list of a certain time interval (Δt) and a probability (*p*) per time interval:

$$Weight(\Delta t) \; = \; [\Delta t_1 \colon p_1, \Delta t_2 \colon p_2, \Delta t_3 \colon p_3, \Delta t_n \colon p_n]$$

These time intervals do not have to be all of the same duration, for example the weights of a cafeteria, with opening hours from 12:00 to 13:00, could be:

$$Weight(\Delta t) = \begin{cases} 0.1\, for\ 08\colon 00 - 11\colon 45 \\ 0.7\, for\ 11\colon 46 - 13\colon 00 \\ 0.1\, for\ 13\colon 01 - 18\colon 00 \end{cases}$$

In this case people are unlikely to navigate to the cafeteria when the cafeteria is closed. However, when the cafeteria is open the probability increases.

These weights could also be extended to a list of weights that is different for each day. For example when there is an event on a specific day, points of interest are different than on another day. This problem is solved by using a list of weights per time interval for each specific day. This process should be repeated for each of the points of interest and acquires a lot of pre-processing time to eventually use the system. However, weights could be determined in advance, so it costs no processing time during the path calculation.

The result of this process is a whole list of points of interest with each their own location and relative weights per time interval. This list is used as input for the determination of the prediction of the target.

3.2.3 Localization of the user and the target

However the aim of this research is not about localization, it is important to address how the locations of the user and the target should be implemented in this framework. For outdoor environments GPS is suitable to locate the user and the target. For indoor environments other positioning systems, like RFID, Bluetooth, WLAN, UWB, Cellular-Based, IR and Ultrasonic (Adalja, 2013) are used to determine the accurate positioning of the user and the target. The positions have to be mapped to the 2D square grid to use them as input data for the path finding algorithm.

Input:

- Position of the user
- Position of the target

Output:

- Tile where the user currently is
- Tile where the target currently is on the user's device

This component consists out of two factors:

- Localization of the user
- Localization of the target

Localization of the user is retrieved by the system of the user itself, for example a mobile device which is used to visualize the path from the user to the target. The sensors of this mobile phone are then used to determine the accurate location. This location is used by the device itself to determine where the user is in the model. Within an outdoor environment the GPS sensor of the mobile device is used to retrieve the up-to-date position.

The localization of the target is different in a way that this data has to be send from the target to the device of the user. When the target is tracked by a mobile device, this mobile device has to send the current location to a database that stores this data. Then the mobile device of the user could access this data by querying the database for the targets current position. This process is depicted in figure 3.6. The GPS sensor would be used to determine the location of the targets device within an outdoor environment.

Important is that these locations have to be determined every 'n' seconds. It is necessary to obtain the locations of both the user and the target in real time, because otherwise false data is used as input for the path finding algorithm.

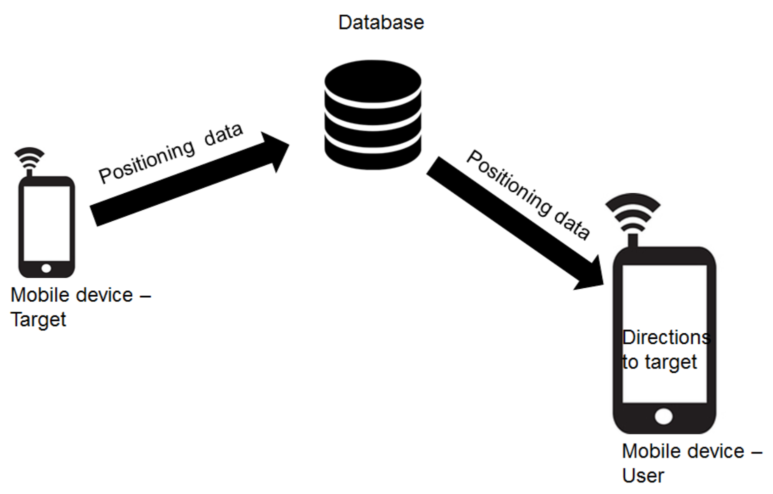

Figure 3.6, retrieving the targets up-to-date location.

When these locations are obtained it is necessary to map these locations to the model. When a regular grid is used for the path finding algorithm, the location of both the user and the target have to be translated to the tiles/shapes of the grid. This process is described in Algorithm 3.3. For each of the positions is checked in which tile they are at the current moment of time. Here the properties of each tile are used to determine where the user and where the target are positioned.

Algorithm 3.3: PositionToGrid

```
1.  Function PositionToGrid(user, target, Squaregrid)
2.    UserTile
3.    TargetTile
4.
5.    For tile in grid:
6.      If tile[min x] > user[x] < tile[max x] and  tile[min y] > user[y] < tile[max y]
7.        UserTile ← tile
8.      Else if tile[min x] > target[x] < tile[max x] and  tile[min y] > target[y] < tile[max y]
9.        TargetTile ← tile
10.   Return UserTile, TargetTile
```

Using an irregular grid, the locations of both the user and the target have to be mapped to the closest node in the spatial model. This node will then be the starting point for the navigation and will determine where the target is located.

3.2.4 Prediction of the target

The most important component of this system is predicting where the target is moving towards. This prediction is used as input to determine the path that the user has to take. This component is a combination of the localization component and the points of interest component. The target's positions determine in what direction the target is moving. A target is moving towards his end goal. The points of interest represent all the possible end goals that are interesting for the target. Therefore the points of interest are used as prediction points of the target. Path finding algorithms are developed to navigate to one single node at the time. This means that the total list of points of interest has to be narrowed down to one point of interest. The movement of the target determines which of the points of interest are approached and which points of interests are not. Therefore in the ideal situation, the length of the path from the target to each of the points of interest has to be calculated. Each time the target moves, the length of these paths should be recalculated and must be compared to the previous length of these paths. In the case that the length of the path, the distance from the target to the point of interest, is decreasing over time, the target is approaching the point of interest. Now that there is a list of approaching points of interests by the target, these have to be narrowed down to one point of interest. Therefore two approaches are defined:

1. The first approach uses the weights per point of interest per time interval to determine the most likely point of interest the target is navigating to. These probabilities are pre-processed in the points of interest component by the user. The point of interest with the highest weight at that moment in time, will be the prediction of the target. When multiple points of interest have the same value for the weights, then the closest point of interest to the target is used.
2. The second approach is not using any probability at all, but is using the closest point of interest the target is heading to as the prediction of the target.

Input:

- Positions of the target
- Collection of points of interest with weights per time interval
- Current time and date

Output:

- The location of the prediction of the target

The first step in this process is to determine the direction of the movement of the target. The positions of the target has to be retrieved from the database. First to initialize, the distance from each point of interest to the target is calculated by using the A* algorithm, then these distances are stored per point of interest. Every following update in the database is used to recalculate the distance from the target to each of the points of interest. Then when the new calculated distance is smaller than the initial distance, the point of interest is added to the points that are approached by the target, the prediction points. Each time when the distance is recalculated this distance is the new up-to-date distance to the specific point of interest. If there is no point of interest that is approached by the target, then the target itself will be used as the prediction. This process of checking which points of interest are approached by the target is described in Algorithm 3.4.

Algorithm 3.4: Prediction points

```
1.  Function Predictionpoints(points of interest, first location of the target, new location of
    the target)
2.    Predictionpoints
3.    For point in points of interest:            //initialize the distance
4.      Point.Distance = A*( first location of the target , point)
5.
6.    While new location of the target:
7.      For point in points of interest:
8.        NewDistance = A*(new location of the target, point)
9.        If NewDistance < Point.Distance:
10.         Predictionpoints ← point
11.       Point.Distance = NewDistance        //Update the initial distance
12.
13.     If Predictionpoints is empty:
14.       Predictionpoints ← Target
15.
16.     Return Predictionpoints
```

Now only the points of interest that are approached by the target are obtained, these points are used to select the most likely point of interest the target is navigating to. When there is only one prediction point this point will be used as input for the path finding algorithm. When there are multiple prediction points, then these could be narrowed down by the two approaches as described before.

The first approach is determined by using the weights that are given per time interval in section 3.2.2. The time and the date are used to determine the weight of each point of interest at that moment in time. Then these weights are compared to each other and the point of interest with the highest probability is chosen as the predication point (Algorithm 3.5).

The second approach uses the closest point of interest compared to the location of the target, the pseudocode for the second approach is described in Algorithm 3.6.

In this case only one point of interest or the position of the target is used as input for the path finding algorithm. The prediction is used as the end goal in the iterative A* search. The targets position is used as prediction when the target is standing still or the target is not moving to any of the predefined points of interests. Otherwise the most likely point of interest is used as a prediction for the target.

Algorithm 3.5: Approach 1: Using weights

```
1.  Function Prediction(PredictionPoints)              //Input Algorithm 3.4
2.
3.     If length(PredictionPoints) = 1:
4.         Prediction ← PredictionPoints[0]
5.     Else:
6.        Prediction.weight = 0                        //initialize a prediction with weight 0
7.        time = current time
8.        date = current date
9.        PredictionPoints(Weight) ← time, date       //get weight at the current time and date
10.
11.       For point in PredictionPoints:
12.          If point.weight >= Prediction.weight:
13.             Prediction ← point
14.       If length(Prediction) < 1:
15.          Prediction ← min(point.distance)    //point.distance is obtained from Algorithm 3.4
16.
17.    Return Prediction
```

Algorithm 3.6: Approach 2: Using the closest distance

```
1.  Function Prediction2(PredictionPoints)    //Input Algorithm 3.4
2.
3.     If length(PredictionPoints) = 1:
4.         Prediction ← PredictionPoints[0]
5.     Else:
6.        Prediction ← min(point.distance)       //point.distance is obtained from Algorithm 3.4
7.
8.     Return Prediction
```

3.2.5 Path planning algorithm

The last component uses the user's current position and the prediction of the target, obtained by the component described in the previous section, to calculate a path for the user. In the theoretical framework is discussed that an iterative A* algorithm is suitable for this application.

Input:

- Position(s) of the user: Node
- Position(s) of the target: Node
- The prediction(s) of the target: Node
- Spatial model

Output:

- Path from the user to the prediction of the target or the target itself

Important to notice is that navigating to a prediction of the target is not always faster than navigating to the target itself. There are six cases that could be distinguished, by comparing the distance of the paths from the user and the target to the predicting of the target and the location of the user and the target compared to the prediction of the target. The length of the path from the target to the prediction of the target is determined by the A* algorithm presented in Algorithms 2.1 and 2.2. The same determination is executed by the path from the user to the prediction of the target. These are represented as:

- Length path user
- Length path target

Then the position of the target and the user compared to the prediction of the target is important to determine different situations. Both the target and the user could be on the same side of the prediction of the target. Together these parameters determine the six following different cases:

- Case 1: Length path target > Length path user & user and target are on the other side compared to the prediction of the target.
- Case 2: Length path target = Length path user & user and target are on the other side compared to the prediction of the target.
- Case 3: Length path target < Length path user & user and target are on the other side compared to the prediction of the target.
- Case 4: Length path target > Length path user & user and target are on the same side compared to the prediction of the target.
- Case 5: Length path target = Length path user & user and target are on the same side compared to the prediction of the target.
- Case 6: Length path target < Length path user & user and target are on the same side compared to the prediction of the target.

These six cases are visualized in figure 3.7. Only for cases 2, 3 and 6 navigating to the prediction is more effective than navigating to the target directly. In the cases 2 and 6 the user and target are likely to meet each other at the point of interest, representing the prediction of the target. Case 3 the target will reach this point as first, therefore the user should navigate towards this prediction. For case 4 navigating to the target itself would be more effective over navigating to the prediction of the target. This is because the user is could intercept the target before the target reaches the prediction of the target. Case 5 depends on the situation, navigating to both the prediction of the target of the target itself could be faster, but with small corridors the target itself would be better. Case 1 represents another problem, where the user first has to navigate to the prediction of the target. Then this problem becomes the same as in case 4, if the user reaches the prediction of the target, navigating to the target itself is more effective than waiting for the target to navigate to the prediction. Therefore in case 1, the user should navigate to the position of the target, while visiting the location of the prediction of the target.

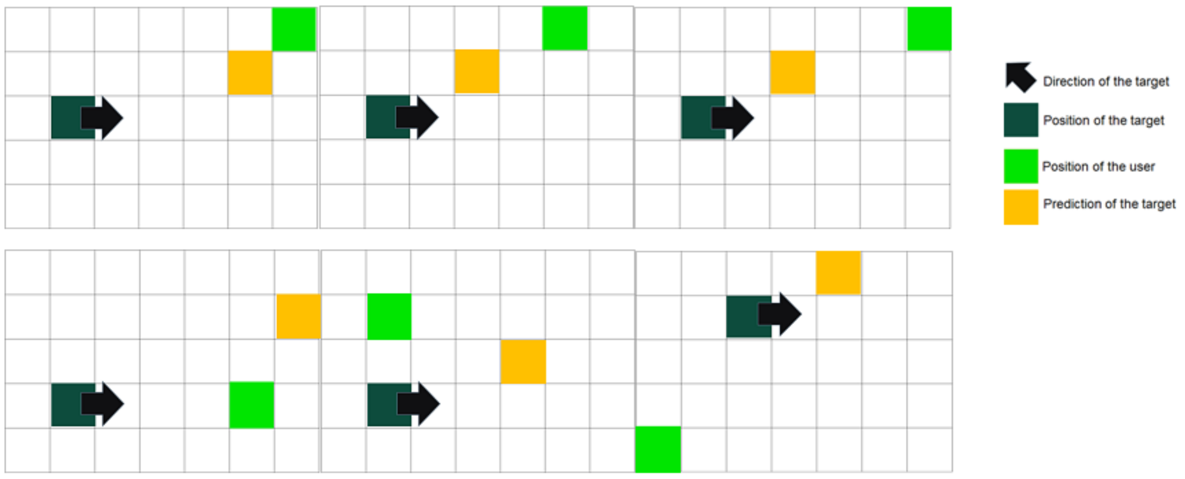

Figure 3.7, different cases as input for the path planning algorithm. Top: left: case 1, middle: case 2 and right: case 3. Bottom: left: case 4, middle: case 5 and right: case 6.

For the path finding the Algorithms 2.1 and 2.2 are used to calculate the fastest path at any given moment in time. The heuristic used for the A* algorithm is the Octile distance in a square grid and the Euclidean distance in an irregular grid, as presented in the theoretical framework. Algorithm 3.7 represents this process of using A* to calculate a path that the user has to take, taking into account the six developed cases.

Algorithm 3.7 only returns the path to the user at a given time. Now this has to be executed iterative, which is shown in Algorithm 3.8. Each time the user has not found the target, a path is calculated, the user moves according to this path, until new input data is obtained. When new input data is obtained, the calculation of the prediction of the target is updated and the search continues. This means that the process is ended when the user and target reached each other. This is determined by a distance buffer, where the user and the target should see each other. This is the minimal distance that is required to actually meet someone. Ideally this distance is determined by the visibility of the user, however it is hard to determine in what specific situation it is possible to actually see the person.

Algorithm 3.7: Determine path for the user

```
1.  Procedure UserPath(UserNode, TargetNode, Prediction)
2.
3.    PathUser = A*(UserNode, Prediction)
4.    PathTarget = A*(TargetNode, Prediction)
5.    PathUserLength = length of PathUser
6.    PathTargetLength = length of PathTarget
7.
8.    If (UserNode[x] <= Prediction[x] <= TargetNode[x] or UserNode[x] >= Prediction[x] >= TargetNode[x]) and
9.      (UserNode[y] <= Prediction[y] <= TargetNode[y] or UserNode[y] >= Prediction[y] >= TargetNode[y]):
10.         SameSide = False
11.   Else:
12.         SameSide = True
13.
14.   If SameSide = False:
15.     Return PathUser
16.   Else if PathTargetLength < PathUserLength and SameSide = True:
17.     Return PathUser
18.   Else:
19.     PathUserToTarget = A*(UserNode, TargetNode)
20.     Return PathUserToTarget
```

Algorithm 3.8: Determine movement for the user

```
1.  Procedure UserMovement(UserNode, TargetNode, Prediction, BufferDistance)
2.
3.    while Distance(UserNode, TargetNode) > BufferDistance:
4.      Path = UserPath(UserNode, TargetNode, Prediction)
5.      userMovement = Path(next step)
6.
7.    Return user reached the target
```

3.2.6 Detailed system flow

Figure 3.8 gives a detailed system flow of all the components and every decision that is presented in the previous sections. The spatial model is used as input for all the processes presented in this system flow. The inputs are presented at the left and decisions are represented as diamond shapes. Note that the determination of the most likely point of interest the target is navigating to could be determined by the two different approaches as presented in section 3.2.4.

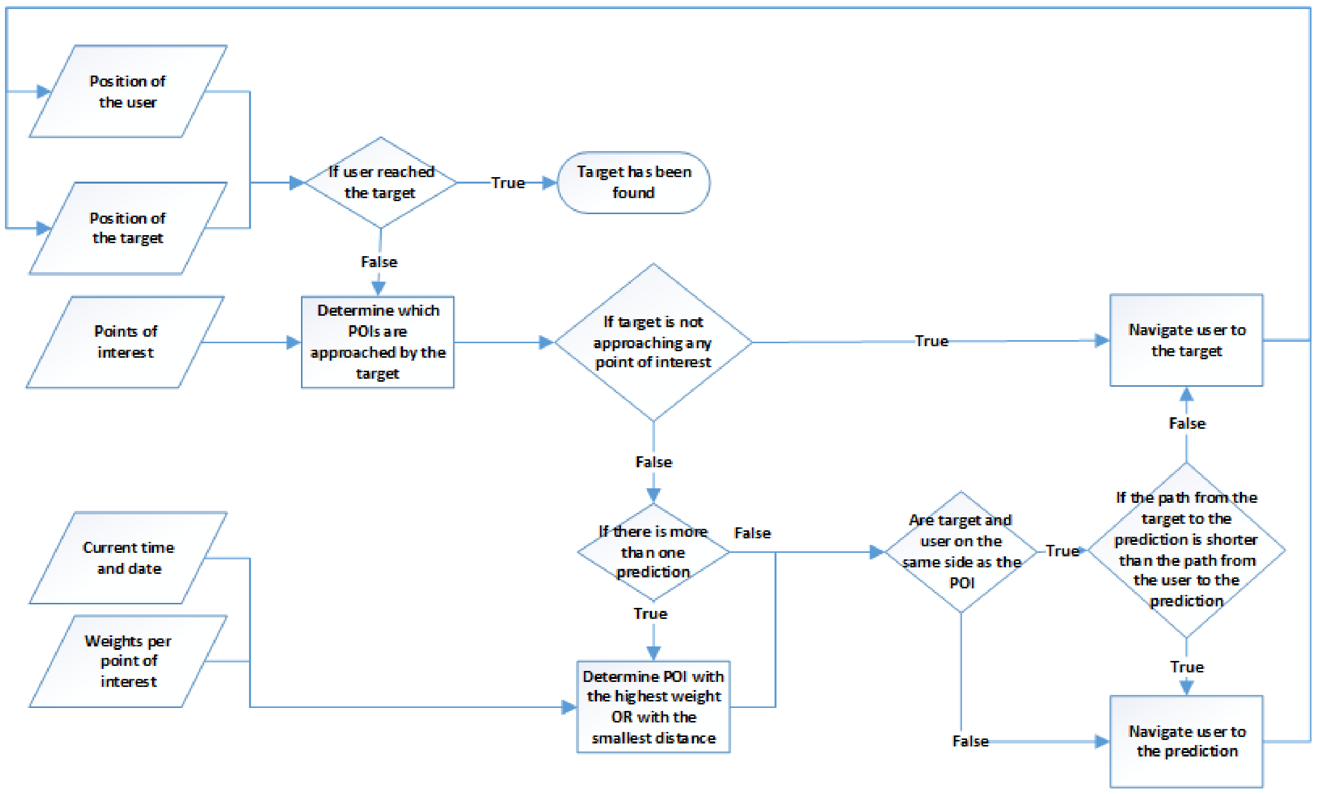
Position of the user
Position of the target
If user reached the target
True
Target has been found
False
Points of interest
Determine which POIs are approached by the target
If target is not approaching any point of interest
True
Navigate user to the target
False
False
If the path from the target to the prediction is shorter than the path from the user to the prediction
Current time and date
Weights per point of interest
If there is more than one prediction
False
Are target and user on the same side as the POI
True
True
True
Determine POI with the highest weight OR with the smallest distance
False
Navigate user to the prediction

Figure 3.8, detailed work flow of the SEA method.*

4. Implementation: Indoor environment

The aim of this chapter is to implement the methodology presented in chapter 3 of this thesis research. The dataset that is used to implement the conceptual framework is a floorplan from the faculty of architecture of the University of Delft. This floorplan is representing an indoor environment. There is no indoor positioning system available and therefore simulation of the positioning of the user and the target will be used to test the methodology. Because there is no existing application that provides the data that is needed to test this method, a new theoretical application will be made. This application is written in the programming language Python.

Section 4.1 discusses the implementation of the methodology. A spatial model must be established, with the geometrical, topological and semantic components as presented in the conceptual framework. The results of this implementation will be presented in section 4.2 where a normal iterative A* algorithm is used to validate the SEA* method. The positions are simulated and the weights are determined per scenario. Section 4.3 gives an overview of all the results described in section 4.2. Also it discusses situations that are hard to visualize, because the method is dynamic. Finally section 4.4 discusses the requirements that are set for this method in chapter 3.

4.1 Pre-processing

First the dataset that is used for the implementation is discussed, section 4.1.1. Then this dataset is translated to a spatial model, section 4.1.2. The spatial model used for this implementation is a 2D square grid. This spatial model is chosen because of three reasons. First of all a 2D square grid gives an accurate description of the positions of the user and the target. The second reason is that a square grid supports using the A* algorithm, which is the used path planning algorithm. The third reason is that a square grid supports adding semantics to the model, because semantics could be added to every tile in the model. Overall the 2D square grid is an easy and clear spatial model to work with and to implement this method. Also a square grid is the best representation according to Algfoor (2015) & Ma (2011) for a 2D environment. Due to the lack of optimizing the developed code, the approached points of interest by the target are calculated differently, this is explained in section 4.1.3. Finally the points of interest are determined by using the semantics of the building, which is described in section 4.1.4.

4.1.1 The indoor dataset

The dataset that is used is a floorplan from the faculty of architecture of the University of Delft, see figure 4.1. This dataset is used because of four factors:

- The used dataset must contain shortcuts in the building. This method is developed to let the user take shortcuts instead of following the target. For example in a building with only one big corridor there are no possible shortcuts. This method will then have the same behavior of a normal iterative target search, which is following the target.
- The environment must be known by the developer of the system, because probability weights must be given to the points of interest.
- There are different users of the building, like teachers, students, visitors etc. Each user has its own points of interest and therefore different behavior can be tested.
- This method gives more insight in the way of navigating inside the faculty of architecture, which could be used for further research in this domain.

Concerning the first factor, only a part of the building of Architecture contains shortcuts for navigation purposes. Therefore only this part, see figure 4.1, will be examined. The floorplan contains data about where what room is located, the obstacles and the navigable space. However, it is not possible to automatically translate this floorplan into a representative model. This is caused by extra information about where what room is located, doors are not modelled as white space and there are some obstacles in the floorplan that are no real obstacles for human navigation.

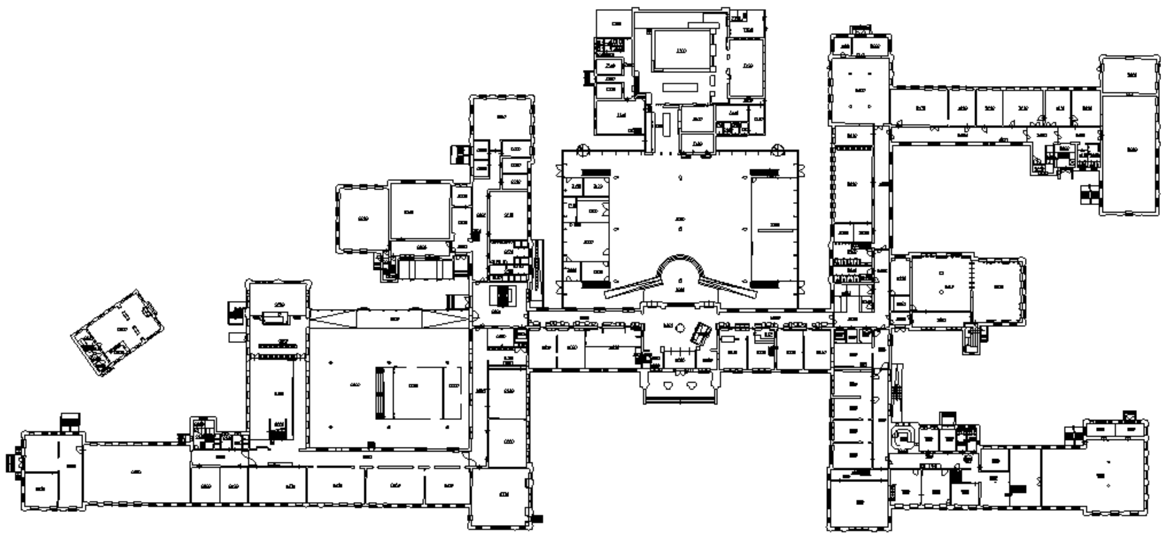

Figure 4.1, floorplan of the faculty of architecture of the University of Delft (Julianalaan 134, Delft)

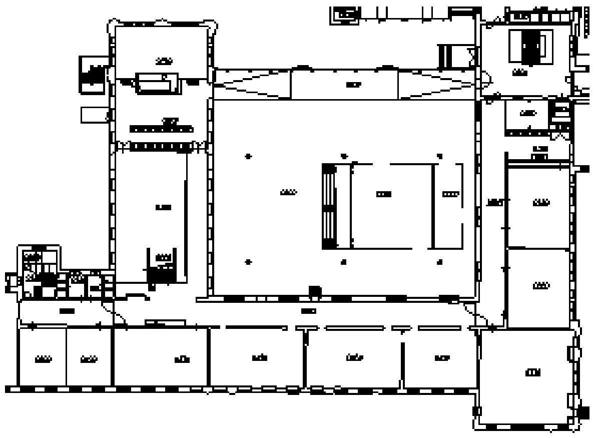

Figure 4.2, selected part from the floorplan of the faculty of architecture.

4.1.2 Creating the spatial model

It is not possible to automatically generate a 2D square grid with this floorplan. Therefore the spatial model is constructed manually. The floorplan is crossed by an empty grid and for each tile is determined, whether a human could navigate in this space or not. Special attention to doors and small obstacles is necessary to give a representative model. The used size of the tiles is 1 m². The result is an occupancy grid as displayed in figure 4.3. In this case brown tiles represent the obstacles, like the walls, and the white tiles represent the navigable space, like the doors and the empty space. This grid has a width of 78 meters and a height of 58 meters.

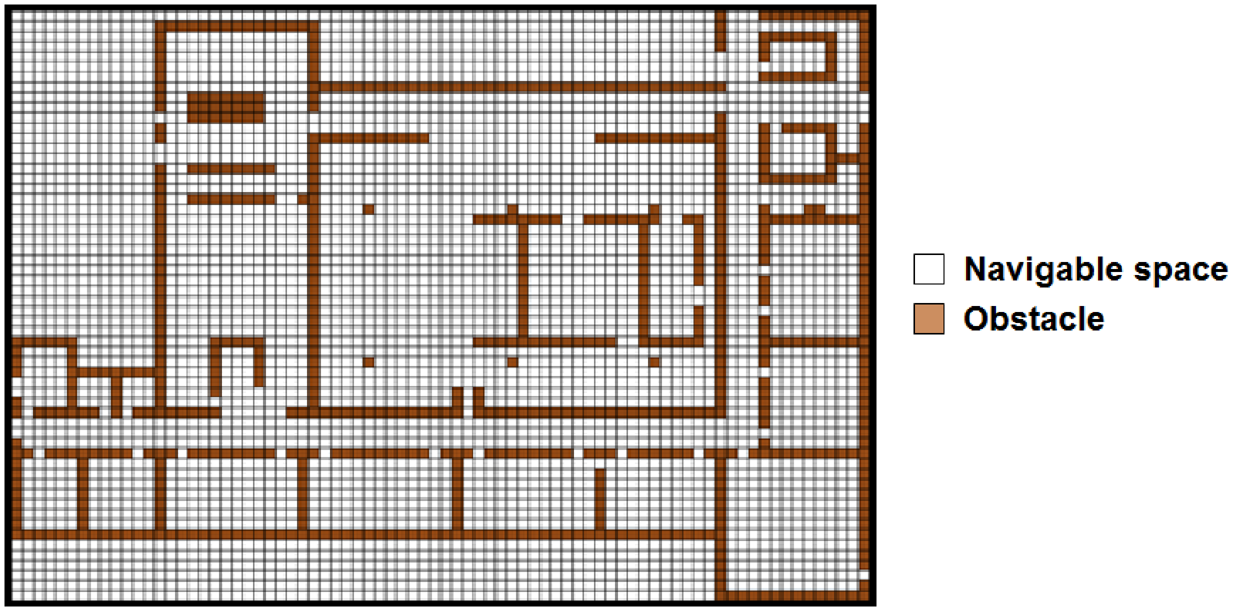

Figure 4.3, 2D square grid representation.

For each navigable space it is possible to navigate to each of the neighboring tiles, if these tiles are also representing a navigable space. Therefore diagonal movement is allowed. To deal with this given factor moving horizontal and moving vertical has a movement cost of 1. While moving in a diagonal way has a movement cost of $\sqrt{2}$.

4.1.3 Calculating the approached points of interest

It turns out that calculating a path to each of the points of interest each time the target moves and checks what points are approached by the target is very inefficient and cannot be optimized due to the lack of programming skills. Therefore the direction that represents the movement of the target is used in combination with the positions of the doors and the knowledge of the current rooms of the user and the target. For this approach the doors per room and the tiles per room have to be known. The doors are needed to check if a person could leave the room if he is heading that direction or not. Therefore for each room all the possible exits are stored. All the doors and exits are manually extracted from the floorplan, see figure 4.4.

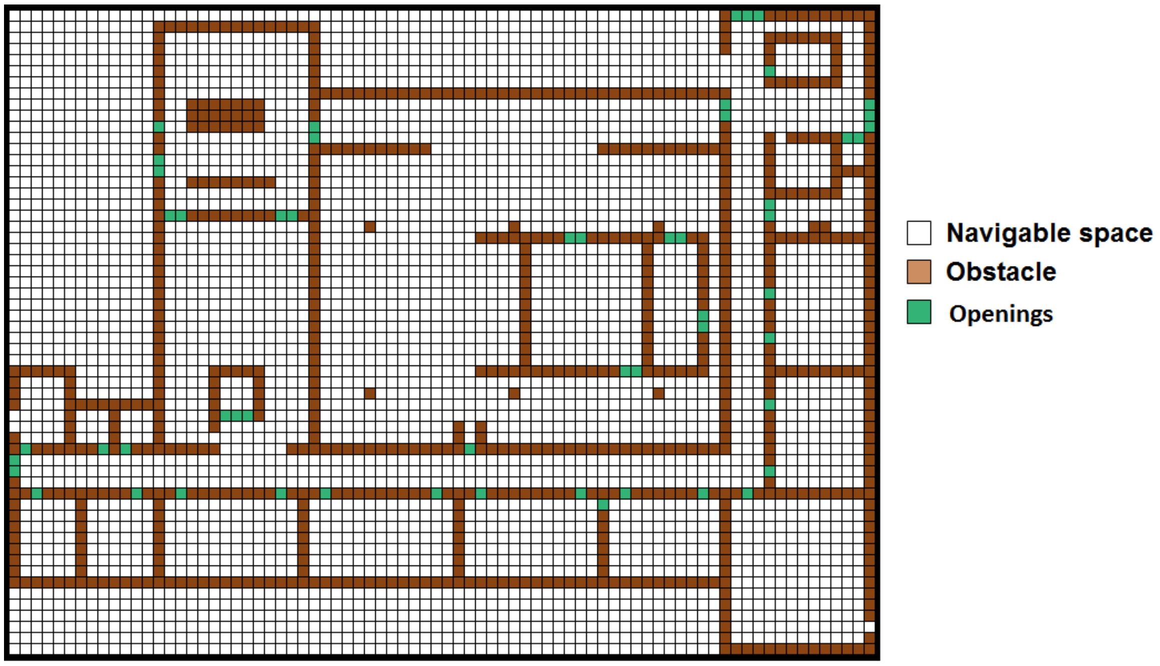

Figure 4.4, determination of all the doors/exits in the model.

The position of the target is compared to the tiles per room. Then the doors per room determine whether the target could reach a point of interest or not. All the doors and openings that are available in the current situation are modelled. However, not all doors could be used by a certain type of users. This depends on closed doors, emergency exits and doors that are always open to everyone.

To calculate the direction of the target, two measurements of the position of the target has to be known. Because the magnitude of a vector only has some meaning in an open space and not in navigating in an environment with obstacles, the vector is normalized and only the direction is taken into account. The direction of the movement of the target is determined by the vector.

$$\vec{a} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$$

In this case a_1 represents the normalized change in the horizontal direction and the a_2 represents the normalized change in the vertical direction. This direction of the target determines to what points of interest this person is navigating to. This is the first step to narrow all the points of interest down to only one single point that is used for the navigation. Figure 4.5 shows how the direction is used to focus on the points of interest where the target is moving towards. There are eight different cases:

- When the direction of the target is $\begin{matrix}1\\1\end{matrix}$ then the search space for points of interest is in the North-East quadrant (II).
- When the direction of the target is $\begin{matrix}1\\-1\end{matrix}$ then the search space for points of interest is in the South-East quadrant (IV).
- When the direction of the target is $\begin{matrix}-1\\1\end{matrix}$ then the search space for points of interest is in the North-West quadrant (I).
- When the direction of the target is $\begin{matrix}-1\\-1\end{matrix}$ then the search space for points of interest is in the South-West quadrant (III).
- When the direction of the target is $\begin{matrix}1\\0\end{matrix}$ then the search space for points of interest is in the East quadrants (II & IV).
- When the direction of the target is $\begin{matrix}-1\\0\end{matrix}$ then the search space for points of interest is in the West quadrants (I & III).
- When the direction of the target is $\begin{matrix}0\\1\end{matrix}$ then the search space for points of interest is in the North quadrants (I & II).
- When the direction of the target is $\begin{matrix}0\\-1\end{matrix}$ then the search space for points of interest is in the South quadrants (III & IV).

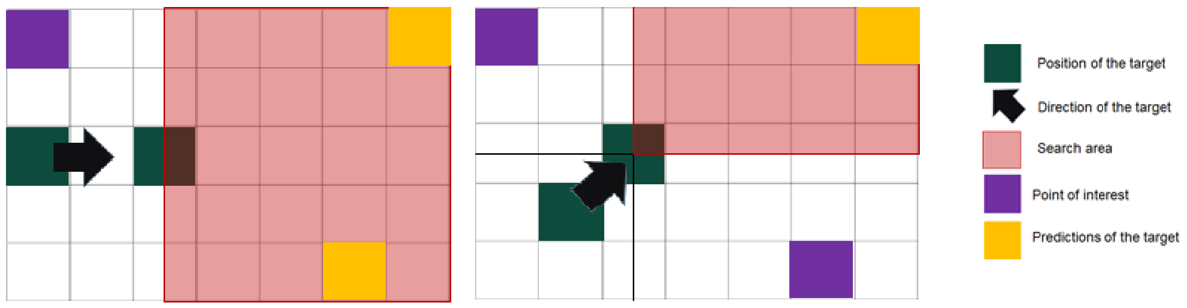

Figure 4.5, two examples of selecting the prediction points based on the direction of the target.

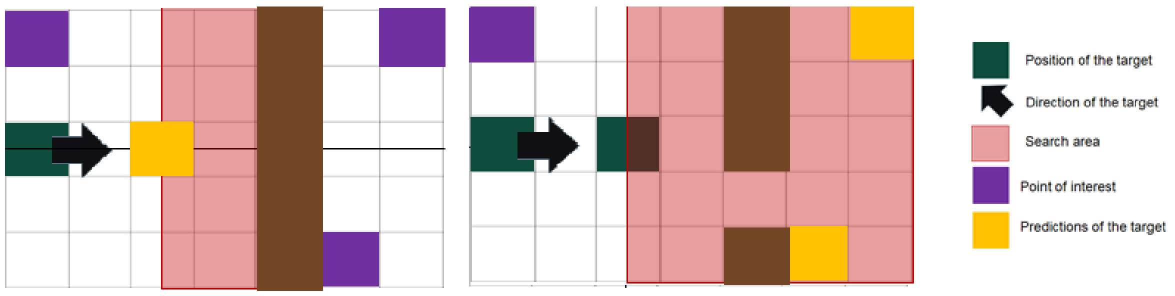

Figure 4.6, the prediction points based on the direction of the target without (left) or with (right) a door.

Figure 4.6 shows how the direction is used in combination with obstacles to focus on the points of interest where the target is moving towards. Now only the points of interest where the target can navigate to are obtained, and if there is no point of interest in the search area, the current position of the target is used as the prediction point (figure 4.6, left). When there is only one prediction point this point will be used as input for the path finding algorithm. When there are multiple points of interest, then the most likely point is determined as described in 3.2.5.

4.1.4 Determination of the points of interest

To predict where the target is moving to, points of interest should be identified. A point of interest could be any of the tiles that is interesting for the target to navigate to. Doors are important decision making points for the behavior of a target. Therefore doors could be used as points of interest. There are rooms with only one door. These doors are the only possibility to enter a certain room or leave a certain room. Each room with only one door/entrance has this door/entrance as points of interest, representing the total room. Rooms with multiple doors are represented by a point of interest in the middle of the room. The doors to the outdoor environment are all important points for the target to navigate to and therefore used as points of interest. Other objects are represented by the tile where the person interacts with the object. In case there is no emergency, emergency doors should not be used by the target, therefore these doors are in a normal situation not marked as points of interest. Next to using doors that represent the rooms as points of interest, some points are more important. For example in the faculty of architecture people often visit a coffee machine or a coffee corner. In case of an event, the lobby is becoming a point of interest for people visiting the event.

Figure 4.7 shows special cases that represent points of interest. In a normal situation the closed door, the door of a storage space, and the emergency exits are not used as points of interest. However, the coffee machines and the location where an event is held are important points to use as possible predictions of the target.

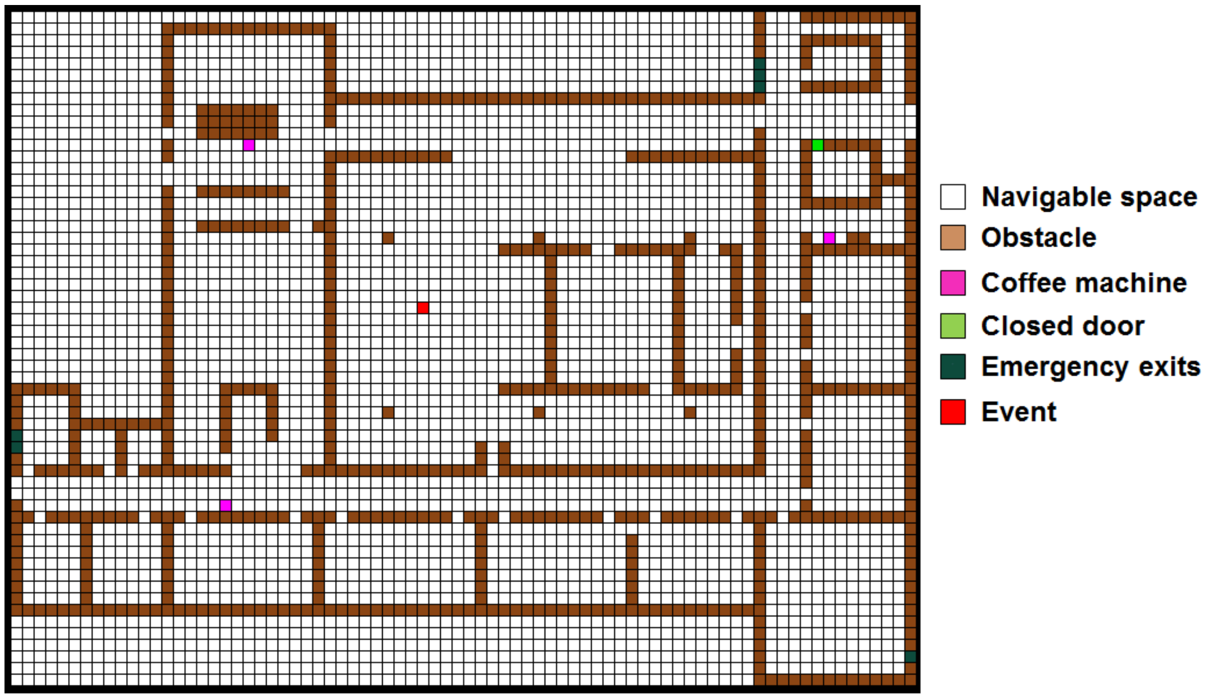

Figure 4.7, special cases to determine points of interest.

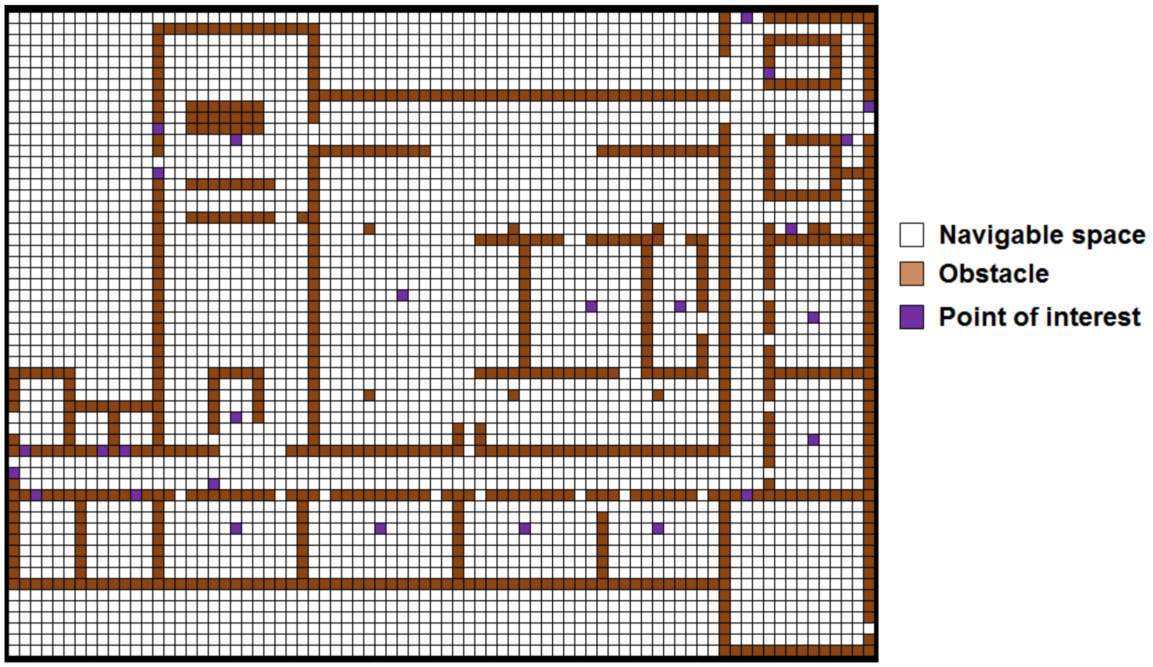

Figure 4.8, all points of interest in a normal situation.

Figure 4.8 shows all the points of interest where the target could navigate to in a normal situation. For each of these points a weight must be given concerning the target the user is navigating to. The next section describes the weights per created scenario. In total there are 26 points of interest defined. These points of interest are representing 3 coffee machines, 4 study rooms, 6 lecture rooms, 1 elevator, 2 staircases, 5 exits (2 outdoor exits and 3 indoor exits), 2 bathrooms, 1 event location and 2 rooms with other destinations.

4.2 Scenario analysis

This section describes several scenarios based on real situations. For each of the scenarios, the position of the user and the position of the target are simulated. Also the path the target takes is simulated in this implementation. The results per scenario are given, this means the path the user takes to reach the target. This path is per scenario compared to the iterative implementation of the A* algorithm. Sometimes the methods have overlapping paths, this means that the iterative A* method takes the same route as the SEA* method.

4.2.1 Scenario 1: Target moves from the hall to the coffee corner, user is in a study room

In this scenario the target is moving from the one side of the building to get a coffee in the other side of the building. The path of the user is the shortest path between the target and the coffee corner that he wants to visit. The user is currently in a study room. Figure 4.9 shows the initialization of this scenario.

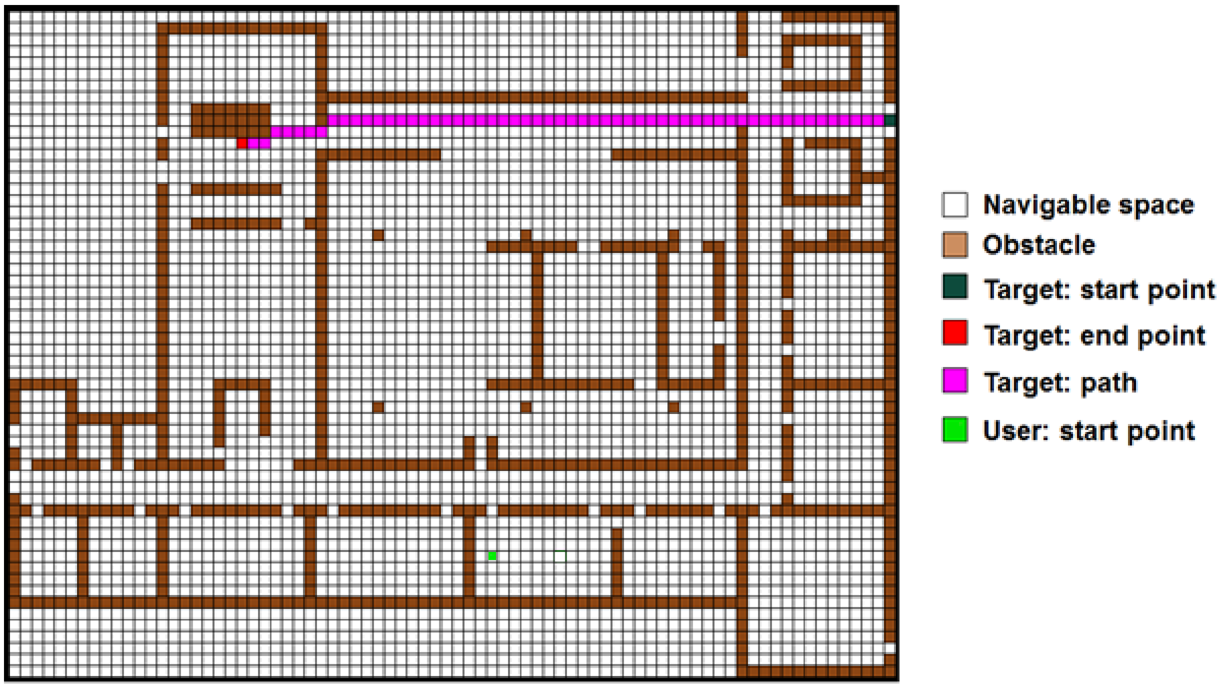

Figure 4.9, scenario 1 initialization of the target and the user.

Weights per point of interest per time interval are needed to determine the most likely point of interest. To test the recommended approach one weight per point of interest is used to represent the likelihood that the target is navigating to this point of interest. The implementation therefore uses a specific time interval. The time interval in this case is a 15 minute break between a lecture. The target is in this case a male geomatics student. On this day there is no special event planned. The reasoning and the weights that are assigned to the points of interest are presented in table 4.1. Now these weights are used to predict the movement of the target. The Geolab, coffee machines, the bathroom and the outdoor exits are the most important points of interest during a break. These points will have the biggest impact on the SEA* approach.

Table 4.1, weights per point of interest during a break for a male geomatics student.

POI	Chance	Reasoning
Lecture rooms	0.1	It is not likely that a student is visiting another lecture room in the break of its own lecture room.
Coffee machines	0.8	During a break most people are visiting a coffee machine.
Bathroom	0.7 [0.0]	People are likely to visit the bathroom during lecture breaks. However a male will only visit the male toilet, the female toilet will have a chance of 0.
Elevator, stairs and indoor exits	0.4	Students will probably not visit another floor during the break, but this is not impossible.
Geolab	0.7	Geomatics students often visit the Geolab.
Study rooms	0.1	Someone could visit another study room than the Geolab, but this is unlikely.
Outdoor exits	0.7	Students are likely to go outside during the break.
Event	0.0	There is no event scheduled, therefore this is not modelled as a point of interest.
Other rooms	0.1	It is very unlikely that students are spending their breaks here.

The result of the SEA* method and the result of the iterative A* algorithm is shown in figure 4.10. First both methods navigate east. When the distance of the target to the prediction is equal or smaller than the distance of the user to the prediction the SEA* method turns and navigates to the prediction of the target. The method detects that the coffee machines are the most likely points of interest. The iterative A* algorithm is navigating to the target at each moment in time and therefore follows the target. The SEA* method reaches the target in 57 tiles, while the iterative A* method reaches the target in 96 tiles.

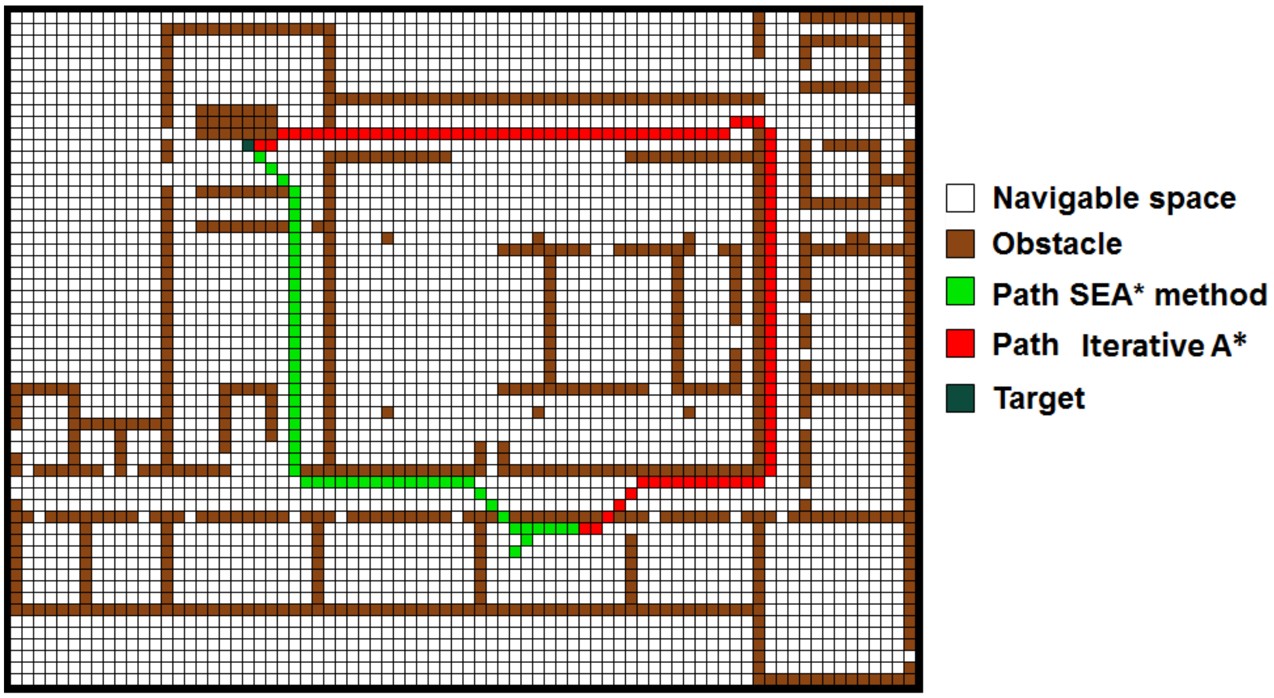

Figure 4.10, result scenario 1 with weights used in table 4.1. SEA: 57 steps & iterative A*: 96 steps.*

The weights of the coffee machines are now dominant over the other points of interest. Figure 4.11 presents the case where the Geolab, the study room of geomatics students, has a higher weight than the coffee machines. Both methods reach the target in this case in 96 tiles.

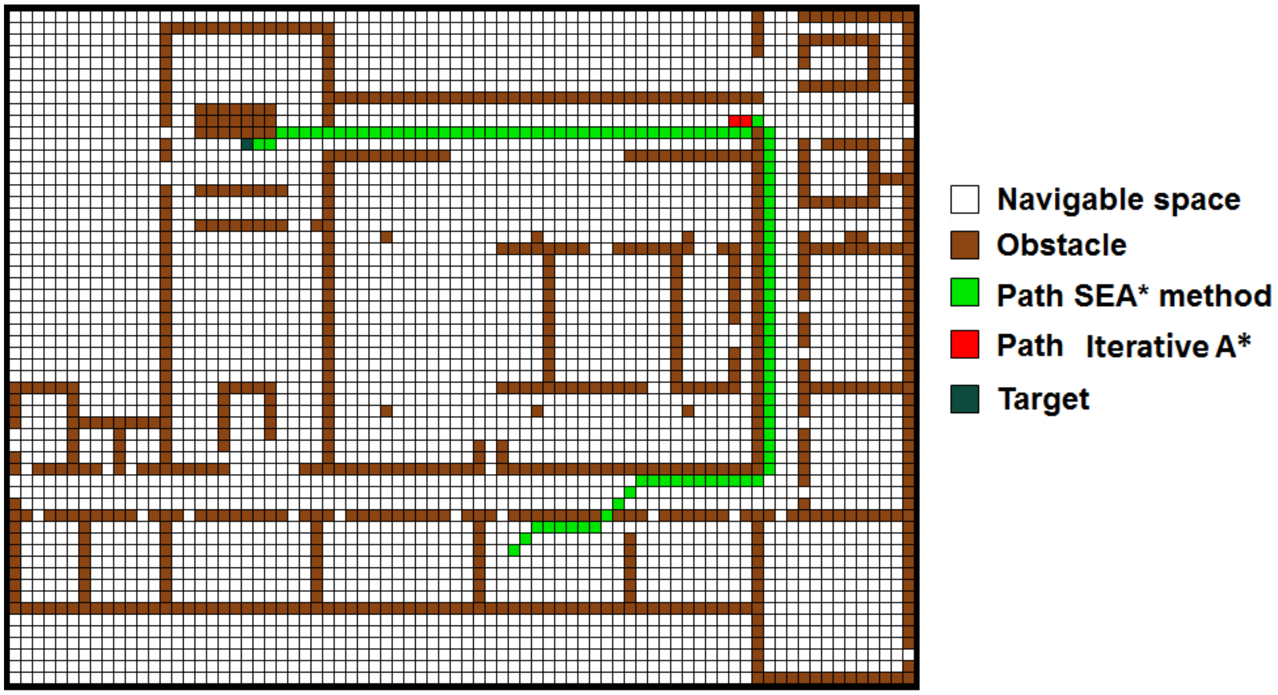

Figure 4.11, result scenario 1 with the Geolab with the highest weight assigned. SEA: 96 steps & iterative A*: 96 steps.*

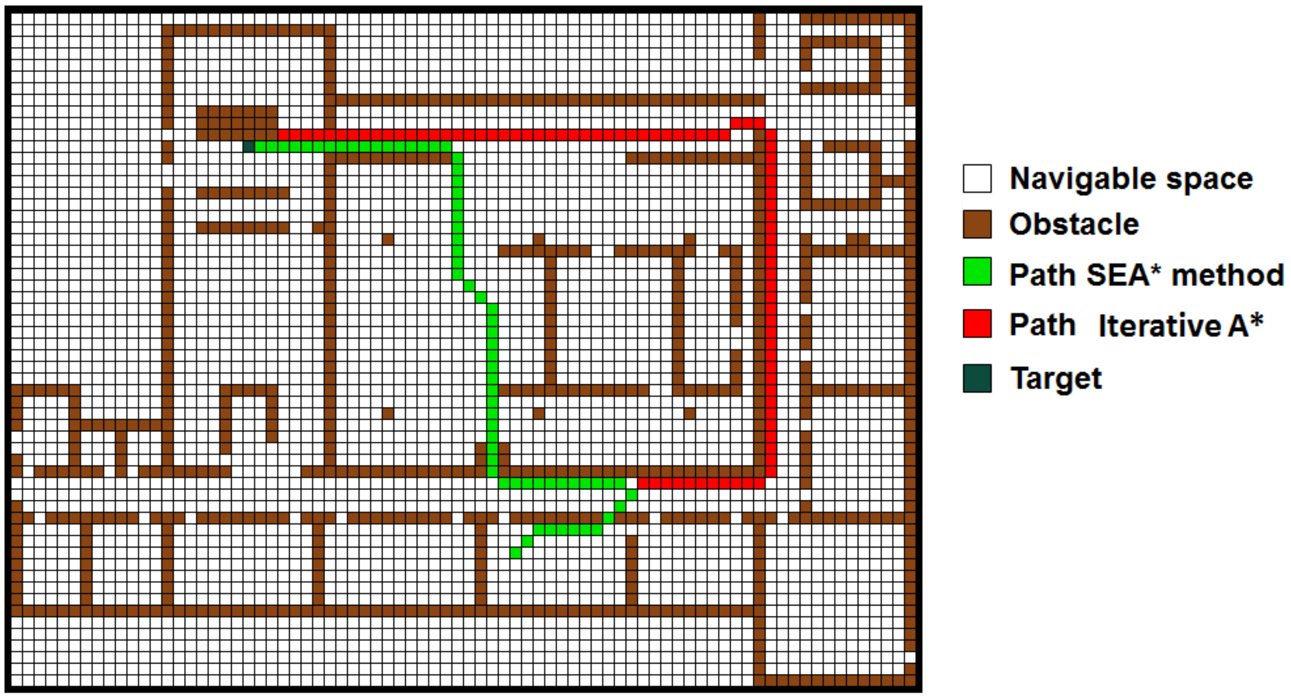

Figure 4.12, result scenario 1 with equal weights for all points of interest. SEA: 67 steps & iterative A*: 96 steps.*

When there are no predefined weights per point of interest, every point has the same weight. At each time the closest point of interest in the direction the target is navigating is used as the prediction point, see figure 4.12. The result for the SEA* method is that the user reaches the target in 67 steps. The iterative A* method reaches the target in 96 steps.

The situation where the user predefined the weights correctly is the fastest method (figure 4.10). However, when the user predefines a high weight to a point of interest where the target does not navigate to, this point becomes the leading factor for the SEA* method. Because this point is further away from the target then from the user, the user navigate directly to the target. In this case both methods behave as the iterative A* method.

When there are no predefined weights the closest point of interest to the target is used as the prediction point. This scenario is slower than a situation where the weights are predefined correctly. On the other side, no predefined weights are faster than the situation where the user gives a high weight to a point of interest the target is not navigating to.

In all the three variations of this scenario the user reaches the target. This is important, because both approaches must always be effective.

4.2.2 Scenario 2: Target leaves the building during closing hours

The second scenario presents the architecture building just before closing hours. Everyone is asked to leave the building. The building closes from Monday to Thursday on 22:00. From 21:45 only the exits of the building are points of interests for people in the building. Therefore these points have a higher weight than all the other points. The outdoor exits and the indoor exit on the east are the only locations where a person can leave the building. Therefore only these are considered in the model. The target is moving from the Geolab to the exit on the west side of the building. The user is currently in the lobby. The initialization of this scenario is depicted in figure 4.13.

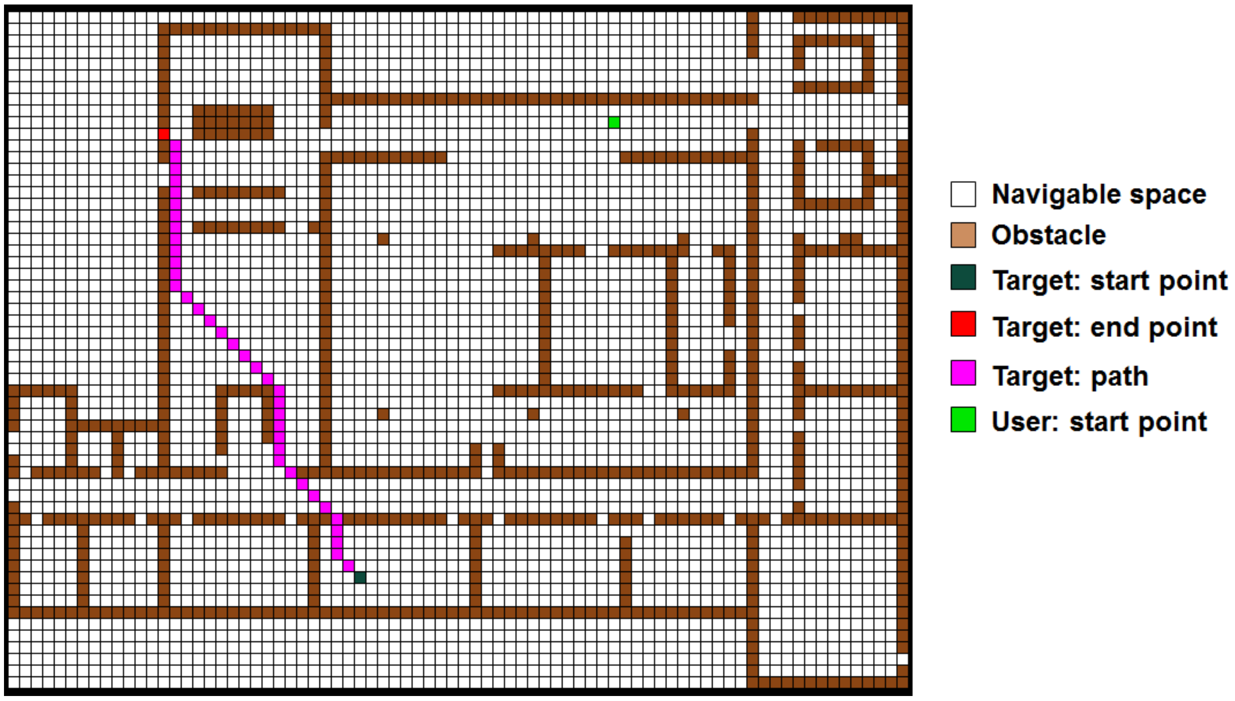

Figure 4.13, scenario 2 initialization of the target and the user.

The result of this scenario where the weights of the possible exits are higher than the other points are depicted in figure 4.14. The SEA* method navigates to the prediction, which is in this case the exit door, and reaches the target in 37 steps. The iterative A* method navigates towards the target and reaches the target in 74 steps.

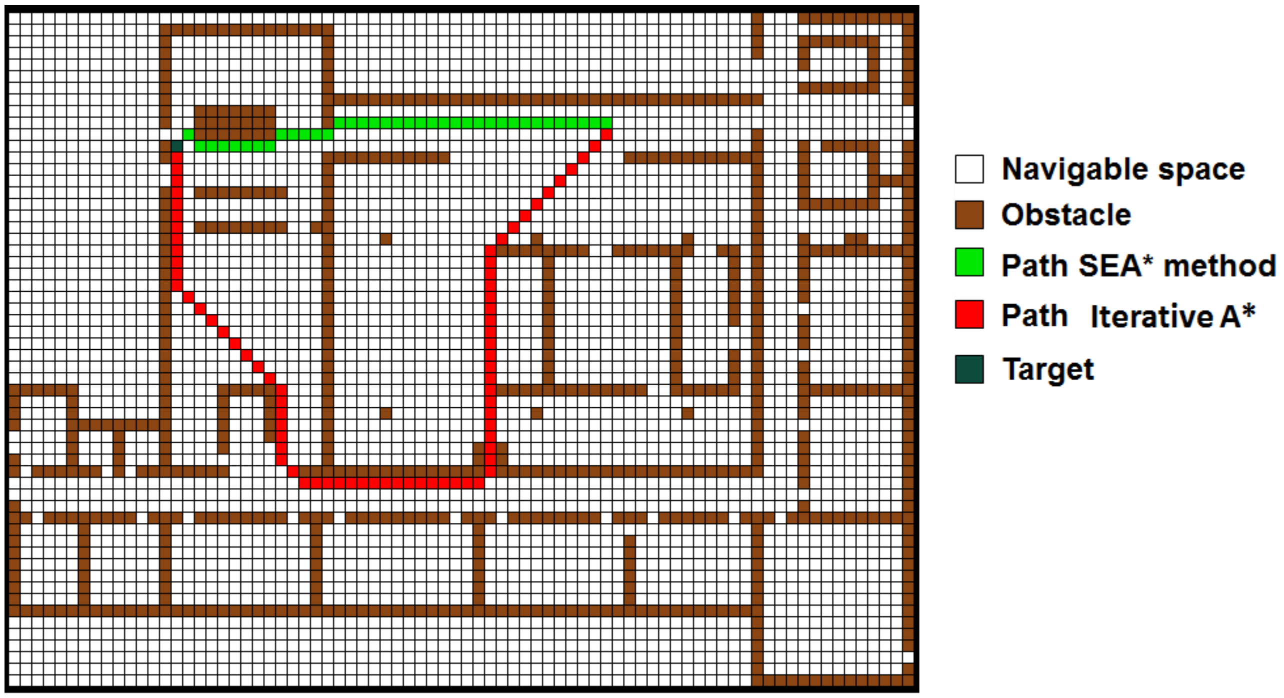

Figure 4.14, result scenario 2 with higher weights for the exit doors. SEA: 37 steps & iterative A*: 74 steps.*

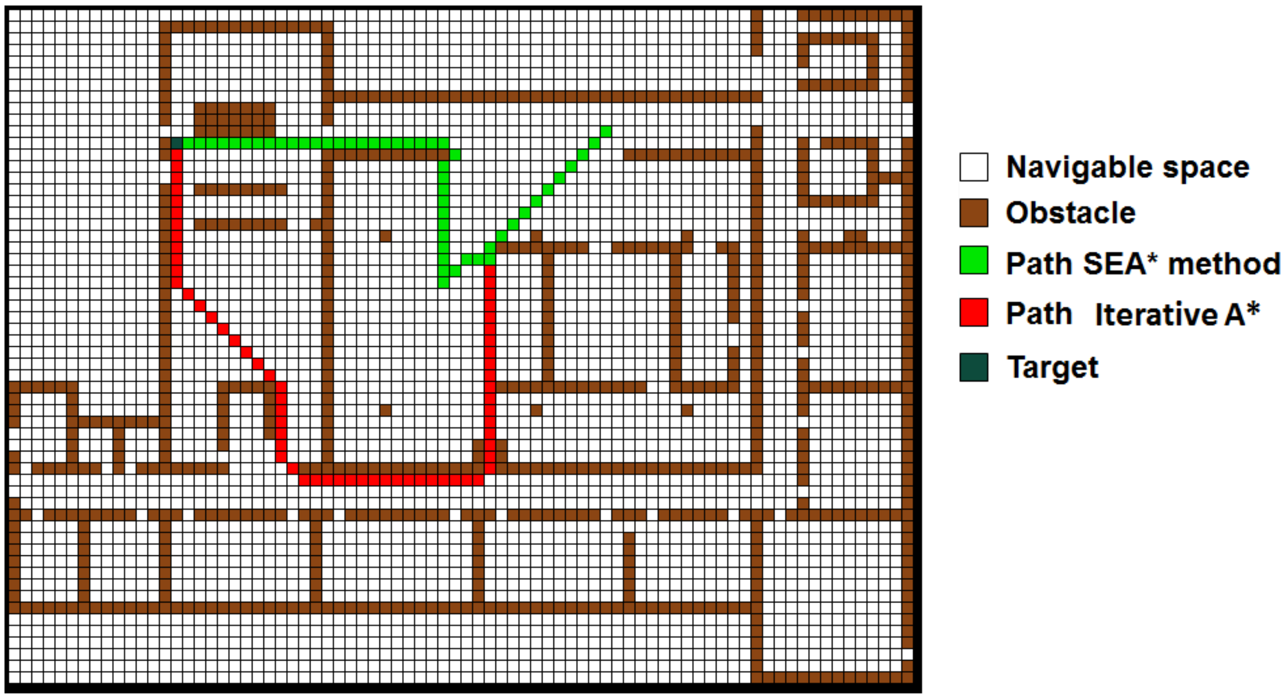

Figure 4.15, result scenario 2 with the same weights for all points of interest. SEA: 50 steps & iterative A*: 74 steps.*

Figure 4.15 shows the behavior of the model where all the weights of the points of interest are the same. In this case the SEA* method first navigates to points of interest that are closer by the target and then navigates towards the exit doors. The SEA* method reaches the target in 50 steps, while the iterative A* method reaches the target in 74 steps. Therefore this method is faster than the iterative A* method, but slower than the situation where correct weights are given (figure 4.14).

To not limit the results for only one specified path, multiple locations of the user are tested to show the behavior of the both presented methods. The path of the target is the same as shown in figure 4.13. Figures 4.16 to 4.18 present the results of both methods with different beginning points for the user.

The first position of the user is depicted in figure 4.16, this is the beginning of the light green line towards the target. The position of the user is in between the target and the destination of the target. Both methods navigate directly to the target. The SEA* method navigates to the target, because the length of the path to the prediction is smaller than the length of the target to the prediction. Also they are on the same quadrant of the prediction. Both methods take the same path, overlapping paths in the figure, and reach the target in 11 steps.

The second position of the user is a position where the target is approached from the left (figure 4.17). In this situation the SEA* method is navigating to the prediction and reaches the target in 39 steps. This is slightly faster than the iterative A* algorithm, which reaches the target in 40 steps. First both paths are parallel approaching the target, then the SEA* method navigates to the prediction, the exit door, while the iterative A* method exactly follows the target.

The third position of the user is a position where the target is approached from the right (figure 4.18). Because the prediction and the target are both in the same direction, both methods behave the same. Both methods reach the target in 53 steps.

All these locations of the user have (almost) equal behavior and both methods determine a path of the same length. Both methods are reaching the target in all these situations, which means that both methods are effective.

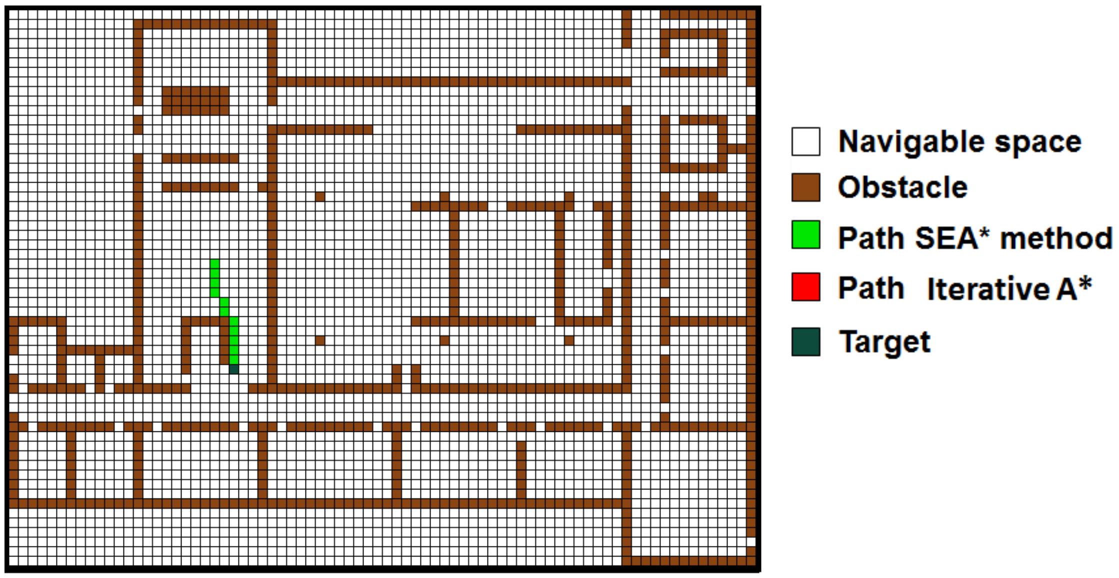

Figure 4.16, result scenario 2 where the user is between the target and the destination of the target. SEA: 11 steps & iterative A*: 11 steps.*

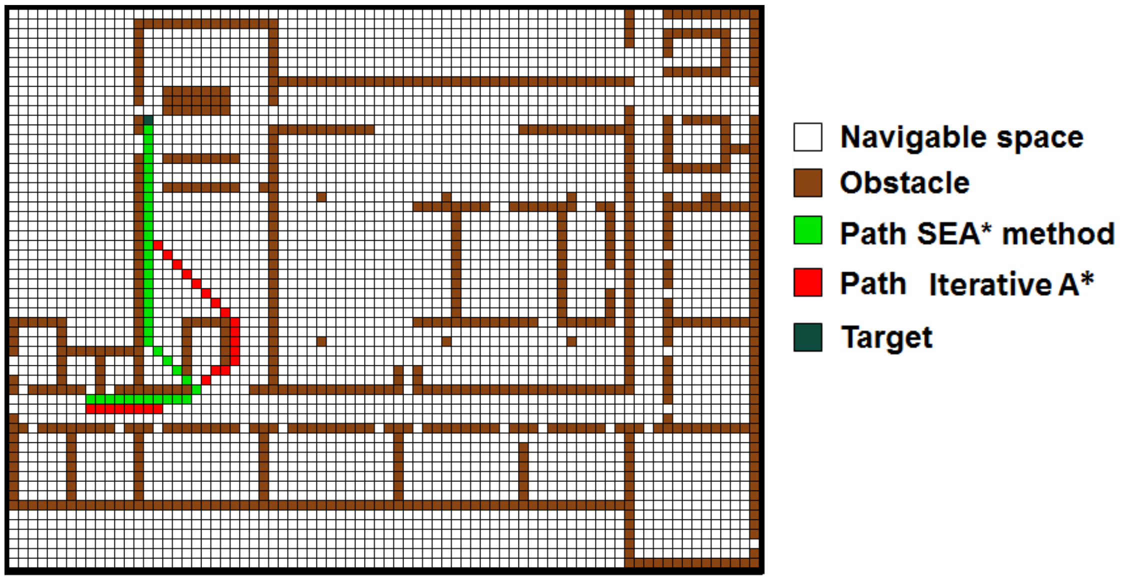

Figure 4.17, result scenario 2 where the user is approaching the path of the target from the left. SEA: 39 steps & iterative A*: 40 steps.*

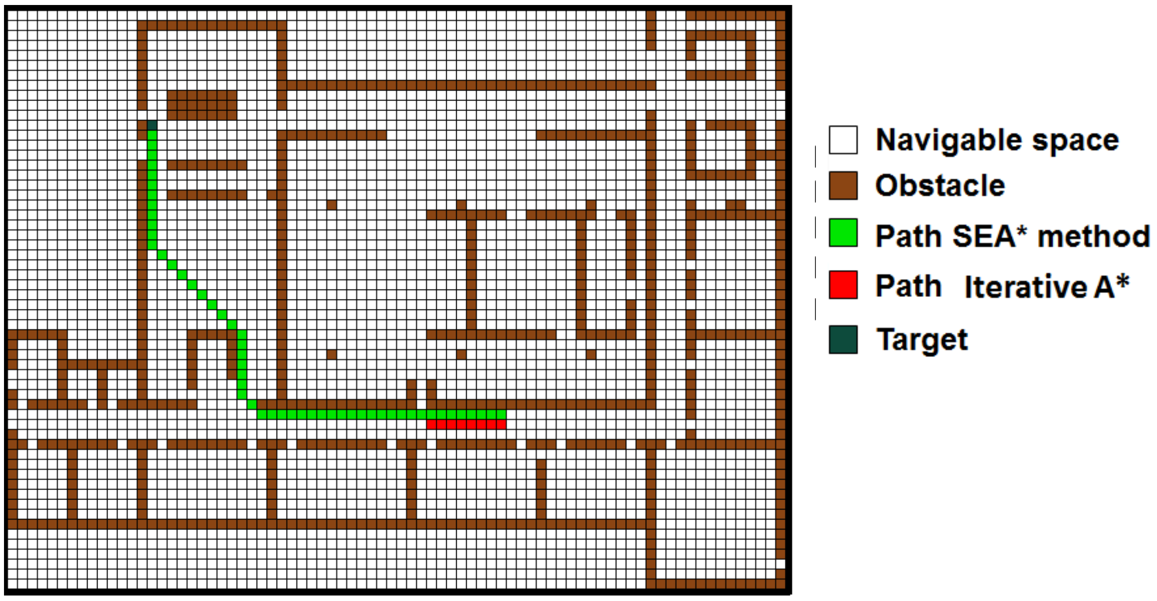

Figure 4.18, result scenario 2 where the user is approaching the path of the target from the right. SEA: 53 steps & iterative A*: 53 steps.*

4.2.3 Scenario 3: Student has a lecture on the first floor.

The third scenario is a student that is in a lecture room and his next lecture is on the first floor. The lecture room the student is currently in is on the ground floor. The student navigates therefore to one of the staircases to reach the first floor. This initialization is shown in figure 4.19. The user of the system is in this case getting a coffee in the corridor on the left. In the preprocessing phase it is possible to obtain the schedule of the student the user wants to follow. If the user knows this student first has a lecture on the ground floor and afterwards a lecture on the first floor, staircases and the elevator are assigned a higher weight than the other points of interest.

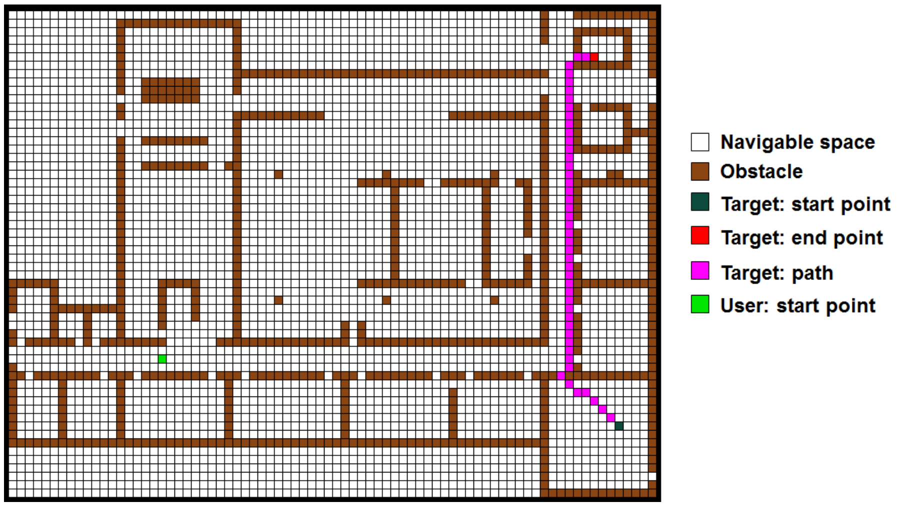

Figure 4.19, scenario 3 initialization of the target and the user.

Therefore the elevator and the staircases are the prediction points if the target is moving in these directions. The result of both methods are shown in figure 4.20. The SEA* method first navigates towards the target and then uses the staircase as the prediction point. This is because the target is moving towards the staircase and the elevator in the top right corner. The SEA* method reaches the target in 69 steps and takes a shortcut compared to the iterative A* implementation, that reaches the target in 81 steps.

In a situation where there are assigned no weights, the user is using the closest point of interest of the target as the prediction point. The result of this scenario without assigning weights is shown in figure 4.21. Both methods have the same behavior and reach the target in 81 steps. However, the SEA* method navigates to predictions close by the target and the iterative A* method navigates to the target directly. Because the weights of the points of interest are all equal no big shortcuts can be made.

Now what happens if the same weights are used, but the target first wants a coffee and navigates to the closest coffee machine? The results are depicted in figures 4.22 and 4.23. Figure 4.22 shows the situation with higher weights for the stairs and the elevator. The SEA* method takes a shortcut, but this shortcut was not totally correct. Therefore the SEA* method does not perform optimal and reaches the target in 74 steps, while the iterative A* method reaches the target in 69 steps. This proves that taking a big shortcut is not always the best option. Figure 4.23 is the result of navigating to the coffee machine, with equal weights for all points of interest. Both methods behave the same and reach the target in 69 steps. In this specific situation it is not always better to take a shortcut. The prediction must be known in advance and if the target is navigating to this prediction the SEA* method is always better. However, navigating to a slightly different point of interest, like the coffee machine, can give undesirable behavior. In both scenarios both methods always reach the target.

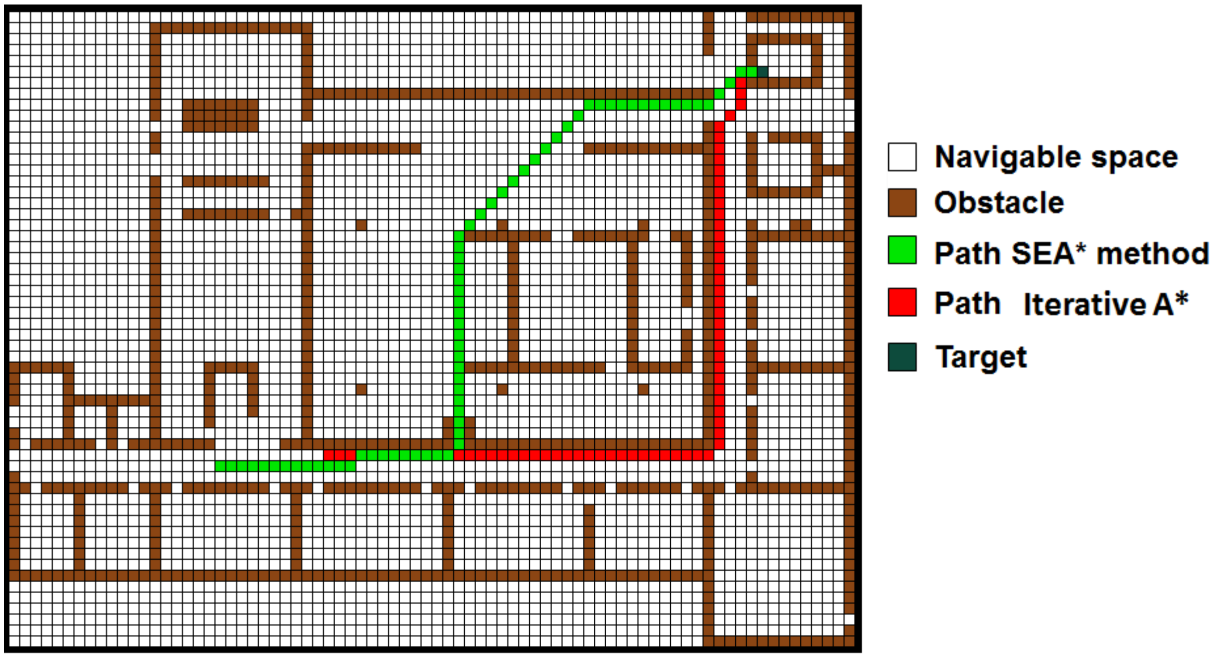

Figure 4.20, result scenario 3 with higher weights for the stairs and elevator. SEA: 69 steps & iterative A*: 81 steps.*

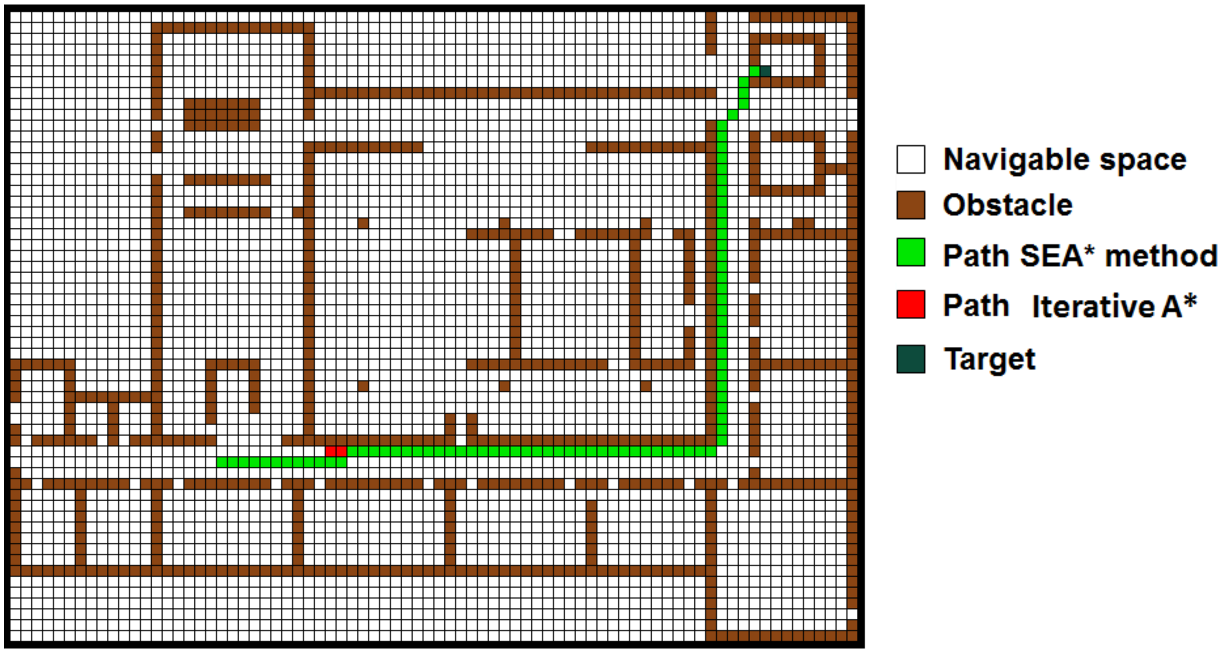

Figure 4.21, result scenario 3 with the same weights for all points of interest. SEA: 81 steps & iterative A*: 81 steps.*

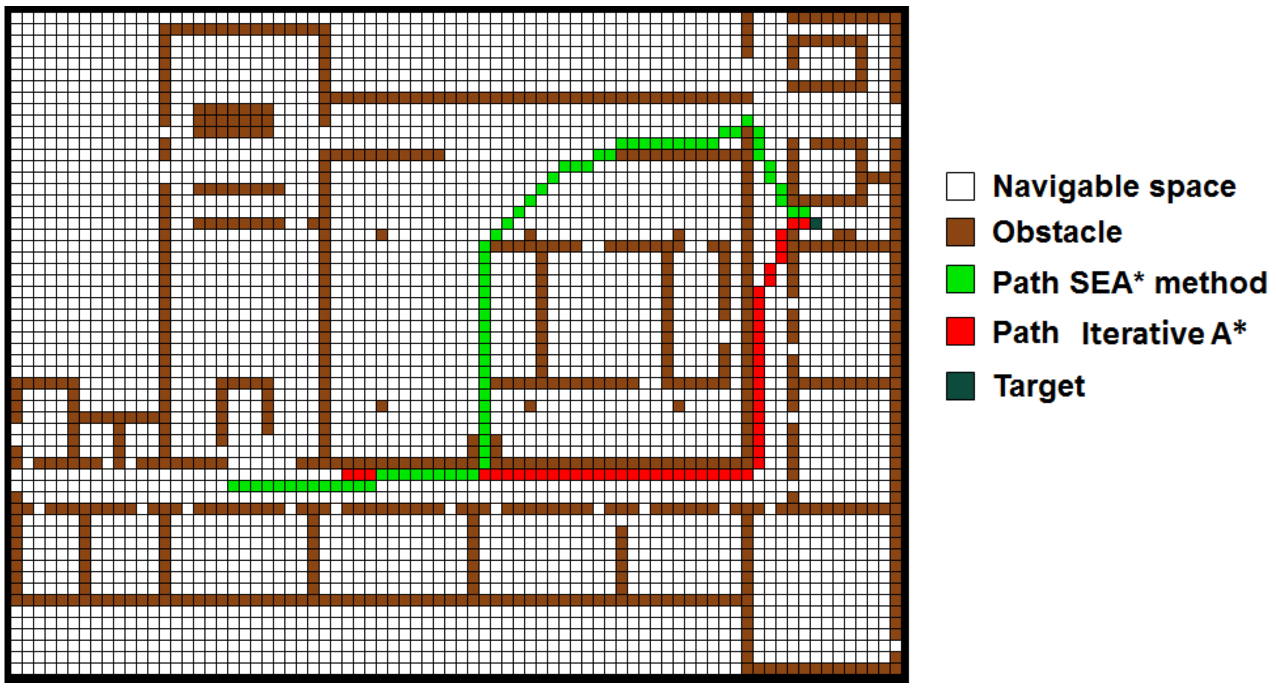

Figure 4.22, result scenario 3, navigating to the coffee machine, with higher weights for the stairs and elevator. SEA: 74 steps & iterative A*: 69 steps.*

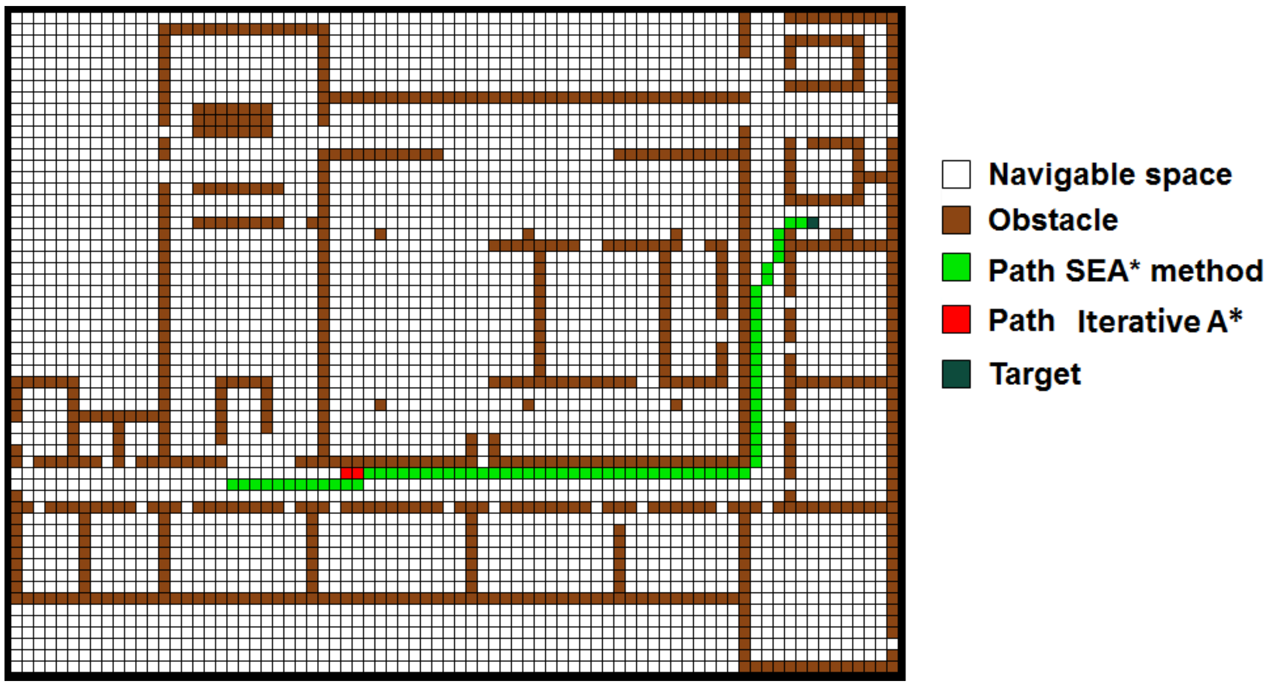

Figure 4.23, result scenario 3, navigating to the coffee machine, with the same weights for all points of interest. SEA: 69 steps & iterative A*: 69 steps.*

4.2.4 Scenario 4: Visitors visit an event

The fourth scenario analyses the models in case of an event. An event is held in the lobby, which is located around the point of interest that represents the event. The event is known in advance and a visitor that want to visit the event and has no further points of interest is likely to navigate to the event. In this case the weight of the point of interest that represents the location of the event is higher than all other points of interest, the other points of interest are all equal. The user wants to navigate to this visitor of the event. This situation is initialized as shown in figure 4.24.

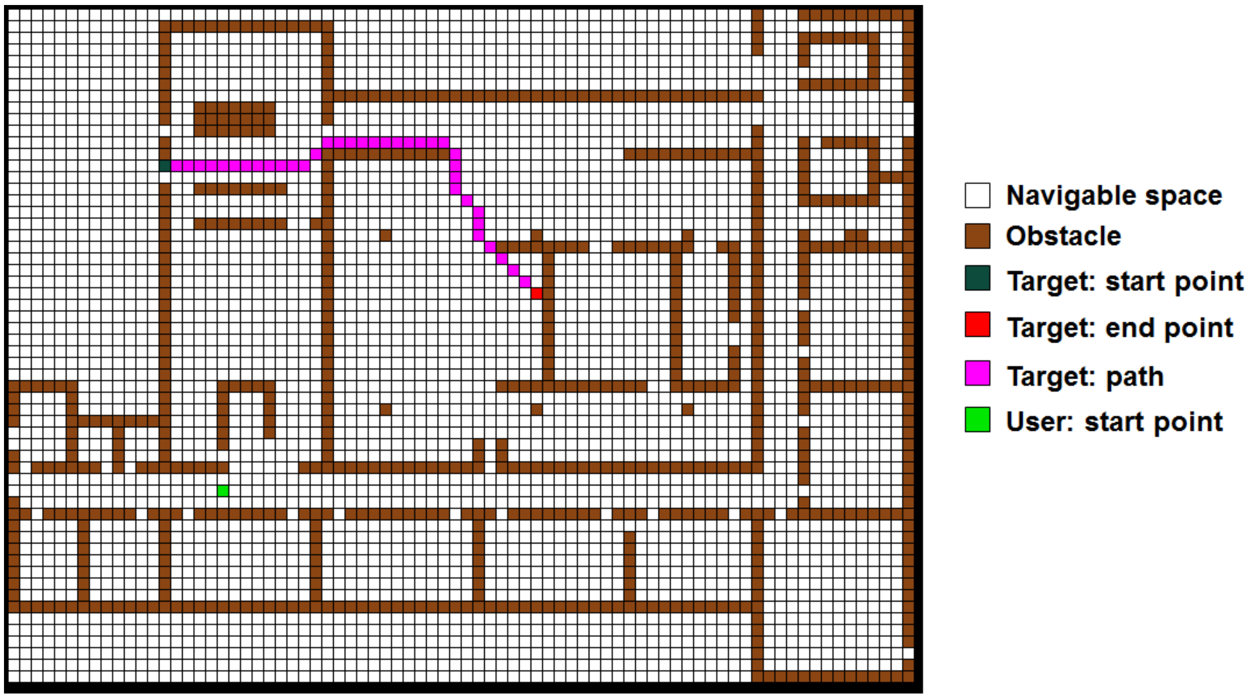

Figure 4.24, scenario 4 initialization of the target and the user.

The point of interest, that represents the location of the event, is used to the prediction point if the target is navigating in this direction. Figure 4.25 shows the result when the event has a higher weight than the other points of interest. The SEA* method first navigates to the target directly, but when the target moves towards the event, the user navigates to the prediction of the target. In this case the user reaches the target in 44 steps. The iterative A* method follows the target and reaches the target in 51 steps. Navigating to the prediction is in this case faster than navigating directly to the target.

Figure 4.26 shows the result of the same scenario, but now uses equal weights for all the points of interest. First the SEA* method navigates towards the points of interest close by the target and then navigates to the event point, which is the closest point of interest at that moment in time. The target is reached in 45 steps, while the iterative A* algorithm reaches the target again in 51 steps. In both situations the targets are reached for all methods. In this case there is (almost) no difference in the behavior of the situation with weights and the situation without weights.

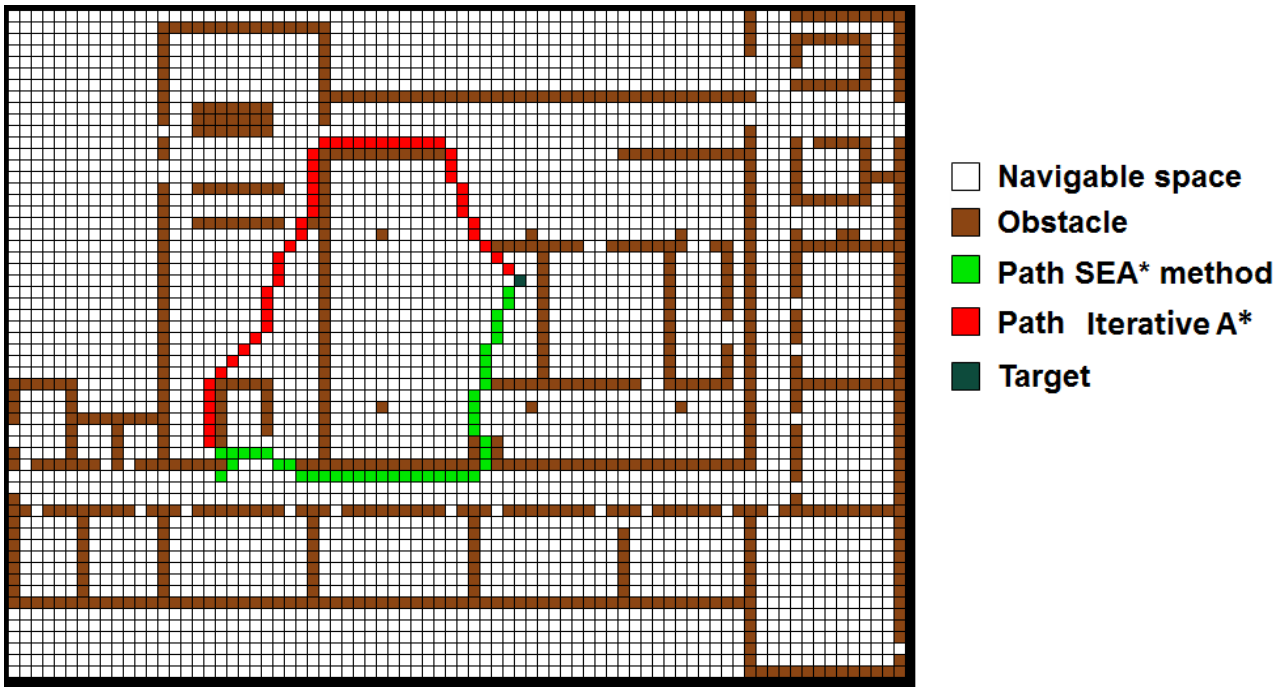

Figure 4.25, result scenario 4 with a higher weight for the event. SEA: 44 steps & iterative A*: 51 steps.*

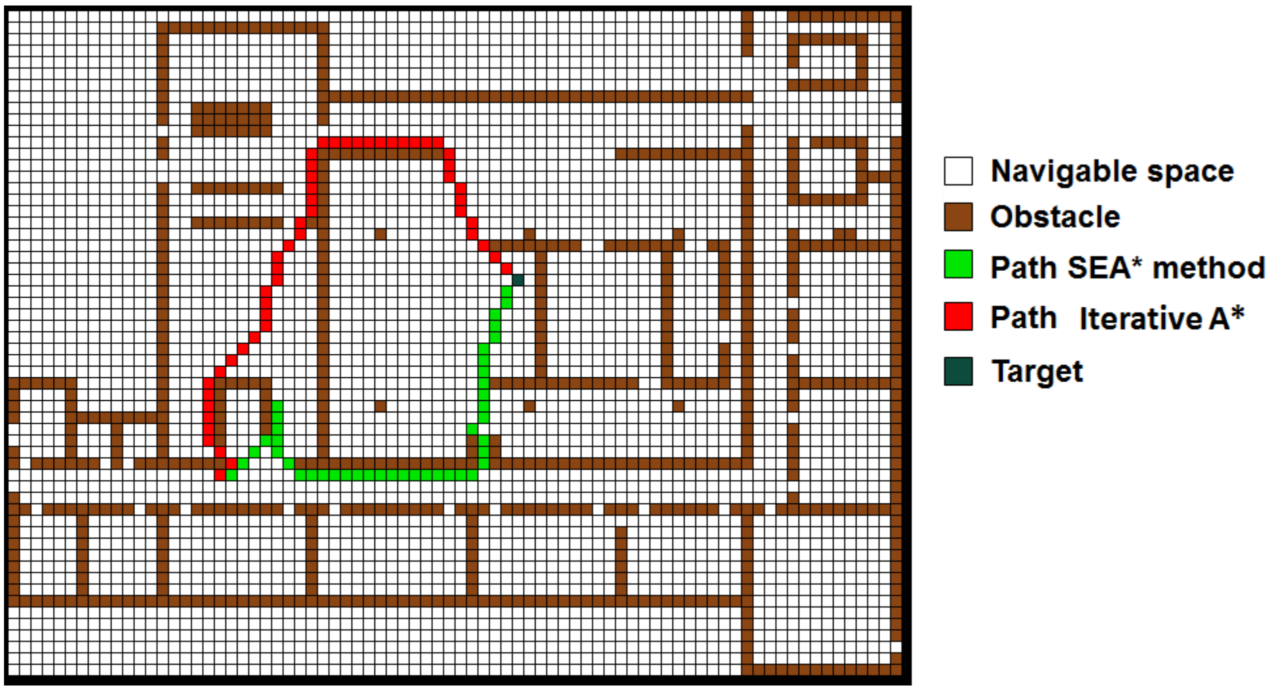

Figure 4.26, result scenario 4 with same weights for all points of interest. SEA: 45 steps & iterative A*: 51 steps.*

4.3 Overview of the results

This section summarizes all the results of the scenarios described in section 4.2. There are four scenarios that discusses several cases and that are all based on real behavior of the user and the target. Each scenario and its sub scenarios are presented in table 4.2. This table shows that, if the user gives the highest weight to a point of interest in the neighborhood of the end destination of the target, the SEA* method always gives a faster path than the iterative A* method. However, when the highest weight is given to another point, the SEA* method gives the same or a slower path than the iterative A* method. Therefore assigning points of interest and its weights must only be used when the user is certain about where the target is moving to.

Table 4.2, summary of the results presented in section 4.2.

Scenario	Dominant weights	User start point	Target start point	Target end point	SEA* [steps]	Iterative A*[steps]
1. Break during a lecture	Coffee machines	Study room	Corridor (east)	Coffee machine	57	96
1. Break during a lecture	Geolab	Study room	Corridor (east)	Coffee machine	96	96
1. Break during a lecture	All equal	Study room	Corridor (east)	Coffee machine	67	96
2. Closing the building	Possible exits	Lobby	Geolab	North-west exit	37	74
2. Closing the building	All equal	Lobby	Geolab	North-west exit	50	74
2. Closing the building	Possible exits	Corridor	Geolab	North-west exit	11	11
2. Closing the building	Possible exits	Corridor (left)	Geolab	North-west exit	39	40
2. Closing the building	Possible exits	Corridor (right)	Geolab	North-west exit	53	53
3. lecture on first floor	Stairs and elevator	Corridor (left)	Lecture room	Stairs	69	81
3. lecture on first floor	All equal	Corridor (left)	Lecture room	Stairs	81	81
3. lecture on first floor	Stairs and elevator	Corridor (left)	Lecture room	Coffee machine	74	69
3. lecture on first floor	All equal	Corridor (left)	Lecture room	Coffee machine	69	69
4. Event	Event	Corridor (left)	North-west exit	Event	44	51
4. Event	All equal	Corridor (left)	North-west exit	Event	45	51

In the case that all the points of interest have the same weight, the nearest point of interest of the target is used as the prediction. These scenarios provide in most cases a faster path than the iterative A* method. It is also possible that the same path is obtained. Therefore when the user has no clue about where the target is navigating to, it is better to use all points of interest with the same weight and use the closest point of interest to the target as the prediction point to navigate to.

Eventually there are cases that are not presented in section 4.2, because they are hard to visualize, but are important to notice. These scenarios are hard to visualize because there are a lot of overlapping paths. The first case is, when a target is walking in a certain direction and suddenly turns around and moves in the other direction. In this case the iterative A* method returns a faster path if the SEA* method already tries to take a shortcut towards the prediction of the target. When all weights of the points of interest are equal, this effect is smaller. The second case is, when a target is moving and does not stop moving. In practice this is unlikely, but it could occur. Then the SEA* method is preferred, because this method reaches the target if the shortcut is towards where the target is actually moving. The iterative A* method will follow the target. If the target is walking around the big lobby, in the middle of the model, this method will never reach the target. The SEA* method will eventually reach the target. The SEA* method finds the target faster if the prediction points are correctly defined, but also with equal weights for all points of interest this method uses shortcuts to finally navigate to the target.

4.4 Discussion of the requirements

In this section the criteria defined in section 3.1.1 are used to discuss the results presented in this chapter. The validation criteria are spatially logical, effective and efficient. Spatially logical means that the model is correctly representing the reality. The method is effective if the method always finds its target. The method is also effective if the target is reached as fast as possible, preferable faster than the iterative A* method. The method is efficient if the user gets its directions as fast as possible.

Spatially logical: The spatial model that represents the terrain is developed manually, therefore this model is spatially logical. The doors represent the connections between the different rooms, corridors, lobby and stairs. Every connection that is there in reality is also present in the spatial model. However, there is a slightly overrepresentation of the obstacles, but this does not influence the behavior of the navigation.

Effective: The SEA* method is implemented and compared to the iterative A* method with using a real floorplan of the faculty of architecture in Delft. The SEA* method will eventually always find the target, in contrast to the iterative A* method, where a target that keeps moving could never be reached. Therefore the SEA* method is more effective than the iterative A* method. Next to this factor, the SEA* method delivers a faster path to the user if the prediction of the target is correctly assigned. If the user assigns a higher weight to a point of interest where the target is not really navigating to, the iterative A* algorithm could be slightly better. This could be solved by using no weights, but the distance to the points of interest to predict where the target is going to. This outperforms the iterative A* method, but is not always as fast as when the prediction is assigned correctly.

Efficient: It depends on the situation which of the two methods is faster in giving directions to the user. When the target is closer to the user, the iterative A* algorithm is most of the times faster, while when the target is farther away from the user, the SEA* method is most of the times faster. In a real case scenario this criteria is more important, because the user is not waiting five minutes while the target is in the same building.

5. Implementation: Outdoor environment

This section describes the implementation of the methodology presented in chapter 3 of this thesis research within an outdoor environment. The Global Positioning System is used to determine both the location of the user and the position of the target. The spatial model that is used for the navigation is obtained from Google Maps, making use of the Google Maps API. The Google Maps Distance Matrix API is used to determine the distance from one point to another point. Further the Google Maps Directions API is used to obtain the fastest route from one point to another point and visualizing the route. To deal with these APIs the Google Maps JavaScript API V3 (Google, 2016) is used. Therefore this implementation is written in JavaScript, HTML and CSS. Eventually this implementation could be converted to a mobile application for Android.

Section 5.1 discusses the pre-processing steps and explains how the SEA* method is implemented. The section 5.2 shows the results of the scenario analysis in the outdoor environment. An overview of the results is given in section 5.3. Finally section 5.4 discusses the requirements that are determined in chapter 3.

5.1 Pre-processing

This section first discusses what spatial model is used, 5.1.1. Then in section 5.1.2 the points of interest are defined. Section 5.1.3 shows the data input that is used for the positioning of the target. Section 5.1.4 explains how the prediction of the target is implemented and determined and finally the used path planning algorithm is discussed in section 5.1.5.

5.1.1 The spatial model

The test area used is the campus of the TU Delft, which is depicted in figure 5.1. The Google Maps API offers a detailed map for navigating using direct GPS data or addresses. It uses a road network model, where the nodes represent waypoints and the edges are the connections between these edges if navigation is possible according to a preferred mode of transport. This API offers a few advantages and disadvantages.

Advantages:

- It is possible to add points of interest to the model, using GPS locations or addresses.
- It provides a way to determine the fastest route from one point to another point.
- The API covers most countries and could be used anywhere, where Google Maps is available.

- It is possible to determine the distance between two locations.
- It is possible to visualize the map and the route the user has to take.
- The API is very efficient and effective in finding the fastest route.
- The API supports routing for walking, cycling, driving and public transport.

Disadvantages:

- Paths that are not marked as navigable in Google Maps are not used to calculate the fastest route. There are situations where there is a sidewalk, but this is not modelled as such. Therefore the model limits the usability in areas that are not frequently mapped. This however should not be a problem in large cities.
- Locations are not always located at the exact same position, because the coordinates are mapped to the nearest node in the spatial model.

Each location is mapped to the nearest node in the Google Maps model. This has both its advantages and its disadvantages. GPS measurements that are not located at a node, but for example in the water (see figure 5.2), are mapped to the nearest node in the model. Most of the times these represent the correct location of the point, however this is not always the case. Figure 5.3 shows a situation where the GPS location of a building is mapped to the nearest node. This is an important notion to add points of interest to the model, the user has to check the exact points of interest used for the navigation.

The topology used to navigate from one point to another is obtained by accessing the Google Maps API obtaining the route by using the walking parameter. This means that only nodes where a person can walk are used to navigate from one point to another.

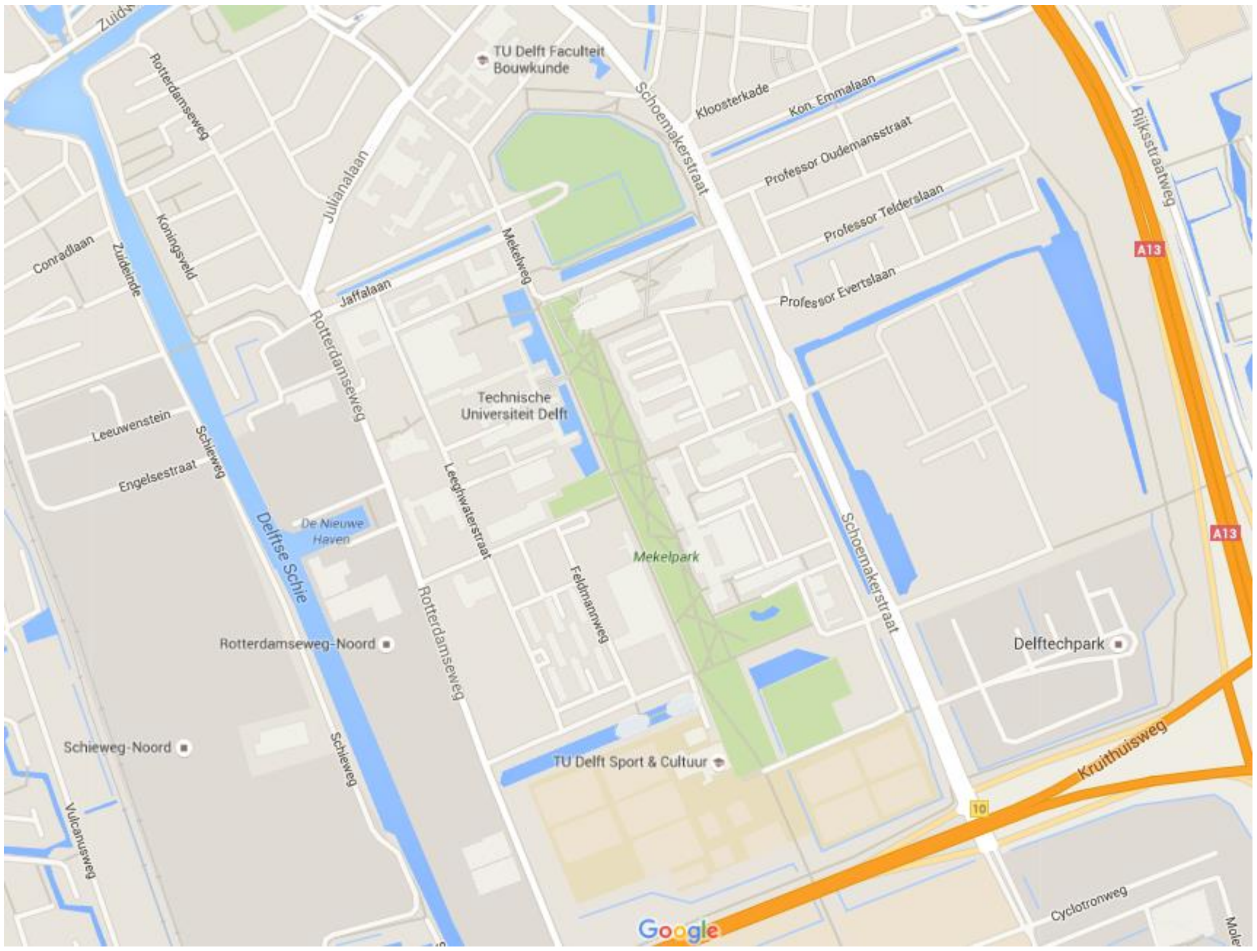

Figure 5.1, campus TU Delft as test area.

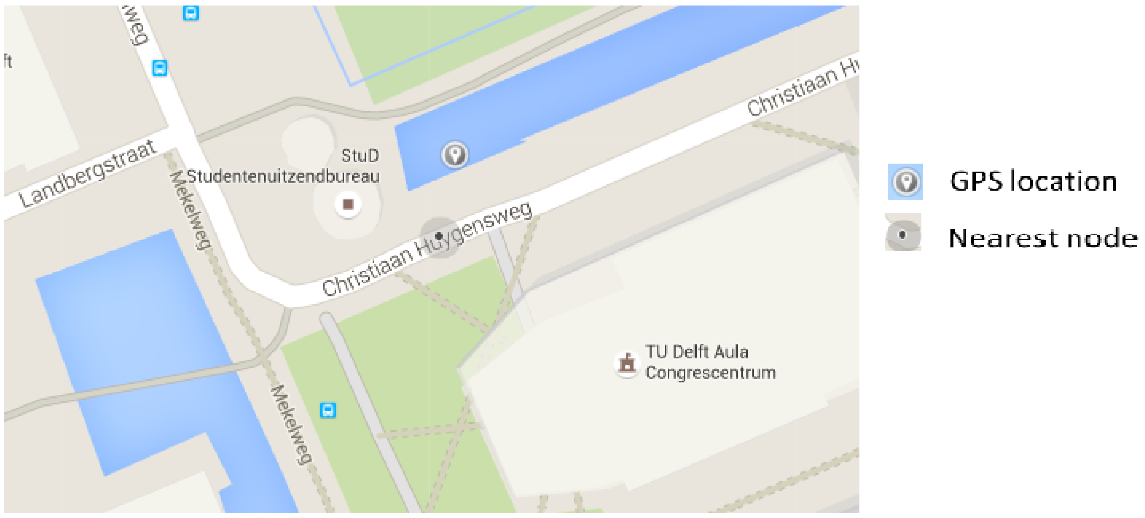

Figure 5.2, mapping GPS location (in the water) to the nearest node.

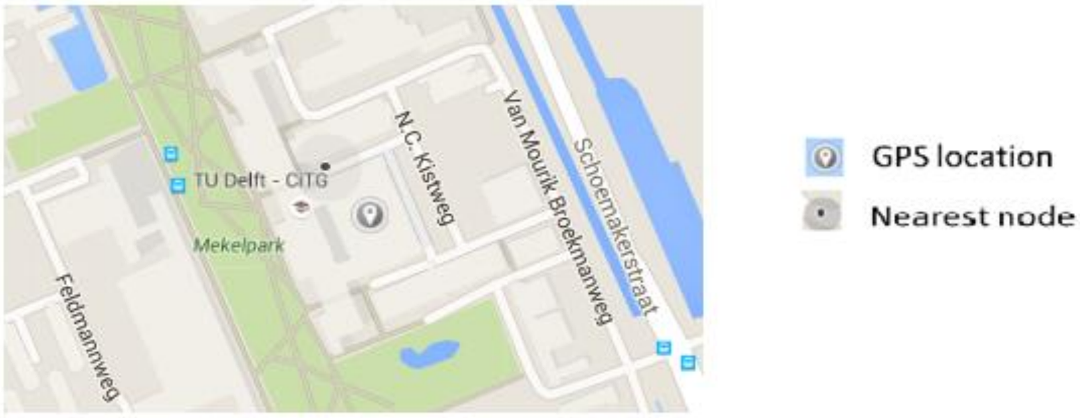

Figure 5.3, mapping GPS location (in a building) to the nearest node.

5.1.2 Adding the points of interest

To predict where the target is moving to, points of interest should be identified. A point of interest could be any location that is interesting for the target to navigate to. The entrances of buildings are important locations, where a person could enter a building, therefore these are used as points of interest. Of course not every building is interesting for every person, therefore only the interesting buildings should be used as points of interest for a specific person. Also a park, a forest, a meeting place could be used as points of interest. Eventually routes to for example a city center could be used to represent the points of interest which are farther away than the test area. In this example such points are not used. The latitude and longitude GPS coordinates are used to represent the locations of the point of interest.

The points of interest are defined as the following and are depicted in figure 5.4:

1. The faculty of Architecture and the Build Environment, represented by the three main entrances.
2. The aula, represented by the two entrances.
3. The library, represented by the entrance.
4. The house of the target, represented by a point nearby.
5. The daycare, represented by the coordinates of the building.
6. The hockey club, represented by the coordinates of the clubhouse.
7. The faculty of Civil Engineering & Geosciences, represented by the central entrance.
8. The faculty of Electrical Engineering, Mathematics and Computer Science, represented by the central entrance.
9. The faculty of Industrial Design Engineering, represented by the central entrance.
10. The faculty of Applied Sciences, represented by the central entrance.
11. The faculty of Mechanical, Maritime and Materials Engineering, represented by the central entrance.

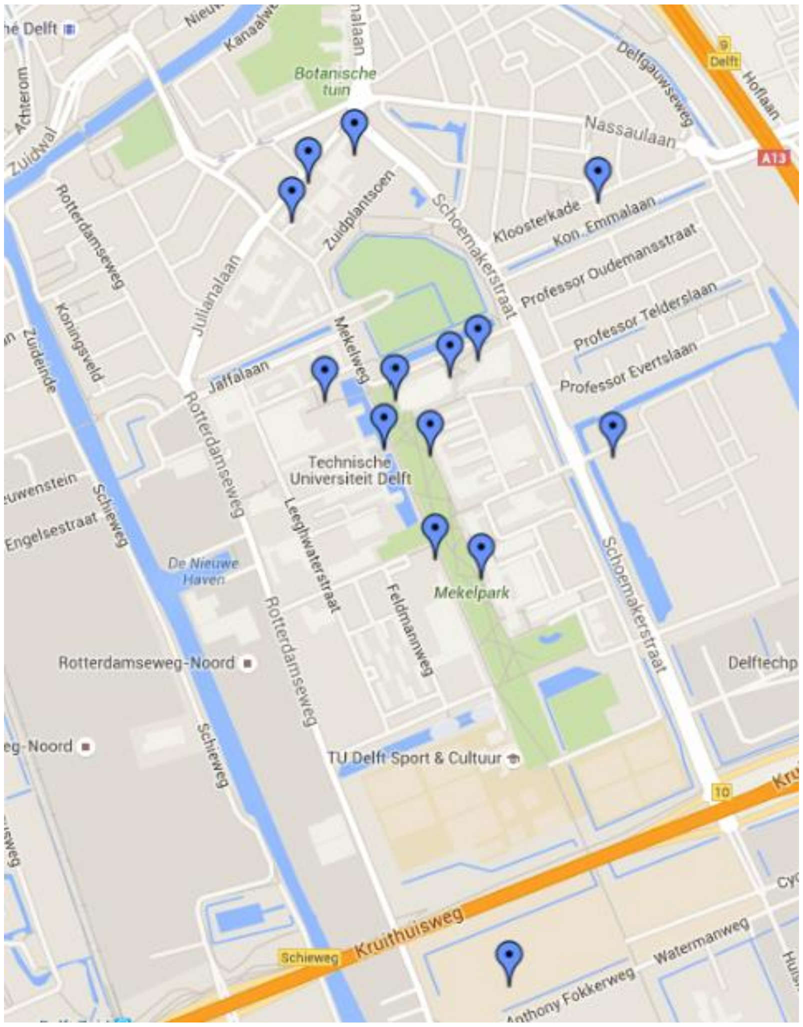

Figure 5.4, points of interest in the outdoor environment.

5.1.3 The positioning of the target

For the position of the target a GPS log is used which is obtained in and around the campus of the University of Delft. The data consists out of a position, in latitude and longitude, per second. This data is reduced to a position for every 10 seconds, which is shown in figure 5.5. Therefore the implementation calculates the path that the user should take every 10 seconds.

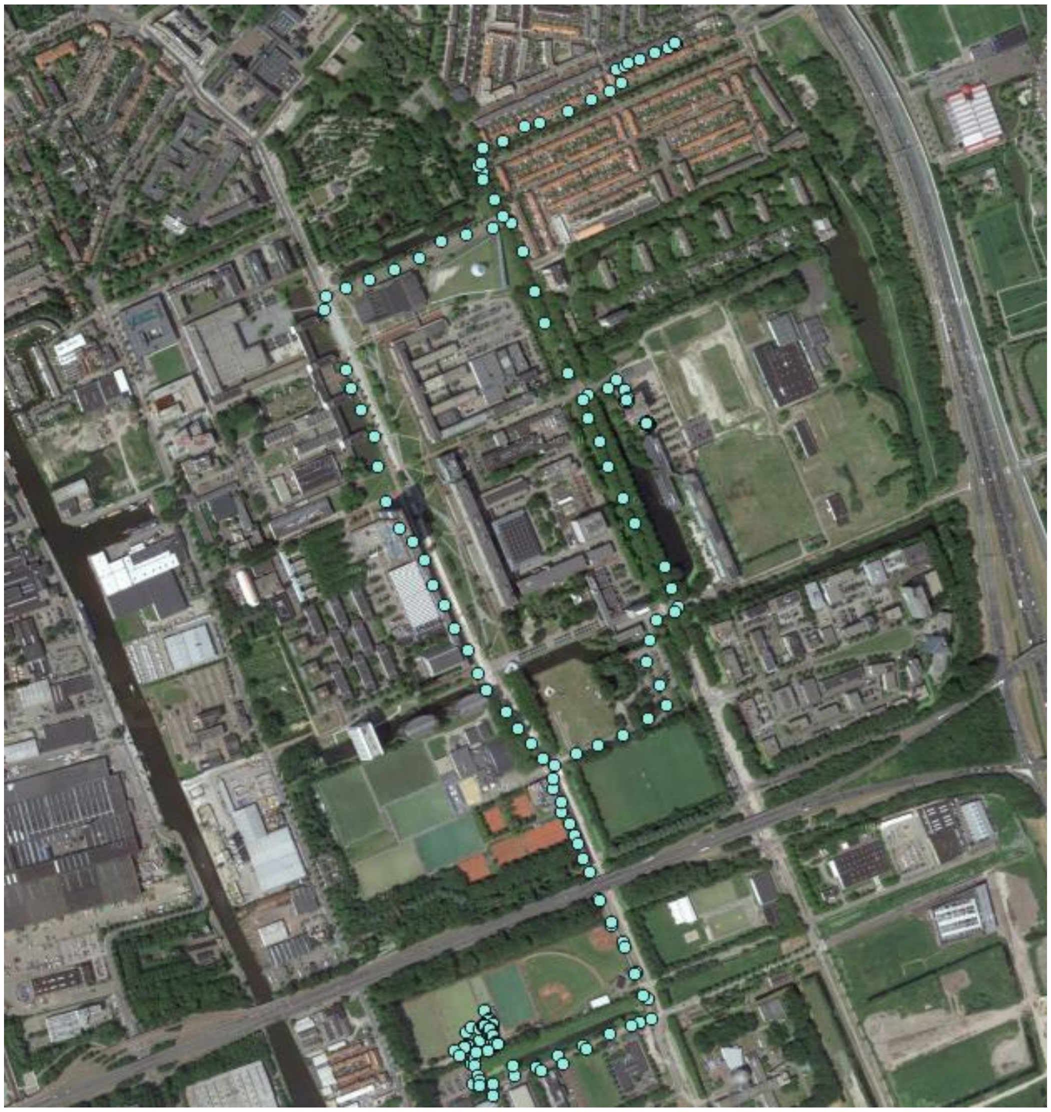

Figure 5.5, GPS log campus TU Delft per 10 seconds.

5.1.4 The prediction of the target

To determine to which points of interest the target is navigating, the prediction of the target, the Google Maps Distance Matrix is used to determine which points are approached by the target. Every time when there is a new measurement the distance to each of the points of interest is calculated. This is the distance of the fastest path from the target to each point of interest. When the distance is smaller than the distance from the previous measurement, the target is approaching this point of interest. All the points of interest that are approached by the target are stored. Then the closest point of these points of interest compared to the target is used as the prediction of the target. This was according to the indoor implementation the most promising option.

5.1.5 Path planning algorithm

The determination whether to navigate to the prediction or directly to the target is according the algorithms 3.7 and 3.8, as presented in chapter 3. The path planning algorithm used is the Google Maps Directions API, which uses a starting point and a destination point and a mode of navigation, which is walking. The Google Maps Directions API provides the fastest path from the starting point to the end point. The starting point is here the position of the user, where the end point is the current location of the target or the prediction of the target.

5.2 Scenario analysis

This section describes several scenarios based on the GPS log as positioning of the target. Each scenario has a different starting point of the user to navigate to the target. The results per scenario are given, this means the path the user takes to reach the target. This path is per scenario compared to the normal implementation of the iterative A* algorithm. Sometimes the methods have overlapping paths, then both methods have the same behavior and one result is shown.

5.2.1 Scenario 1: Target is moving, but stops in between.

For this scenario the first part of the GPS log is used. The duration of this log is 9 minutes and 30 seconds on a bicycle. This means that the user must find the target within this timeframe using a bicycle. The implementation uses walking as the mode for travel. Therefore the target must be found in 16 minutes by walking, obtained by using Google maps and drawing the GPS log. This part of the GPS log is illustrated in figure 5.6. The position of the target differs every time.

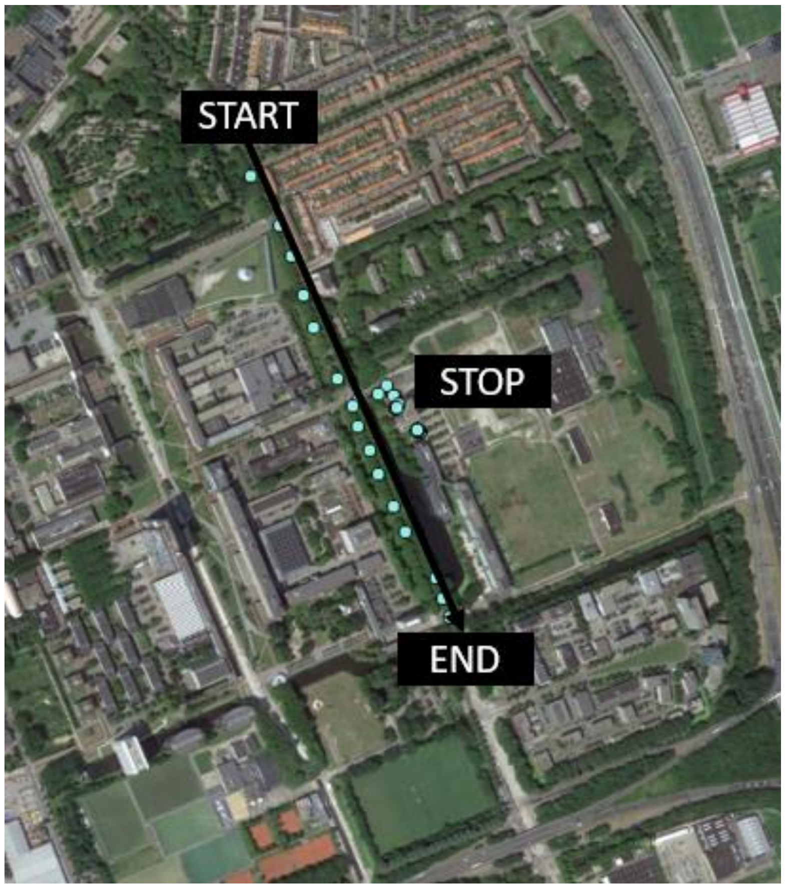

Figure 5.6, GPS log data, where the target starts at the start point, navigates to the stop point and then continues to the end point.

There are 3 different starting positions used for the user, figure 5.7:

1. In the Mekelpark, at the other side of and closer by the point of interest.
2. In the Mekelpark, at the same side of and closer by the point of interest.
3. Christiaan Huygensweg, near the TU Delft aula, farther from the point of interest than the target.

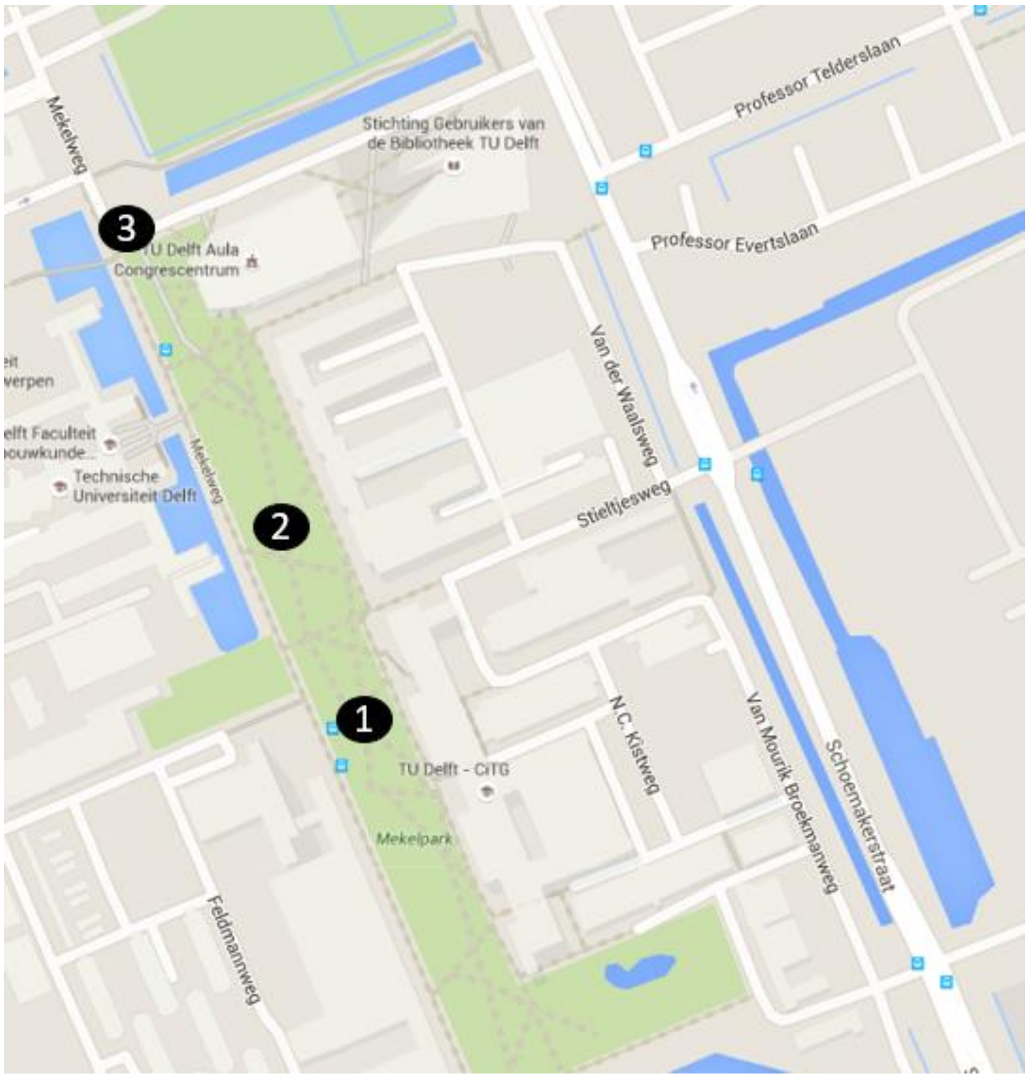

Figure 5.7, starting positions of the user for scenario 1.

For the first starting position the result is shown in figure 5.8. Both the implemented methods behave the same and reach the target at the Van Embdenstraat.

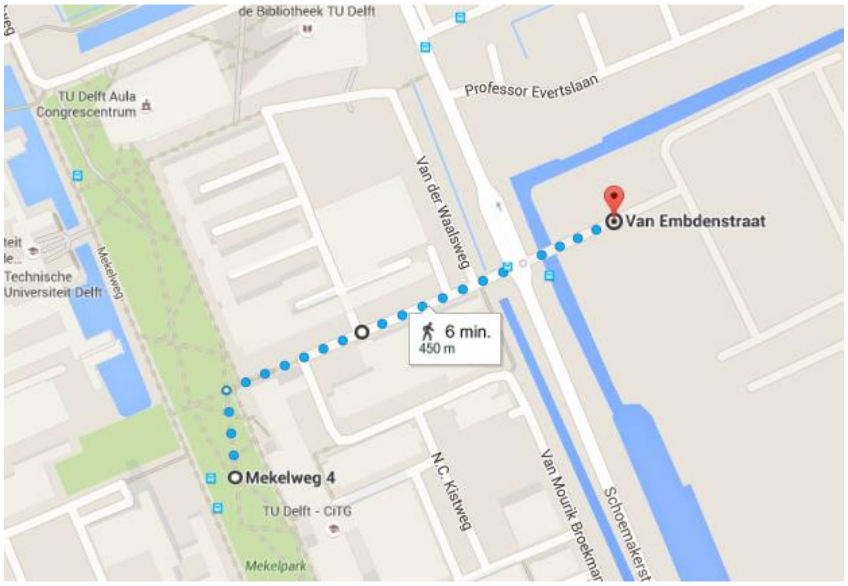

Figure 5.8, 1st position: path for the SEA method and the iterative A* method, reaching the target in 6 minutes (450 meter).*

The results of the second starting position of the user are shown in figure 5.9, the SEA* method, and figure 5.10, the iterative A* method. The SEA* method reaches the target at the Schoenmakerstraat in 4 minutes, which is faster than the iterative A* method which reaches the target at the Van Embdenstraat in 9 minutes.

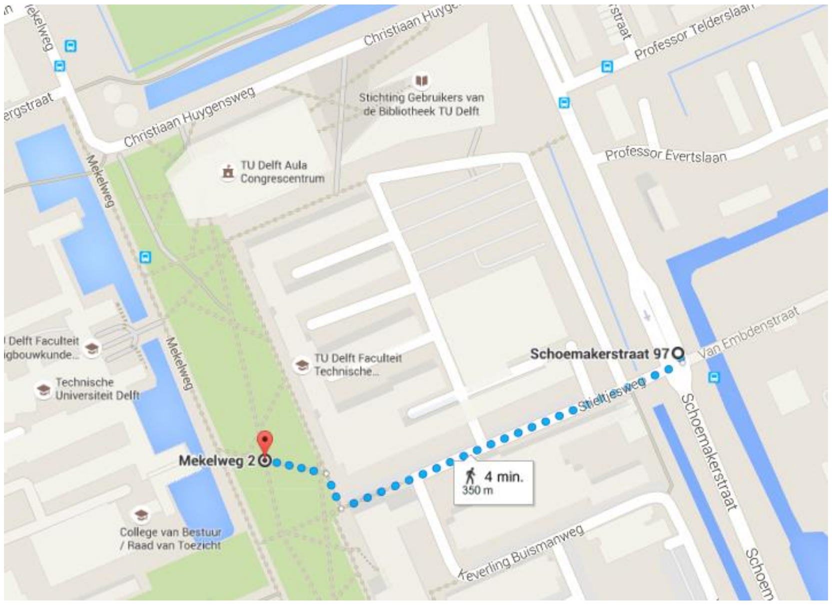
Christiaan Huygensweg
Stichting Gebruikers van de Bibliotheek TU Delft
Professor Telderslaan
Professor Evertslaan
TU Delft Aula Congrescentrum
Mekelweg
TU Delft Faculteit Technische...
Schoemakerstraat 97
Van Embdenstraat
Stieltjesweg
Technische Universiteit Delft
Mekelweg 2
4 min.
350 m
Schoemakerstraat
College van Bestuur / Raad van Toezicht
Feverling Buismanweg

Figure 5.9, 2nd position: path for the SEA method, reaching the target in 4 minutes (350 meter).*

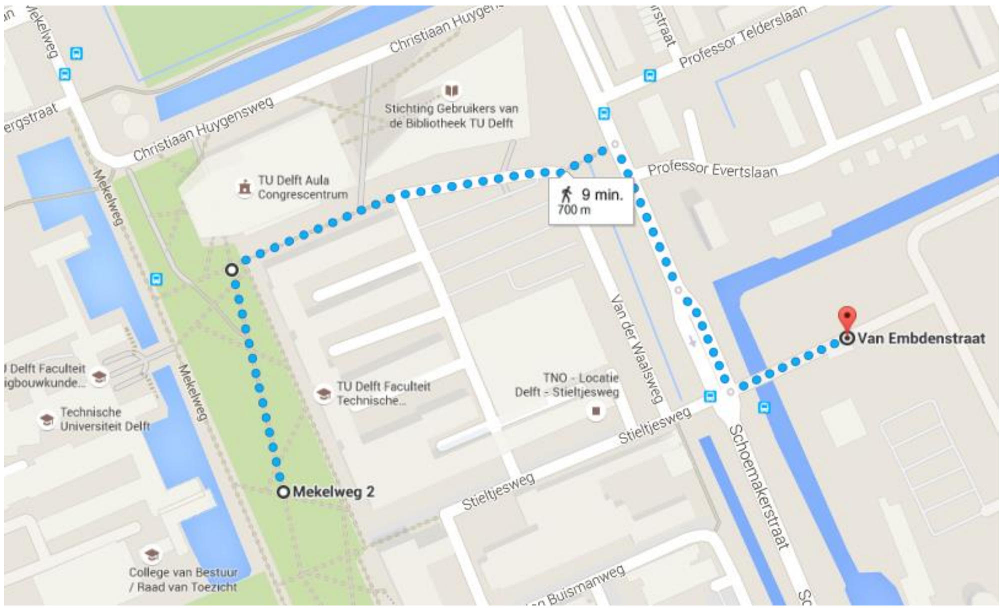
9 min.
700 m
Mekelweg 2
Van Embdenstraat
TU Delft Aula Congrescentrum
Christiaan Huygensweg
Professor Evertslaan
Professor Telderslaan
Van der Waalsweg
Stieltjesweg
Schoemakerstraat
Mekelweg
TNO - Locatie Delft - Stieltjesweg
TU Delft Faculteit Technische...
Technische Universiteit Delft
College van Bestuur / Raad van Toezicht
Stichting Gebruikers van de Bibliotheek TU Delft

Figure 5.10, 2nd position: path for the iterative A method, reaching the target in 9 minutes (700 meter).*

The results of the third starting position of the user are shown in figure 5.11, the SEA* method, and figure 5.12, the iterative A* method. Both methods reach the target at the Van Embdenstraat in 8 minutes. This is because the target stops at this location for a while. The path that the SEA* method takes is slightly shorter, 650 meters, compared to the iterative A* method, 700 meter.

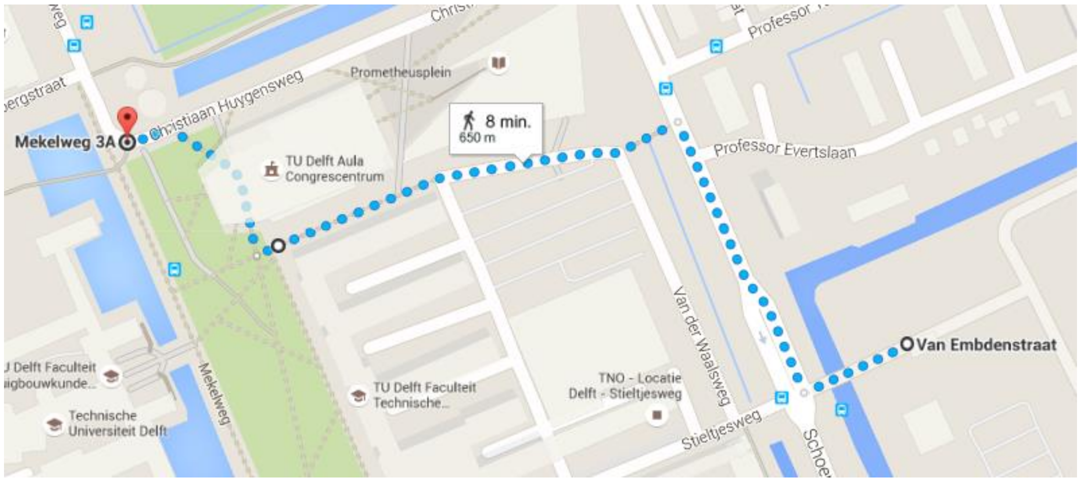

Figure 5.11, 3rd position: path for the SEA method, reaching the target in 8 minutes (650 meter).*

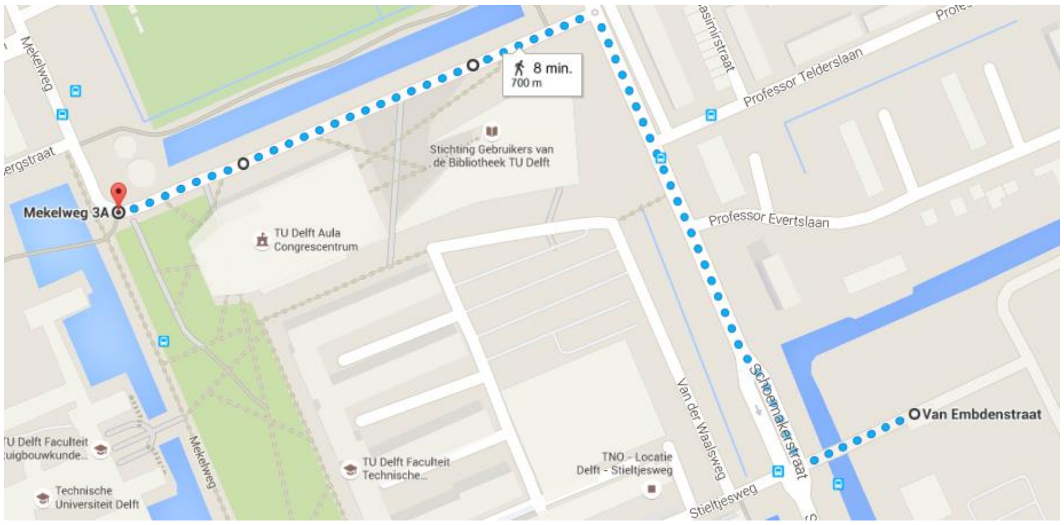
8 min.
700 m
Mekelweg 3A
Van Embdenstraat
Stichting Gebruikers van de Bibliotheek TU Delft
TU Delft Aula Congrescentrum
Professor Telderslaan
Professor Evertslaan
Schoemakerstraat
Van der Waalsweg
Stieltjesweg
TNO - Locatie Delft - Stieltjesweg
TU Delft Faculteit Technische...
Technische Universiteit Delft
Mekelweg

Figure 5.12, 3rd position: path for the iterative A method, reaching the target in 8 minutes (700 meter).*

5.2.2 Scenario 2: Target is moving, without stopping.

For this scenario a part of the GPS log is used. The duration of this log is 4 minutes and 20 seconds on a bicycle. This means that the user must find the target within this timeframe using a bicycle. The implementation uses walking as the mode for travel. Therefore the target must be found in 13 minutes by walking. This part of the GPS log is illustrated in figure 5.13. The position of the user differs every time.

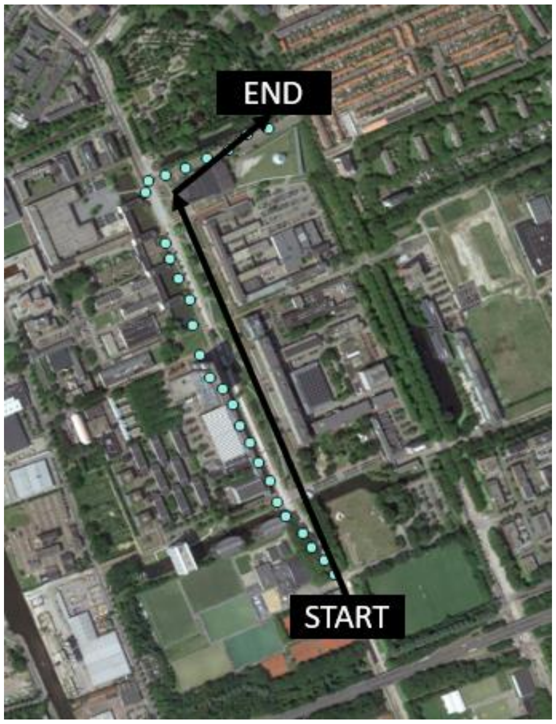

Figure 5.13, GPS log data, where the target starts at the start point and navigates to the end point.

There are 3 different starting positions for the user, figure 5.14:

1. At the library, at the opposite side approaching the target.
2. The Leeghwaterstraat, approaching the target from the left.
3. Pieter Calandweg, approaching the target from the right.

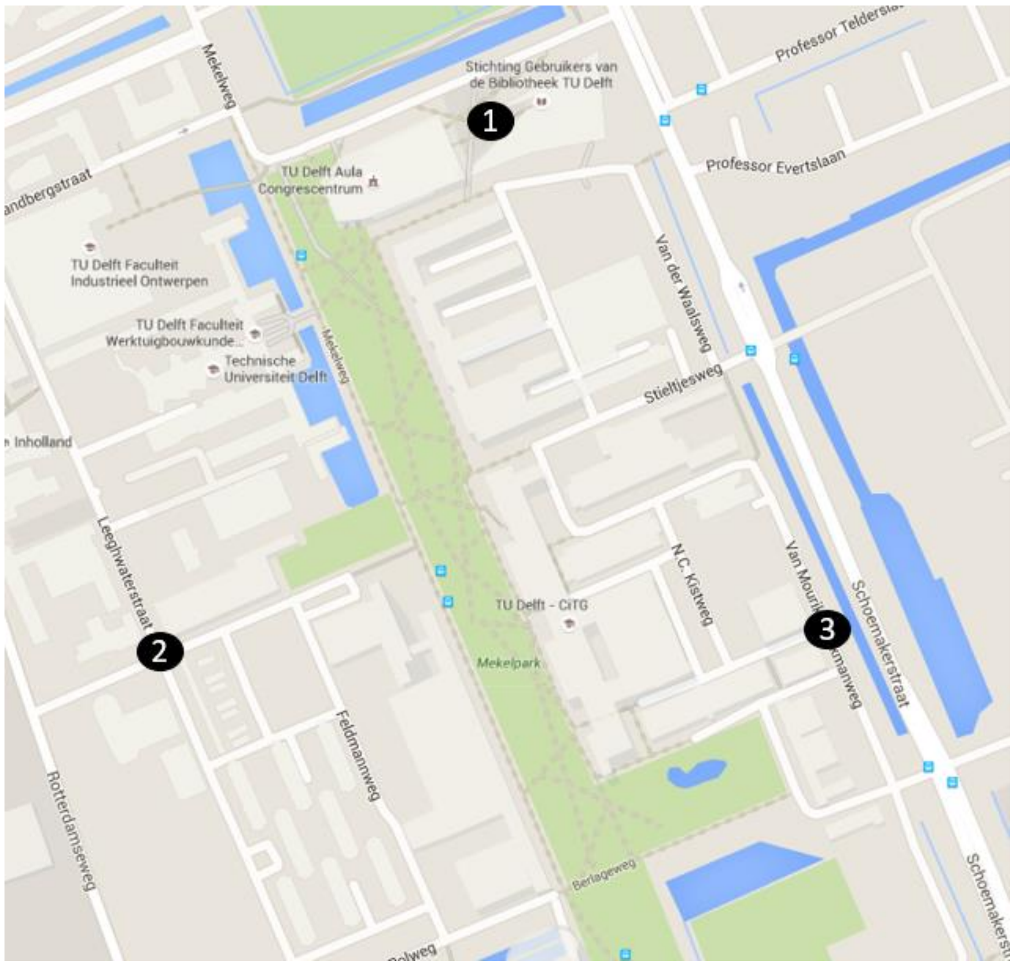

Figure 5.14, starting positions of the user for scenario 2.

The results of the first starting position of the user are shown in figure 5.15, the SEA* method, and figure 5.16, the iterative A* method. Both methods find the target in 6 minutes. The distances are both rounded to 450 meter. However, the iterative A* method finds the target a little faster than the SEA* method. This small difference between the two methods is negligible.

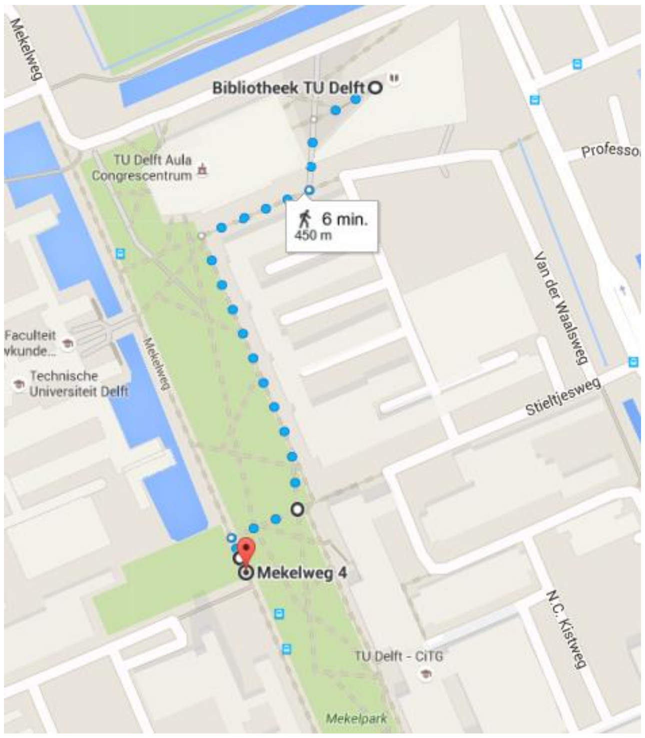

Figure 5.15, 1st position: path for the SEA method, reaching the target in 6 minutes (450 meter).*

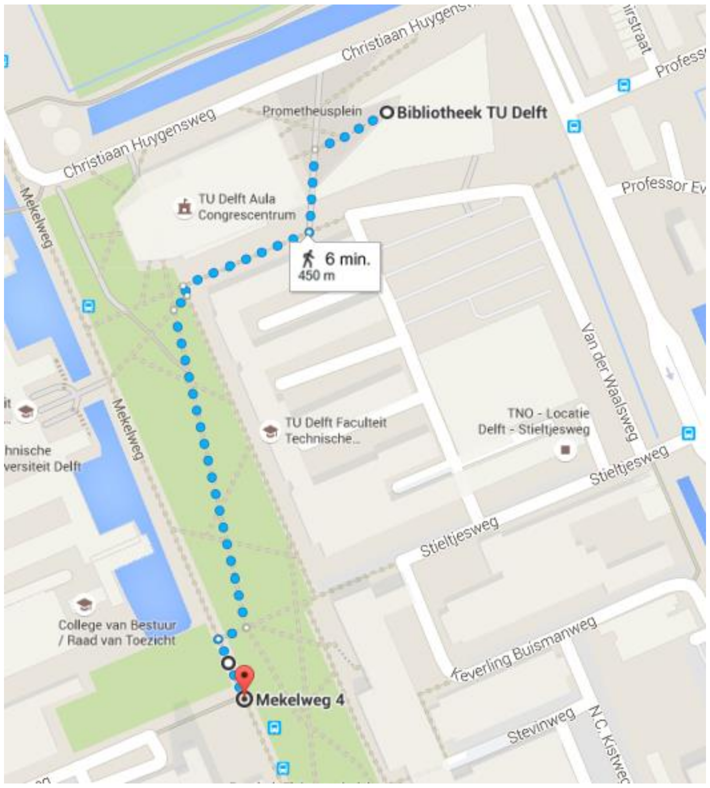

Figure 5.16, 1st position: path for the iterative A method, reaching the target in 6 minutes (450 meter).*

The results of the second starting position of the user are shown in figure 5.17, where the SEA* method, and the iterative A* method both reach the target in 4 minutes. Both methods have the same behavior and the distance of the route is 350 meter.

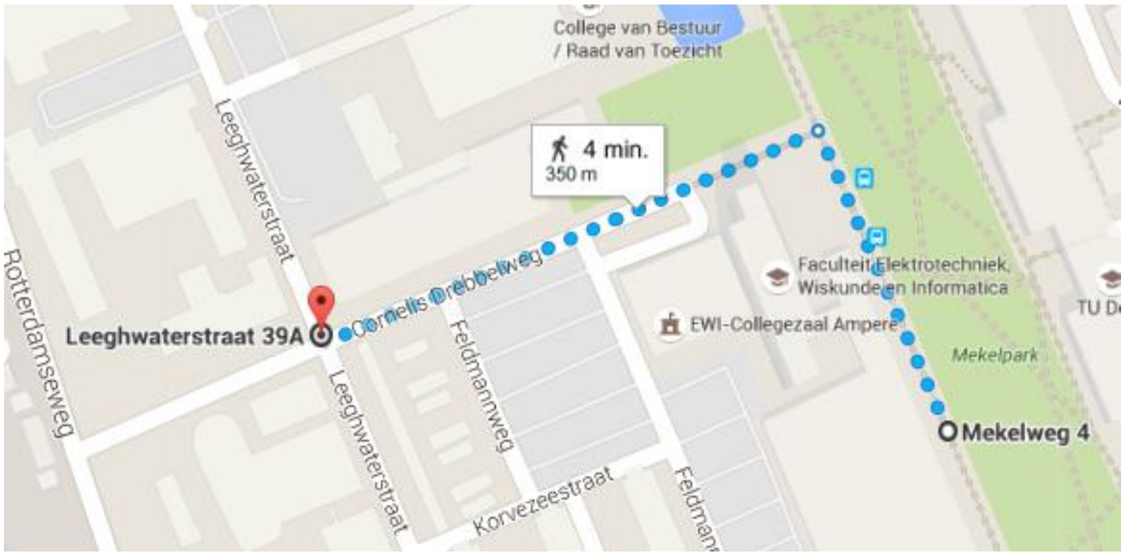

Figure 5.17, 2nd position: path for the SEA method and the iterative A* method, reaching the target in 4 minutes (350 meter).*

The results of the third starting position of the user are shown in figure 5.18, the SEA* method, and figure 5.19, the iterative A* method. The SEA* method reaches the target in 6 minutes, where the iterative A* method does follow the target and not reach the target in time.

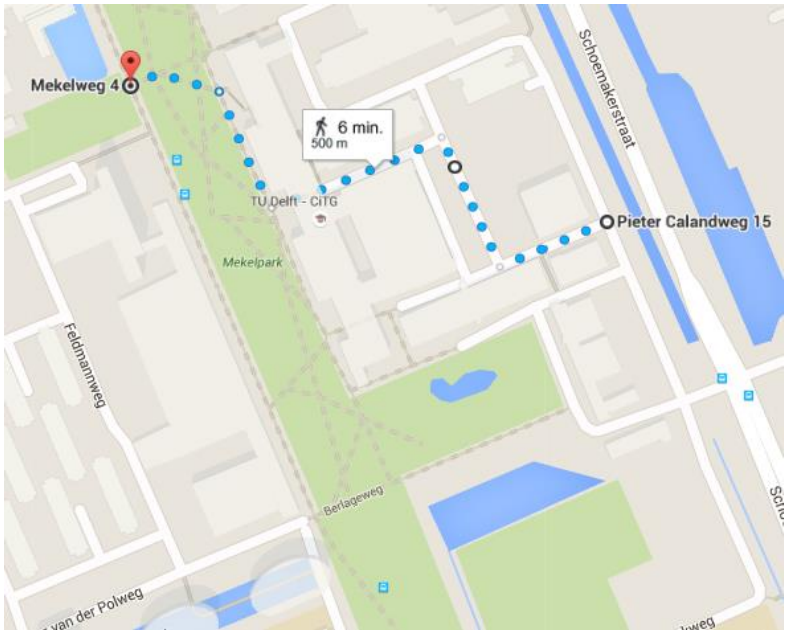

Figure 5.18, 3rd position: path for the SEA method, reaching the target in 6 minutes (500 meter).*

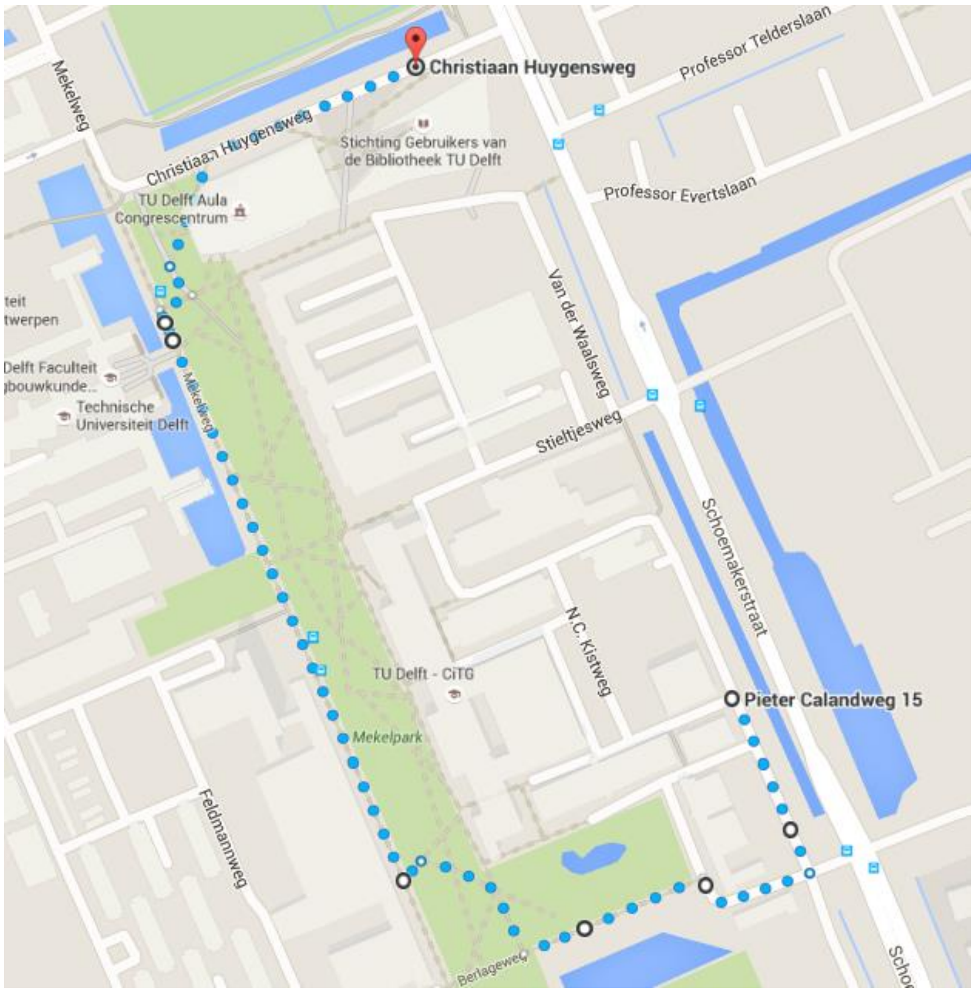

Figure 5.19, 3rd position: path for the iterative A method, not reaching the target.*

5.3 Overview of the results

This section summarizes all the results of the scenarios described in section 5.2. The SEA* method is overall reaching the target faster than the iterative A* method. Also the methods are compared to the optimal solutions. A sub-optimal path is a path that is slightly different than the optimal solution.

Table 5.1, summary of the results presented in section 5.2.

Scenario	Starting point	SEA* method	Iterative A* method
1	1	6 minutes (450 meter): Optimal	6 minutes (450 meter): Optimal
1	2	4 minutes (350 meter): Optimal	9 minutes (700 meter): Not optimal
1	3	8 minutes (650 meter): Optimal	8 minutes (700 meter): Sub-Optimal
2	1	6 minutes (450 meter): Suboptimal	6 minutes (450 meter): Optimal
2	2	4 minutes (350 meter): Optimal	4 minutes (350 meter): Optimal
2	3	6 minutes (500 meter): Optimal	- (target not reached)

5.4 Discussion of the requirements

In this section the criteria defined in section 3.1.1 are used to discuss the results presented in this chapter. The validation criteria are spatially logical, effective and efficient. Spatially logical means that the model is correctly representing the reality. The method is effective if the method always finds its target. The method is also effective if the target is reached as fast as possible, preferable faster than the iterative A* method. The method is efficient if the user gets its directions as fast as possible.

Spatially logical: The spatial model that is used is the Google Maps model. For the test area this model was sufficient and well modelled. In general there are a lot of sidewalks not present in the Google Maps model, especially in small cities. The GPS data used is mapped to the nearest node in the spatial model. Most of the times this is the correct location, however there are cases where this gives unwanted behavior.

Effective: The SEA* method is implemented and compared to the iterative A* method within an outdoor environment. The SEA* method is compared to the iterative A* method beneficial in reaching the target faster. When a person however is standing still, different GPS measurements will be obtained, where it will look like the target slightly changed its position. To assume the target is standing still, a buffer could be used. This is a small buffer around the last position of the target that is retrieved, so that not by every new GPS measurement the target is approaching points of interest far away.

Efficient: The SEA* method is efficient in a way that it calculates a new path every time a new measurement is obtained. In the current implementation this is calculated every 10 seconds, which is fine for human navigation. The SEA* method also works when the target gets each second a new measurement to calculate a path. Wrong GPS measurements however, due to a limited GPS signal, could give a wrong positioning and therefore a wrong prediction of the target.

6. Conclusions and future work

This final chapter gives a summary of all the conclusions of this thesis research. It discusses these conclusions and presents areas that are interesting for future work. The first section, 5.1, gives answers to the sub questions of this research, which will eventually answer the main research question. Then these findings are discussed in section 5.2. Eventually these findings will lead to interesting research topics, which are presented in section 5.3.

6.1 Research questions

To provide an answer to the main research question, the sub questions have to be answered.

1. *What are the current limitations of the A* algorithm for navigating to a dynamic target?*

To provide an answer to this first sub question a literature study is carried out. The A* algorithm is created to navigate a user to a static destination. However, this research shows that the A* algorithm could be used to navigate to a dynamic destination, a target. Then the A* algorithm should be iterative, so that every time the position of the target changes a new path could be calculated. In this way a user could navigate in real-time to a dynamic target. The iterative use of the A* algorithm is the basis for the incremental A* algorithm, which is only a faster implementation of the iterative A* approach. By using the iterative A* approach the user will not always navigate in the most optimal way. This research concludes that the A* approach will result in following behavior of the target, instead of finding the fastest path towards the moving target. The iterative A* method does not use any prediction of the target. This following behavior could result in a long path, while shorter paths are possible. This limitation of the iterative A* method is solved by not directly navigate to the dynamic target itself, but to the prediction of the target. This prediction of the target is defined by the most likely interesting point the target is navigating to. This research focusses on semantics within the building to represent the prediction of the target, the next sub question investigate what semantics could be used. This research proves that navigating to static points around the target, instead of directly navigate to the target, could provide faster paths towards the target.

The A* algorithm is implemented in all common spatial models and therefore support a large variety of models that could use the SEA* method.

2. *What semantics are important for navigation to a moving target?*

Semantics are used to predict where the target is moving towards. People that navigate in a building mainly have a clear end destination. This research shows that the static environment could be used to represent these end destinations for the target. The static environment is everything that cannot move or be moved, which are doors, walls, rooms, windows and the floors. Also navigating to static points is more secure than navigating to a dynamic point, because the accuracy of the position of a dynamic point is less than the accuracy of the position of a static point. The semantics that could predict where a person is going to is for each person different. Therefore prediction points are target specific and defined as points of interest for the target. These points could be every static point in the model that is interesting for the target and if the target is able to navigate to this static point. These points could be general rooms or building entrances as shown in both implementations or could be very specific for a certain target. Information about the target helps to define the points of interest. External data is therefore a promising resource to define proper points of interest. There could be other methods to determine points of interest within an environment, these methods should be investigated. The SEA* method proposes three rules to determine the location of a point of interest. These rules give good results in the two implementations. However, not for every building these rules are applicable. Therefore more rules should be defined.

3. *What are the improvements and limitations of the proposed method over using the iterative A* algorithm?*

The proposed SEA* method shows promising results in both the simulated indoor environment, represented by a 2D square regular grid, as in the outdoor environment, represented by a road network, using real GPS data.

First of all the SEA* method determines what points of interest are approached by the target, because these are the points the target is navigating to. The indoor implementation is sub-optimal due to a lack of knowledge to optimize the code. Here not the distance of the paths are used to determine if the target is approaching a point of interest, but it uses a work around. Therefore the determination of the points of interests that are approached by the target have a small error. Within the outdoor implementation this is solved and gives the correct determination of the points of interests that are approached by the target. Using the lengths of the path to each point of interest should therefore also be implemented within the indoor implementation, as described in the conceptual framework. This would improve the indoor implementation of the SEA* method.

Then the SEA* method uses two approaches to limit down the approached points of interests by the target to only one point, the prediction of the target. The first approach is using pre-processed probabilities, the weights, which shows promising results if these probabilities are well defined. Otherwise these probabilities will have a negative influence on the behavior of the SEA* method. In this case the iterative A* method is even reaching the target faster. Because it is hard to correctly predefine the probabilities, how a user should behave, this approach is not recommended. The second approach shows more promising results, which uses the closest point of interest that is approached by the target as the prediction of the target. In this way the SEA* method always outperforms the iterative A* method. Alternative methods could be used to determine the prediction of the target, because this part is independent from the further implementation.

The SEA* method also uses a scenario based decision in what cases the user should navigate directly to the target or to the prediction of the target. These six cases are defined in section 3.2.5. When the target and the user are at the opposite sides of the prediction of the target (1, 2 and 3), it is always better to navigate first to the prediction of the target and then to the target itself. In all cases this behavior is better than using the iterative A* method. Also in the case where both the user and target are on the same side of the prediction of the target and the target is closer to the prediction (6), then navigating directly to the prediction shows good results. In the cases where both the user and the target are at the same side of the prediction of the target and the target is not closer to the prediction (4 and 5), it is not always clear what the user should do. In an open space it is better to navigate directly to the target, but when there are obstacles this depends on the specific situation. These cases should be investigated and split into different cases, where the obstacles determine what path the user should take.

The SEA* method behaves as the iterative A* algorithm when the target is standing still. Important here is the definition on standing still, where real measurements are not always the same. Using a regular grid this is no problem, because with an accurate positioning system the position of the user would refer to the same location. However, when using GPS data there must be created a small buffer to determine if the target is standing still or not. If the target is still inside this buffer within a 10 second timeframe it is likely to assume that the target is standing still. More research about the size of this buffer must be acquired. Now this buffer is not applied and every new position of the target will lead to a prediction of the target, as if the target moves towards this target, while the target is in fact standing still.

The limitation of the SEA* method is that if the target is walking towards a point of interest and suddenly walks back to where this person comes from the iterative A* method is outperforming the SEA* method. However, the SEA* method will eventually always reach the target if the target is moving slower or at the same speed as the user.

In the case that there are no points of interest defined, the SEA* algorithm will act as the iterative A* method. Also when there are no points of interest defined in the direction of the target, the SEA* algorithm will act as the iterative A* method. This is a nice back-up when no information about points of interest are available. However, using points of interest could provide a faster path towards a moving target.

Now that all the answers to the sub questions are provided, the main research question is answered. The main research question for this thesis is the following:

Which defined objects could be used to estimate the predicting location of a moving person to support navigation to a person in motion?

This research thesis proposes the SEA* method to use semantics, in the form of points of interest, to determine the prediction of the target and uses this prediction to approach the target. Overall the SEA* method uses the positive components of the iterative A* method, semantics and the direction of the target to predict where the target is going to, to successfully reach the target. Points of interest, landmarks, are critical points to check where a person is moving towards. These static locations are promising for navigating to a person in motion. Estimating the predicting location of the target is recommended by first limiting down the points of interest by the approaching points of interest by the target and then using the point of interest that is the closest to the target. This process gives a good prediction of the target in both implementations.

The SEA* approach does only need the current position of the target and does not have to store any information about where the person has been. The target has to share its current location to a user that wants to navigate towards him or her. Therefore the privacy of the target is intact.

6.2 Discussion

The main contribution is to provide users a framework to deal with navigating a user to a person he needs as fast as possible and always find this person. However several assumptions and limitations are defined. First of all the positioning of the user and the target must be accurate to use the SEA* method. Nowadays positioning techniques are sufficient to obtain an accurate location of the user and the target.

The framework is implemented using different spatial models in the indoor and outdoor environment. Therefore this approach is independent of a specific spatial model and could be implemented using a variety of spatial models. This eases the user, because the user is free to select the preferred spatial model. However, the spatial model must support the A* algorithm, must be able to translate the positions of the user and the target to the spatial model and must support adding points of interest to the model. Both for a regular grid and a navigable graph, this research provides an implementation of the SEA* method. The SEA* method is also applicable using irregular grids, a Quadtree, Octree and voxel representation as long as there is a way to use the A* algorithm. For the implementation in 2D this research discusses the applicability in chapter 2. For an Octree and a voxel representation the third dimension must be taken into account, but the basic framework is provided in this research. Therefore could be concluded that the method is scalable for many applications involving navigation to moving targets.

The SEA* method uses the predictions of the target to approach the target in motion. However, there is no knowledge about the path the target will take. Therefore it is not possible to determine exactly where the target and the user could meet each other. Ideally this would be the situation, but calculating every possibility will not be efficient. Offline methods are used to determine these paths, but could not be used in large environments in real time to navigate to a moving person (Sun et. al., 2010).

The speed of the target and the user have no influence on the SEA* method. The search will continue until the user has founds its target. However, the speed of the target could support the determination of the prediction of the target. If a target is walking faster, then points of interest near the target are less likely to be the end destination of the target. On the other hand is a target that is slowing down its speed, is more likely to navigate to a point of interest near the target. Although many factors could have influence on the speed of the target, for example a large crowd will slow the target down or the target could prefer to walk at a fast pace.

The SEA* method is highly dependent on the movement of the target. When a target is moving in a certain direction and then turns 90 degrees and navigates in another direction. The SEA* method should have a high frequency of the current position of the target to react to these changes. Using points of interest near the target as the prediction will limit this effect.

Now the SEA* method only navigates one person towards another person. This method could also be used by multiple individuals that all have to navigate to one target. Each user then needs the location of the target and uses the SEA* method to navigate towards this target. Navigating to a person that follows another person is also possible. The SEA* method is a good way to reach the target if there is no knowledge about that the target is following another person. If there is knowledge about which person the target is navigating to, this point should be used as the prediction of the target. In the case that two people want to navigate to each other it is better to use the iterative A* method, because this path provides the fastest path between two points. Then both users should follow this path to eventually meet each other. Navigating multiple people, three or more, to each other is a more complicated case. In this situation both the SEA* method and iterative A* method could give unwanted circular behavior, where the first person is navigating towards the second person, the second person is navigating to the third person and the third person is navigating to the first person. Therefore these methods could not be used for this kind of navigation.

6.3 Future work

This section presents further research topics about this thesis research. Then all the results of this future work should be used to improve this method and give more insight in navigating a person to another person in motion.

Real-time application

The next step is to create a real-time application to test the SEA* method within the outdoor environment. The implementation described in chapter 5, could be converted to an Android application using Cordova. Then the GPS sensors of the Android devices will give the correct positioning and the method could be tested. Due to defect GPS sensors in the test devices this could not be tested within the time frame of this thesis. Also a real-time application could be made for the indoor environment by using one of the previous discussed indoor positioning techniques.

3D application

Now that the SEA* method is applicable within the 2D environment on both a regular and irregular grid, showing good results, the method should be extended to use it within a 3D environment. An irregular grid, a voxel representation or an Octree are the spatial models that could be used to represent a 3D terrain. Therefore the method should adapt to use it in a 3D environment.

Prediction of the target

Now the prediction of the target is defined by using the movement and points of interest. Other factors, like the speed of the target, could estimate a better prediction. Newly methods to define the prediction of the target could be used to enhance the SEA* method.

Extract points of interest

Now the points of interests are picked manually, with knowledge about the test areas. New rules should be examined to extract points of interest from the map data, both manually as automatic. For example the Google Maps places API could be used to define important places, for example shops, parking spots or churches (Google Places API, 2016).

Redefine cases when to navigate to the target

In the cases where both the user and the target are at the same side of the prediction of the target and the target is not closer to the prediction, it is not always clear what the user should do. In an open space it is better to navigate directly to the target, but when there are obstacles this depends on the specific situation. These cases should be investigated and split into different cases, where the obstacles determine what path the user should take. Here a check is needed what path the target will take to get to the prediction of the target and if there are obstacles in between the user and the target.

Using a visibility analysis

For now the user reaches the target, when both are on the same location, or within a minimal distance of each other. A visibility graph could give useful insight in whether the user and target could see each other. In an open place the user could see the target and just walk towards the target. However, in crowded places this has not always to be the case. Then the vision could be blocked by other people or dynamic obstacles. Therefore a good visibility analysis could support the SEA* method. A visibility graph could be used to give the user a signal that the target could be in the vision range, so that the user should pay extra attention.

Bibliography

Adalja, D., M. (2013). A Comparative Analysis on indoor positioning Techniques and Systems, *International Journal of Engineering Research and Applications (IJERA), Vol. 3, Issue 2, March - April 2013, pp.1790-1796.*

Alfgoor, Z. A., Sunar, M., S., and Kolivand, H. (2015). A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games, *International Journal of Computer Games Technology*, Volume 2015 (2015), Article ID 736138, 11 pages.

ArcGIS (2016). Rasterizing features for 3D. http://desktop.arcgis.com/en/arcmap/latest/extensions /3d-analyst/rasterizing-features-for-3d.htm, seen on 26-04-2016.

Baier, J., A., Botea, A., Harabor, D., and Hernandez, C. (2015). Fast algorithm for catching a prey quickly in known and partially known game maps. *IEEE Transactions on Computational Intelligence and AI in Games, 7*(2), 193-199. doi:10.1109/TCIAIG.2014.2337889.

Bandi, S. & Thalmann, D. (1998). Space discretization for efficient human navigation. *Computer Graphics Forum* 17(3): 195-206.

Berg, de M., Cheong, O., Kreveld, M., Overmars, M. (2008). Computational geometry: algorithms and applications*, chapter 15*. Berlin: *Springer*, 3rd edition, 323–333.

Black, P. E. (2004). Euclidean distance, in *Dictionary of Algorithms and Data Structures* [online], Vreda Pieterse and Paul E. Black, eds. 17 December 2004. (Accessed 25-04-2016) Available from: http://www.nist.gov/dads/HTML/euclidndstnc.html.

Black, P. E. (2006). Manhattan distance, in *Dictionary of Algorithms and Data Structures* [online], Vreda Pieterse and Paul E. Black, eds. 31 May 2006. (Accessed 25-04-2016) Available from: http://www.nist.gov/dads/HTML/manhattanDistance.html.

Boguslawski, P., Gold, C. (2009). Construction operators for modelling 3D objects and dual navigation structures (2009) Lecture Notes in Geoinformation and Cartography, pp. 47-59.

Brown, G., Nagel, C., Zlatanova, S., Kolbe, T.H. (2013). Modelling 3D topographic space against indoor navigation requirements (2013) *Lecture Notes in Geoinformation and Cartography*, pp. 1-22.

Camebridge Dictionaries Online (2016). Landmark, *Camebridge Dictionaries Online* http://dictionary.cambridge.org/dictionary/english/landmark, seen on 25-04-2016.

Chartand, G. (1984). Introductory Graph Theory, Dover Books on Mathematics, *Dover Publications*. ISBN-13: 978-0486247755.

Deo, N. and Pang C. (1984). Shortest-path algorithms: Taxonomy and Annotation. *Networks 14*, 275–323, 1984.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik 1: 269–271*, doi: 10.1007/BF01386390.

Eranki, R. (2002). Pathfinding using A* (A-Star), http://web.mit.edu/eranki/www/tutorials/search/, seen on 06-01-2016.

European GNSS Agency (2013). GNSS Market Report, issue 3, 2013.

Fallah, N., Apostolopoulos, I., Bekris, K., and Folmer, E. (2013). Indoor Human Navigation Systems: A Survey, *Interacting with Computers (2013)*, doi: 10.1093/iwc/iws010.

Fleischmann, P. (1999). DISSERTATION: Mesh Generation for Technology CAD in Three Dimensions, obtained from http://www.iue.tuwien.ac.at/phd/fleischmann/diss.html, seen on 17-06-2016.

Garmin (2016). Garmin DriveSmart. https://buy.garmin.com/nl-NL/NL/cOnTheRoad-cAutomotive-p1.html. Seen on 24-05-2016.

Girard, G., Côté, S., Zlatanova, S., Barette, Y., St-Pierre, J., van Oosterom, P. (2011). Indoor pedestrian navigation using foot-mounted IMU and portable ultrasound range sensors (2011) *Sensors*, 11 (8), pp. 7606-7624.

Goetz, M. and Zipf, A. (2011). Extending openstreetmap to indoor environments: bringing volunteered geographic information to the next level, *Proceedings of the Urban and Regional Data Management: Udms Annual 2011*, pp. 47–58, 2011.

Google (2016). Google Maps JavaScript API V3 Reference, https://developers.google.com/maps/documentation/javascript/3.exp/reference, seen on 03-02-2016.

Google Places API (2016). https://developers.google.com/places/supported_types#table1, seen on 15-06-2016.

Hall, E., T. (1969). The hidden dimension (Vol. 1990). *New York: Anchor Books.*

Hart, P. E., Nilsson, N. J. & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths, *IEEE transactions of systems science and cybernetics*, vol. ssc-4, no. 2, July 1986.

Infsoft (2016). Indoor navigation products. http://www.infsoft.com/products. Seen on 24-05-2016.

Ishida, T., and Korf, R. E. (1991). Moving Target Search. *In Proceedings of IJCAI*, 204–210.

Koenig, S., and Likhachev, M., Liu, Y. and Furcy, D. (2004). Incremental Heuristic Search in Artificial Intelligence. *Artificial Intelligence Magazine*, 25(2), 99-112, 2004.

Koenig, S., and Likhachev, M. (2005). Fast replanning for navigation in unknown terrain. *Transaction on robotics*, 21(3), 354-363)

Koenig, S., and Likhachev M., and Sun, X. (2007). Speeding up moving-target search. *In proceedings of AAMAS*, 1136–1143.

Korf, R., E. (1990). Real-time heuristic search. *Artificial Intelligence*, vol. 42, no. 2-3, 189–211.

Kresse, W. and Danko, D. M. (2012). Springer handbook Geographical information, *Springer Berlin Heidelberg 2012,* DOI 10.1007/978-3-540-72680-7.

Kuffner, J.J. (1998). Goal-directed navigation for animated characters using real-time path planning and control (1998) *Lecture Notes in Computer Science*, 1537, pp. 171-186.

Lee, J. (2004). A spatial access-oriented implementation of a 3-D GIS topological data model for urban entities (2004) GeoInformatica, 8 (3), pp. 237-264.

Li, X., Claramunt, C. and Ray, C. (2010). A grid graph-based model for the analysis of 2D indoor spaces, *in Elsevier Computers, Environment and Urban Systems, Volume 34, Issue 6*, November 2010, Pages 532–540.

Liu, L., Zlatanova, S. (2012). A semantic data model for indoor navigation (2012) *Proceedings of the 4th ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness*, ISA 2012, pp. 1-8.

Lorenz, B. Ohlbach, H.J. & Stoffel, E.P. (2006). A Hybrid Spatial Model for Representing Indoor Environments. In *Proceedings of W2GIS* (LNCS 4295): 102-112. Hong Kong, China.

Ma, T., Yan, Q., Liu, W., Guan, D. and Lee, S. (2011). "Grid task scheduling: algorithm review," *IETE Technical Review, vol. 28, no. 2*, pp. 158–167, 2011.

MarketsandMarkets (2011). Global GPS Market: Products (Marine, Aviation, Automotive, Outdoor/fitness & GPS Enabled Smart Phones), Applications (Navigation, Machine Control, & Logistics Tracking) & Geography (2011 - 2016). November 2011, SE 1089.

MarketsandMarkets (2014). Indoor Location Market by Solution (Tag-based, RF-based, Sensor-based), by Application (Indoor Maps & Navigation, Indoor Location-based Analytics, Tracking & Tracing, Monitoring & Emergency Management), by Service, by Vertical, & by Region - Global Forecast Up to 2019. November 2014, TC 2878.

Math.NET (2016). Math.NET Numerics, distance metrics. http://numerics.mathdotnet.com/Distance.html. Seen on 11-04-2016.

Nagy, B. (2003). Shortest paths in triangular grids with neighbourhood sequences, *Journal of Computing and Information Technology*, 11(2), (2003) 111–122.

Nussbaum, D., and Ÿoṙukc̆u, A. (2015). Moving Target search with subgoal graphs. *Proceedings of the Eighth International Symposium on Combinatorial Search*.

Patel, *A*. (2006). Thoughts on grids, http://www-cs-students.stanford.edu/~amitp/game-programming/grids/, seen on 11-04-2016.

Patel, A. (2006). A*'s Use of the Heuristic, http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html, seen on 11-04-2016.

Rabin, S., & Sturtevant, N. R. (2013). Pathfinding architecture optimization. *Game AI Pro: Collected Wisdom of Game AI Professionals*, 241-252.

Samet, H. (1988). An overview of quadtrees, octrees, and related hierarchical data structures. *Theoretical Foundations of Computer Graphics and CAD*, pp. 51–68, *Springer*, 1988.

Skantrae (2016). http://www.skantrae.com/service_center/veelgestelde_vragen/ standaard_deurmaten, seen on 15-06-2016.

Stentz, A. (1994). Optimal and efficient path planning for partially-known environments. *In proceedings of the International conference on robotics and automation*, 3310-2217.

Stentz, A. (1995). The focused D* algorithm for real-time replanning. *In proceedings of IJCAI*, 1652-1659.

Sun, X., Koenig, S., and Yeoh, W. (2008). Generalized Adaptive A*. *In proceedings of AAMAS*, 469-476.

Sun, X., Yeoh, W., and Koenig, S. (2009). Efficient incremental search for moving target search. *In Proceedings of IJCAI*, pages 615–620, 2009.

Sun, X., Yeoh, W., and Koenig, S. (2010). Generalized Fringe-Retrieving A*: Faster moving-target search on state lattices. *In Proceedings of AAMAS*.

Sun, X., Yeoh, W., and Koenig, S. (2010). Moving target d* lite. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*: volume 1-Volume 1, 67–74.

TomTom (2016). Navigatie app, sneller naar je bestemming. https://www.tomtom.com/nl_nl/drive/sat-nav-app/go-mobile/. Seen on 24-04-2016.

Trovato, K., and Dorst, L. (2002). Differential A*. *IEEE Transactions on knowledge and data engineering*, 14(6), 1218-1229).

Trudeau, R. J. (1994). Introduction to Graph Theory, Dover Books on Mathematics, *Dover Publications*; 2nd edition (February 9, 1994). ISBN-13: 978-0486678702.

Undeger, C., and Polat, F. (2007). Real-time edge follow: a real-time path search approach. *IEEE Transactions on systems, man, and cybernetics* —part C: applications and reviews, Vol. 37, No. 5.

Uras, T., Koenig, S., and Herńandez, C. (2013). Subgoal graphs for optimal pathfinding in eight-neighbor grids. In *Twenty-Third International Conference on Automated Planning and Scheduling*, 224–232.

Worboys, M., Modeling indoor space, in *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness,* pp. 1–6, ACM, 2011.

Zlatanova, S., Sithole, G., Nakagawa, M., Zhu, Q., (2013). Problems in indoor mapping and modelling, Acquisition and Modelling of Indoor and Enclosed Environments 2013, Cape Town, South Africa, 11-13 December 2013, ISPRS Archives Volume XL-4/W4, 2013.