Indoor Navigation to a moving target

MSc Geomatics Graduation Plan

T. Nagelkerke

January 06, 2016

Contents

1. Introduction

In the last decade the navigation market has grown a lot. Especially car navigation boosted this industry. Nowadays most cars already have a build in navigation device. These navigation devices are based on the Global Positioning System (GPS) which determines the position of the car. The position of the car is the starting point of the search. The user can then navigate to a desired end point.

GPS does not have accurate measurements to obtain the position in an indoor environment. 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. This accurate positioning is then used to navigate to a preferred destination. The process of finding a path to the desired end point is called pathfinding.

Current indoor positioning techniques are used to find a path from the user to a static end point, for example a room in a building. For most cases navigating to a static end point is accurate, like an appointment in a room or navigating to someone or something that is staying in the same place. But what if the person wants to navigate to a moving target? Navigating to a location can also be solved by using signs in a building, e.g. room 1.10. Navigating to a moving target is not that easy, even if the starting is known, it is not guaranteed that the user might meet the target as is depicted in figure 1. In this figure user A is navigating to a target B, but while A moves towards B, B moves. In this case A and B never meet each other.

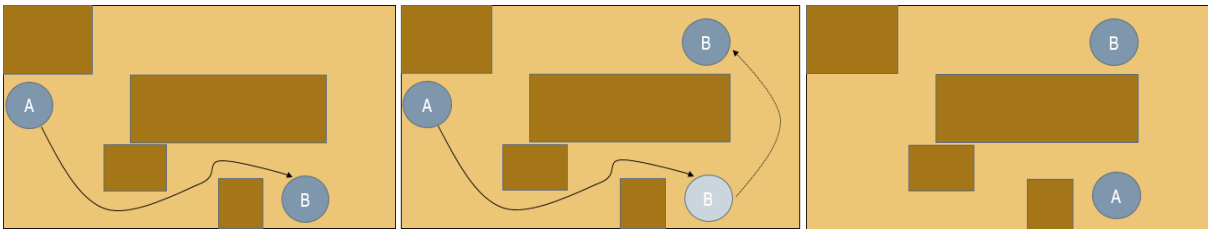


Figure 1, problem navigating to a moving target.

Therefore there must be an up-to-date position of the target or there must be knowledge where the target is heading to. Current positioning techniques and systems are able to obtain the up-to-date accurate position. In this case it is possible to keep following the moving target.

A moving target is someone or something that moves within a building. This could be someone or something that needs to be found, like a lost child, is needed, like an asset or a coworker, or has to be caught, like an intruder. An asset could be a scarce tool in a hospital that is expensive and is used frequently in different operating rooms. Also it could be a moving robot within a building that has to be found. In all cases it is possible to track assets by using cheap RFID tags or using cellphones. Apart from finding assets and robots quickly, the user could find his child within an indoor playground more quickly, or the police can track and arrest an intruder when they have the permission to track this intruder by his cellphone.

These examples show the benefits which navigating to a moving target has over navigating to static targets. Therefore this research will focus on indoor pathfinding to moving targets.

In the current literature there is nothing found about navigation of a human to a moving target. The literature shows different ways to find a path from the starting point to a static end point, but lacks the knowledge how to navigate to a moving target.

In other domains like robotics and the gaming industry, navigating to a moving target is widely

explored and implemented. These concepts can be used to navigate persons within a building to a moving target. But navigating persons differs from navigating robots or game characters. Persons do behave differently, while robots and game characters have to be programmed. Different from robots is that people can avoid obstacles in real-time. Humans also differ from game characters in the way that they walk toward something within a building, for example a door, a room or the toilet.

1.1 Scientific Relevance

There is no literature found in navigating a person to a moving target. In the previous section examples are explained, why navigating to a moving target could be useful. Especially navigating to a person that can't contact the user (small children or people with dementia) will benefit from this. This research will examine the possibilities to navigate to a moving target within a building.

1.2 Related work

Moving Target Search (MTS) has been introduced by Ishida and Korf (1991). MTS 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 isn't efficient for a large environment. Also the position of the moving target is not known in advance.

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, when the target moves, it will update the information that is available and calculate a new path. Two algorithms that use the online approach are real-time search algorithm (Ishida and Korf, 1991) and incremental search algorithms (Sun, Yeoh and Koenig, 2009).

1.2.1 Real-time search algorithms

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 (RTEF-ARM), which uses perceptual information. Undeger and Polat their method has improvements over previous real-time search algorithms like real-time A* (RTA*) 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.

1.2.2 Incremental search algorithms

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). 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* (GAA*) (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 (MT-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 an indoor 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 Ÿoŕukču (2015) invented a new incremental search algorithm called Moving Target Search with Sub-Goal Graphs (MTSub) which has a faster runtime than the previous named incremental search algorithms. Nussbaum and Ÿoŕukču use the subgraph algorithm of Uras, Koenig and Hernandez (2013). The idea is to simplify the map representation 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.

1.3 Research objective

All the incremental search algorithms are used and implemented in the gaming industry. The ideas to navigate game characters to each other in a game will be applied to navigate a user to a moving target in a real buildings. The main research question for this thesis is as follows:

To which extent is it possible to navigate a person to a moving target within a building using incremental search algorithms?

The goal of this research is to obtain knowledge about whether incremental search algorithms support human navigation to a moving target. Standard incremental search algorithms will be implemented and tested according to predefined criteria. The following sub-questions emerge to solve the main research question:

1. What incremental search algorithm will be used to perform the pathfinding search to a moving target?
2. What map representation of a building is most suitable to represent the static obstacles and to test the chosen search algorithm?
3. What simulation of the moving target is most suitable for testing these algorithms?
4. What criteria are important to judge whether the incremental search algorithm add value over a simple A* search?

1.4 Scope of the research

In this research the aim lies on implementing incremental search algorithms to navigate to a moving target. The positioning of the moving target is out the scope of this research and this will be solved by simulating the moving target. The 3rd sub-question will give the answer on how this will be

simulated. A simple program will be made to test and analyze the algorithms. The indoor environment in which the pathfinding algorithms are implemented is static and two dimensional. When there is additional time, the third dimensional will be examined. The navigation is done with a single agent, which means that there is only one user that has to navigate to only one moving target. The building of the faculty of architecture at the University of Delft will be used. Eventually additional buildings could be examined.

2. Methodology

In this section first the overall methodology will be explained, then the requirements of each step will be clarified. The first step is to obtain/create a digital floor plan. Then this map has to be converted to a map representation. An appropriate simulation of the moving target has to be created. Then the A* algorithm is implemented to validate the advantages and disadvantages of the incremental search algorithms that will be implemented and tested. The last step is to actually implement the incremental search algorithms and compare these with the A* algorithm to certain criteria.

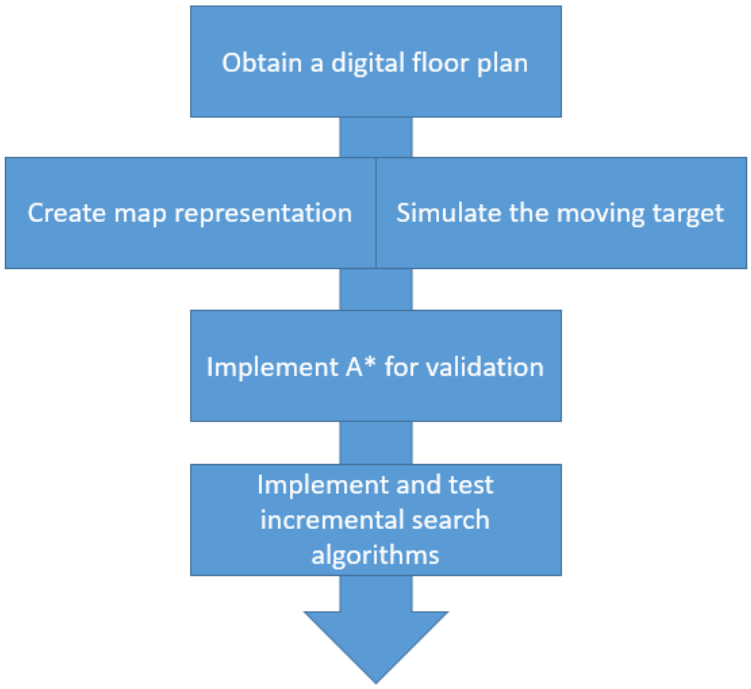


Figure 2, overall methodology used in this thesis.

2.1 Obtain/create the digital floor plan

In this study the building of the faculty of architecture will be used. The first step is to obtain the digital floor plans. When there is no digital floor plan available, this has to be created, or another building is used. Worth mentioning here is that the digital floor plans represent the building in a moment in time. Therefore it is possible that the floor plan does not represent the current status of the building. For testing purposes this is no problem.

2.2 Create map representation

2.2.1 Map representation theory

The building that is used must have a representative map so that it is possible to calculate a route from the starting point to the end point. In a recent study Algfoor et. al. (2015) created an overview of the current map representations. They divide the map representations into two categories: Grids and Hierarchical techniques. The grids are subdivided to regular grids and irregular grids. A further decomposition is shown in figure 3. In this recent study they note that a regular square grid gives the best representation for a plane surface, while irregular grids are better to represent three dimensional surfaces.

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		

*Figure 3, decomposition of map representations (*Algfoor et. al., 2015).

Of importance is that a building is a static environment and the floor plans will represent the obstacles for humans correctly. Mention that a floor plan is only a moment in time, so it could be that the floor plan is not representing the actual status of the building. The environment where the user is navigating to the moving target is known in advance.

2.2.2 2D square grid

The map representation must be clear and support all the algorithms that will be tested. A 2D square grid will be used, because it represents flat surfaces better than other map representations and the algorithms mentioned before are all implemented and tested on a 2D square grid. Although it gives some overrepresentation of the obstacles (figure 4), it is a clear and fast map representation to calculate paths.

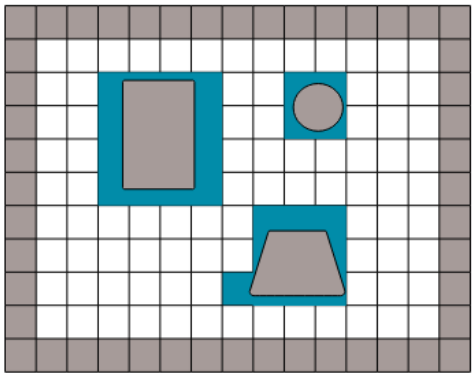


*Figure 4, 2D square grid representing three obstacles (*Algfoor, 2015).

The digital floor plan will be crossed by a 2D square grid. When there is an obstacle in the digital floor plan the whole ‘tile’ will be marked as an obstacle and it is not possible to cross this tile during the navigation, see figure 3. The process to convert a vector map to a raster map is called rasterization.

The size of the grid is determined by several factors. The first factor is the floor plan, the grid size has to represent the obstacles and the free space in a way that the grid size does not block a passage. 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, but in this case the user and the moving target can move the same number of blocks in the same time interval.

When navigating on a grid three possible movements are possible, movement from each tile to a neighboring tile (figure 4a), movement by the edges (figure 4b) or moving along the vertices (figure 4c).

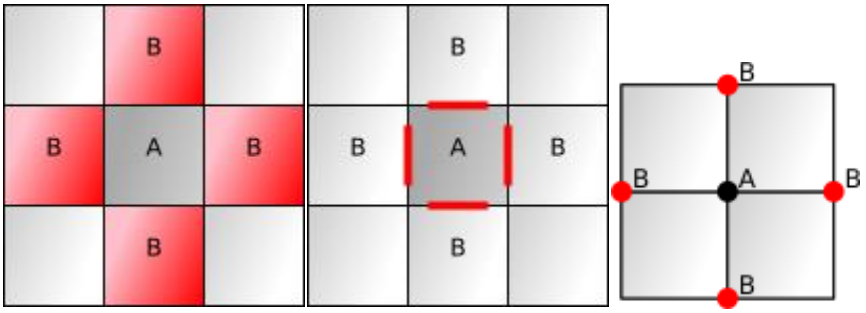


Figure 5, a) Tile movement, b) Edge movement, c) Vertex movement. (http://www-cs-students.stanford.edu/~amitp/game-programming/grids/)

For this research the tile movement (figure 4a) is used and vertical, horizontal and diagonal movement is allowed. The diagonal movement is allowed to represent the human behavior.

2.3 Simulate the moving target

Indoor positioning is outside the scope of this thesis, therefore simulation will be used to simulate the behavior of the moving target. The behavior should be according to the natural movement of persons, either the persons are moving, or persons move an asset. Several situations will be tested.

The person is heading towards a specific direction, for example a door or a room. This will be simulated by using the shortest route from the position of the target to this door or room, using the A* algorithm. Also a person could act randomly and therefore for each step that the user takes the moving target can either move any direction, or stand still. This is the same according to testing the incremental search algorithms in the literature. The last possible simulation is simulating the moving target to walk logical patterns in the building. The actual simulation has to be determined when the floor plans are obtained/created.

In this way the position of the moving target is always known when the user searches for a path towards the moving target. Also the direction, vector, of the moving target is known as it is the movement of the target. The position of the user and the moving target will spawn at random at some preselected places, like doors or specific rooms.

2.4 Implement A* for validation

To check if the used algorithms are useful, the A* algorithm is implemented to compare the results to the used incremental search algorithm. In this way it is possible to conclude what possibilities are useful and what algorithms aren't. The A* is programmed from scratch, the code is available and has to be adjusted to the square grid. Also the A* algorithm will be used to simulate the movement of

the target. As heuristic (h (n)) the Euclidean distance will be used, because diagonal movement is allowed. The costs of the tiles are calculated by the following formula, according to the A* algorithm: f(n) = g(n) + h(n). The distance from each tile to the neighboring tiles is for all tiles the same. The A* algorithm will be requested by every movement the user makes, in this way the benefit of a faster incremental search algorithm (next section) is shown. The pseudocode to program the A* is the following (Eranki, 2002):

```
// A*
initialize the open list
initialize the closed list
put the starting node on the open list (you can leave its f at zero)

while the open list is not empty
    find the node with the least f on the open list, call it "q"
    pop q off the open list
    generate q's 8 successors and set their parents to q (if successor is
                                                          not a obstacle)
    for each successor
     if successor is the goal, stop the search
        successor.g = q.g + distance between successor and q
        successor.h = distance from goal to successor
        successor.f = successor.g + successor.h

        if a node with the same position as successor is in the OPEN list
            which has a lower f than successor, skip this successor
        if a node with the same position as successor is in the CLOSED list
            which has a lower f than successor, skip this successor
        otherwise, add the node to the open list
    end
    push q on the closed list
end
```

2.5 Implement and test incremental search algorithm

Now that the map representation is available and the simulation is ready, the actual implementation of an incremental search algorithm is the last step before testing. For this step the incremental search algorithm FRA* (Sun, Koenig and Yeoh, 2009) is implemented and tested. This algorithm has less preprocessing than the faster MTSub (Nussbaum and Ÿořukču, 2015) and using the CPDs (Baier et. al., 2015) and is proven to be better in static environments by Sun, Koenig and Yeoh (2010). The pseudocode to program the FRA* is the following (Sun, Koenig and Yeoh, 2009):

```
procedure InitializeCell(s)
{01} if (generatediteration(s) ≠ iteration)
{02}     g(s) := ∞;
{03}     generatediteration(s) := iteration;
{04}     expanded(s) := false;
function TestClosedList(s)
{05} return (s = s_start OR (expanded(s) AND parent(s) ≠ NULL));
function ComputeShortestPath()
{06} while (OPEN ≠ ∅)
{07}     delete a state s with the smallest g(s) + h(s, s_goal) from OPEN;
{08}     expanded(s) := true;
{09}     forall s' ∈ N(s)
{10}         if (NOT TestClosedList(s'))
{11}             InitializeCell(s');
{12}             if (g(s') > g(s) + c(s, s'))
{13}                 g(s') := g(s) + c(s, s');
{14}                 parent(s') := s;
{15}                 insert s' into OPEN if s' is not in OPEN;
{16}     return true if (s = s_goal);
{17} return false;
function UpdateParent(direction)
{18} forall s ∈ N(cell) in direction order, starting with parent(cell)
{19}     if (g(s) = g(cell) + c(cell, s) AND TestClosedList(s))
{20}         parent(s) := cell;
{21}         cell := s;
{22}         return true;
{23} return false;
procedure Step2()
{24} cell := s_start;
{25} while (UpdateParent(counter-clockwise)) /* body of while-loop is empty */;
{26} cell := s_start;
{27} while (UpdateParent(clockwise)) /* body of while-loop is empty */;
function Step3()
{28} parent(s_start) := NULL;
{29} forall s ∈ S that belong to the search tree rooted at previous_s_start
{30}     parent(s) := NULL;
{31}     delete s from OPEN if s ∈ OPEN;
procedure Step5()
{32} forall s ∈ S on the outer perimeter of the CLOSED list, starting with anchor[a]
{33}     OPEN := OPEN ∪ {s} if (s is unblocked AND s ∉ OPEN);
{34} forall s ∈ OPEN
{35}     InitializeCell(s);
{36} forall s ∈ OPEN
{37}     forall s' ∈ N(s)
{38}         if (TestClosedList(s') AND g(s) > g(s') + c(s', s))
{39}             g(s) := g(s') + c(s', s);
{40}             parent(s) := s';
{41}             insert s into OPEN if s is not in OPEN;
function Main()
{42} forall s ∈ S
{43}     generatediteration(s) := 0;
{44}     expanded(s) := false;
{45} iteration := 1;
{46} InitializeCell(s_start);
{47} g(s_start) := 0;
{48} OPEN := ∅;
{49} insert s_start into OPEN;
{50} while (s_start ≠ s_goal)
{51}     return false if (NOT ComputeShortestPath());
{52}     openlist_incomplete := false;
{53}     while (TestClosedList(s_goal)) /* Check of Steps 1 and 4 */
{54}         while (target not caught and target is on shortest path from s_start to s_goal)
{55}             follow shortest path from s_start to s_goal;
{56}         return true if target caught;
{57}         previous_s_start := s_start;
{58}         s_start := the current cell of the hunter;
{59}         s_goal := the current cell of the target;
{60}         if (s_start ≠ previous_s_start) /* Check of Step 1 */
{61}             Step2();
{62}             anchor := parent(s_start);
{63}             Step3();
{64}             openlist_incomplete := true;
{65}     if (openlist_incomplete)
{66}         iteration := iteration + 1;
{67}         Step5();
{68} return true;
```

2.6 Test criteria

To check which of the implemented algorithms are suitable to navigate a user to a moving target some important criteria are needed to compare the algorithms. In all the possible cases, named in the introduction, speed is an issue. Therefore the time that it takes to actually reach the target is an important criteria. The number of tiles that the user has to walk to the moving target represents the speed of reaching the target. Second, the speed of the calculation of the algorithm must be quick, people are not waiting minutes to calculate a route. Therefore the runtime of the algorithm is important. The two most important criteria are stated in table 1.

Table 1, test criteria and the measurement.

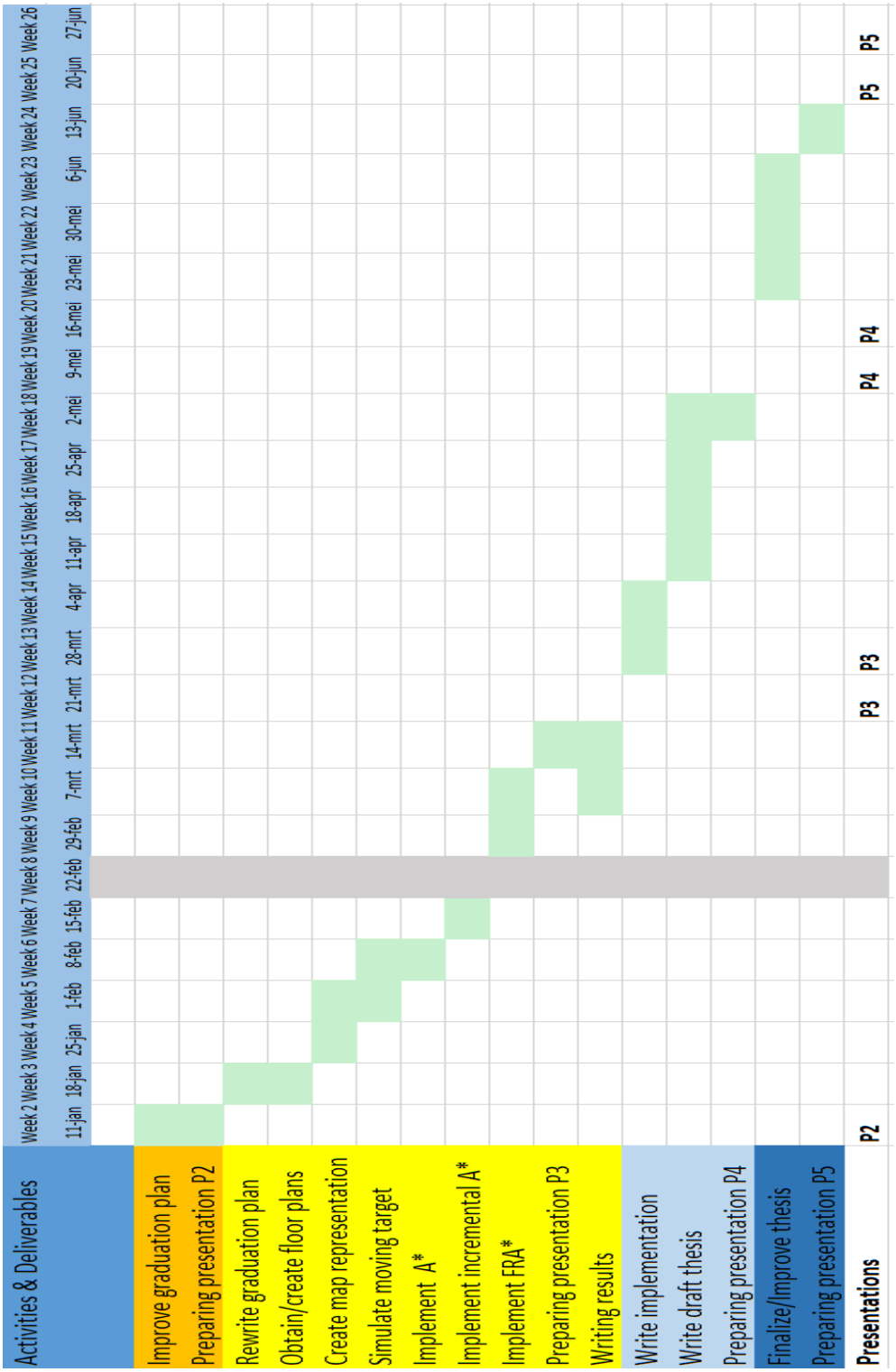
Criteria	Measurement
Time to reach the moving target	# Tiles moved
Total runtime	Seconds

2.7 Programming language

The programming language that will be used to program the algorithms is Python 2.7.8. If it turns out that Python is too slow to deal with the algorithms, C++ will be used.

3. Planning

The following GANNT table is used to schedule all the activities to meet the requirements to fulfill this thesis. All the dates previous from the P2 presentation are excluded. The colors represent the tasks that have to be executed before the next milestone (P). The grey highlighted week is the week that I will visit family in Austria, so no work can be done in that week.



Weekly meetings will be held with the first mentor when necessary, these has yet to be decided. Guidance and feedback will be provided by the second mentor during meetings and the presentations. The co-reader has yet to be decided.

During my thesis I will follow the course ME1130 Robotics Practicals to get some quick hands-on experience with Linux, C++ and the Robot Operating System. This course does not have an exam, but two assignments that has to be delivered. There is no set date for these deliverables.

The dates of all the milestones according to the graduation calendar are shown below. Only the exact date of the P2 presentation is set, the dates of the other milestones will follow. If those dates are known I will email those to both mentors.

Milestone	**Date**
P2	14-01-2016 11:45 BK-IZ Z
P3	21-03-2016 - 01-04-2016
P4	09-05-2016 - 20-05-2016
P5	20-06-2016 - 01-07-2016

4. References

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, http://dx.doi.org/10.1155/2015/736138.

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

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

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

Koenig, S., and Likhachev (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.

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

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.

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

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.