Adaptive Indoor Map Generator for USARSim

Olaf Zwennes
5974100

Adaptive Indoor Map Generator for USARSim

Olaf Zwennes
5974100

Bachelor thesis
Credits: 9 EC

Bachelor Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Dr. A. Visser

Institute for Informatics
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

June 24th, 2011

Abstract

This paper focuses on the method and implementation of a map generator for the USARSim environment, which is capable of generating indoor environments. The generator adapts to a difficulty measure, which signifies how difficult the generated map should be, when mapped by a robot. Both the method of the procedural generation process as well as the theory of the difficulty measure are explained, followed by the implementation of the generator. Multiple maps with various difficulties are generated and mapping runs are simulated by experienced robot operators. Then the difficulty is assessed by these operators and compared to the difficulty level of the maps. The rules of the generator turn out to be able to influence the difficulty of the maps, but are not able to consistently generate maps with a specific difficulty, due to being too naive an encoding of the complexities of 'difficulty'.

Keywords: Simulation, Robotic Mapping, Procedural Map Generation, Rule Based Configuration

Contents

1 Introduction

For many a roboticist, the ultimate thrill is to have your programmed robot successfully fulfilling a task in front of your eyes. But before a real life robot is capable of impressive feats like mapping out rooms and finding its way around a campus, a lot of testing has to be performed. It is not uncommon for this testing to be performed in a simulated environment, in order to simplify testing and making analysis of the experiments easier. In order to best simulate the sensor input of a real-world robot, sophisticated video-game engines are used to simulate the environments and the robot itself. A popular software package used for these kinds of robot simulations is the USARSim package [2], build on the Unreal Engine 3^{TM} game engine.

USARSim offers various environments with its default package, for the simulated robots to experiment inside of. There are various types of environments, like indoor and outdoor maps, as well as various levels of complexity. Users of USARSim can use the game engine's tools to create arbitrarily many and diverse maps to offer a representative collection of maps to test the algorithms for mapping and localization of the robot. The experiments in a simulated environment are usually used to test an algorithm in as many situations as possible, in order to prove the performance of the algorithm in general. While an algorithm may be written for a specific type of environment, like indoor maps, it always needs to be tested on a wide variety of maps in order to confirm that the algorithm performs well on a arbitrary environment of that type.

Procedural generation can be used to generate diverse maps of various types in a game engine. The process has already been used to generate both outdoor and indoor maps, as well as assets, like vegetation in outdoor maps or complete rooms in indoor maps, to fill those maps with a rich variety of objects [6][5][8]. Procedural generation is the use of algorithms, usually with one or more randomized factors, to generate graphical assets, like the examples mentioned before. In procedural generation, there is a balance between randomizing parameters of the generation algorithm and having these parameters set by strict rules that encode knowledge about the type of asset to be generated.

Indoor environments procedurally generated for use in USARSim could possibly have a variable difficulty for robot mapping tasks. This difficulty measure may be encoded in the rules of the generation algorithm, in order to ensure that the difficulty of a generated map is controlled. This would require the generation rules to generalize past just encoding the structure of an indoor environment, to also encoding a property like difficulty for the robot's mapping task.

The central question of this paper is: Can the rules used in an algorithm for generating indoor environments be used to make the generator adaptive to a difficulty measure for a robot mapping task? This means the algorithm will have to be able to generate indoor maps that reflect a user-set difficulty, where an increase in difficulty results in a indoor environment that is harder to map for the simulated robot. This will a require theory of the difficulties of mapping an environment, as well as procedural generation techniques.

In the next section, related work is discussed and compared to this research. Following the related work is a section on the method and application of the adaptive indoor map generator, explaining the choices made in the application. The next section discusses how the experiments with the generator are performed and the results of these experiments are given afterwards. Finally, a conclusion is drawn from those results and the research as a whole is evaluated in a discussion section. This report finishes with suggestions for future work.

2 Related Work

Related and relevant work can be found in both the field of procedural generation and the field of robotics. In the field of procedural generation, there is little work in the generation of indoor environments. An exception is the work of Tutenel, Bidarra, Smelik and de Kraker in rule-based layout solving [8]. Their rule-based system defines classes consisting of a particular type of object and rules associated with these classes determine their placement in an interior layout. A solver then takes these rules into consideration and places objects inside an interior plan to generate indoor environments. A rule-based system is also, in a highly simplified manner, implemented in this report, but it is extended to encode the abstract notion of a difficulty measure for each entity. The rule-based system of Tutenel et. al. is also fairly rigid, solving the rules to one best solution. By assigning probabilities to solutions, a more diverse set of interiors can be generated, at the cost of not always generating the best solution. This approach is used in this report.

Having an environment, like a program or a game, adapt to a difficulty measure, has been researched in a variety of contexts. Carro, Breda, Castillo and Bajuelos presented a framework for creating adaptive educational games [3]. The framework requires a set of parameters, one of which is a difficulty of each game. A game of a specific difficulty (among other features) is then considered, given the progress of the student/player. The system thus uses annotation and rules to adapt an educational game to the skill level of the player. In the context of game entertainment, a system for procedurally generating levels for platform games has been proposed by Compton and Mateas [4]. The difficulty of such a platform level is calculated by determining the timing and the spacial window required for successfully landing a jump from one platform to the next. So contrary to the previously noted research, in this case a simulation of the player's behavior is used to calculate the difficulty, rather than annotations and a rule-based system. A rule-based system is used in this report, instead of simulation as a method to calculate difficulty, because the freedom of movement of a robot is too large to reliably calculate the difficulty of a map using simulations alone.

While there has not been any research in adapting virtual environments to a simulated robot's skill at tasks like mapping, there have been reports noting the specific difficulties particular algorithms have when mapping environments. A book by Thrun, Burgard and Fox notes four important mapping problems [7]. The first problem is size, which means the larger an environment is, compared to the range of the robot's sensors, the more difficult the mapping task becomes.

This problem is relevant to this report, because a map generator can vary the size of elements of the map (or the entire map) in order to vary the difficulty of that map. The second problem is noise in sensors, because when noise increases, localization and recognition becomes harder and thus mapping becomes harder. The third problem is perceptual ambiguity. When different places in an environment look a lot alike, it is more difficult for a robot to distinguish between these places. This problem can be used by the map generator, by creating ambiguity inside a map, it will generate a more difficult map. The last problem Thrun et. al. note is cycles, where a robot can come back to the same location using a different path. The odometry errors accumulate when a robot moves through a cycle, which results in mapping inaccuracy. This last problem is also relevant to the indoor environment generator, because cycles can be generated or prevented in order to increase or decrease map difficulty.

3 Method

USARSim includes several tools to make certain tasks required for robot simulation easier. One of those is the so called World Generator [1], which is a simple map creator, where users can drag pre-made rooms and hallway pieces to a grid of arbitrary size and then export the created map to a format that can be converted to a 3D environment for use in USARSim. The tool also includes a procedure to automatically generate a map using simple generation rules. For instance, The hallways are laid out using a Manhattan-like grid with a constant distance between hallways. This simple generator is expanded in this report to adapt to a difficulty that the user sets when generating a new map.

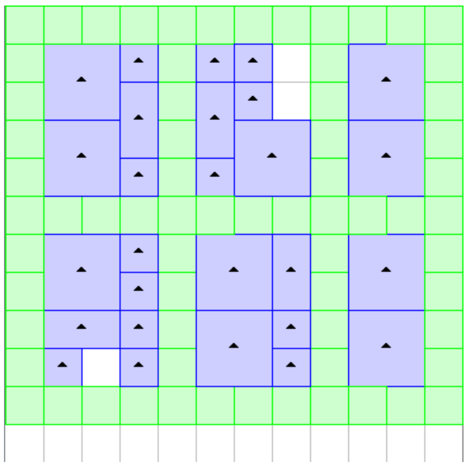


Figure 1: A standard map generated by the World Generator tool.

There are three main rules in the existing World Generator, that encode the geometric properties of the environment. The first is the layout of the hallways in the map, where horizontal and vertical hallways are placed a specific distance apart, which span the entire width and height of the map respectively. In Figure 1 this hallway layout is clearly visible. The light-green colored grid positions are hallway pieces and the light-blue colored boxes are pre-made rooms inserted into the map with the triangle in the center determining their orientation. The white grid positions are empty (solid wall) and not accessible by the robot. The hallway rule looks like this in pseudo-code:

```
**hallwayRule(horizontalSpacing, verticalSpacing)**
for every column dividable by horizontalSpacing
        fill entire column with hallway pieces
for every row dividable by verticalSpacing
        fill entire row with hallway pieces
```

The second rule places the rooms in the empty space randomly:

```
**roomRule(roomList)**
for every empty grid position starting from the top-left to the bottom-right
        take a random room from roomList
        if there is enough empty space for the room to fit
                place room in empty space
        else: pick another random room
        if there is still empty space below the current position
                move down the length of the room
        else
            move to the right the length of the largest room
            move back up to the top empty space
```

The third and last rule places the doorways to the rooms:

```
**doorwayRule(doorwayPercentage)**
for every side of every room
        if random number is smaller than doorwayPercentage
                remove a doorway-sized part of the wall in the middle
```

These decidedly simple rules encode all geometric properties of the generated map, with the exception of the content of the rooms, which is pre-made by human designers. Together, these rules generate the entire environment, but they do not create very varied maps, as variation is only achieved by choosing rooms at random and randomly placing doorways (if $doorwayPercentage < 1$).

The rules that encode the difficulty measure and thus result in an adaptable map generator are of the same type as the above rules that encode the geometric properties of the generated map. In fact, they are sometimes just a small enhancement to the existing rule. For instance, the rule that places the hallways in a grid structure equidistant from each other can encode a difficulty by simply changing the distance between the hallways. This means creating hallways with less cross-section with other hallways and that results in long monotonous hallways which should create location ambiguity for the robot. The reason that the same type of rules can be used to encode an abstract concept like difficulty, as well as encode the actual geometry of the environment, is that the difficulty measure has a tangible effect on the geometry and it is this effect that is encoded in the rules. This does however mean that the actual difficulty is not directly encoded and the effect of different difficulties is interpreted by the creator of the rules. This will result in a simplification of the likely complex dynamics of real-life locations that show different difficulties, when mapped by a robot. This problem could possibly be solved by designing a system that learns the rules that result in different difficulties, from interactions between real-life robots and different environment. However, a rule-based system where the rules are designed by an expert will generate more predictable maps with fewer odd features that would not appear in the real world, because the rules can be controlled.

The tool and generator use a low resolution, where a single grid unit is an entire hallway width and rooms always fill a number of those grid units exactly. Changes at a higher resolution, like moving or placing objects inside a room or hallway can not be achieved without creating an entirely new room or hallway with that change incorporated. This low resolution has the advantage of limiting the amount of possible actions to perform, because the only ones are placing a hallway or room on one or more of the grid positions. Limiting the amount of actions makes the generation process easier, because there are not infinitely many possible actions to perform during generation. A downside of a low resolution is that the range of possible environments that can be generated is also limited. This means that there is a trade-off between controlling the generation process and generating diverse maps. For this paper, the low resolution of the existing World Generator is kept, because if a difficulty measure can be encoded using rules on a low resolution grid, a higher resolution will surely allow for difficulty to be encoded, likely even better.

There are two more trade-offs to be considered when building the indoor map generator. In order to generate diverse environments different room and hallway layouts should be generated, even when the user set the same difficulty. Also, rooms are annotated with a difficulty measure, in order to put the right rooms in a map with a particular difficulty, by minimizing the error between the annotation and the user-set difficulty. However, just minimizing the error would result in one or few rooms being the best pick to place in the map and there would be little variation between maps of the same difficulty. To solve this, the generator assigns probabilities to rooms for being picked to be placed on the map, which are higher when the maps annotated difficulty is closer to the maps difficulty. The trade-off is that assigning a higher probability to rooms with an incorrect difficulty will result in more varying generated environments, but those environments are also less likely to have the exact difficulty that the user wants. The equations used in the generator are shown in the application section. And while those are functional, they may not capture this trade-off the best.

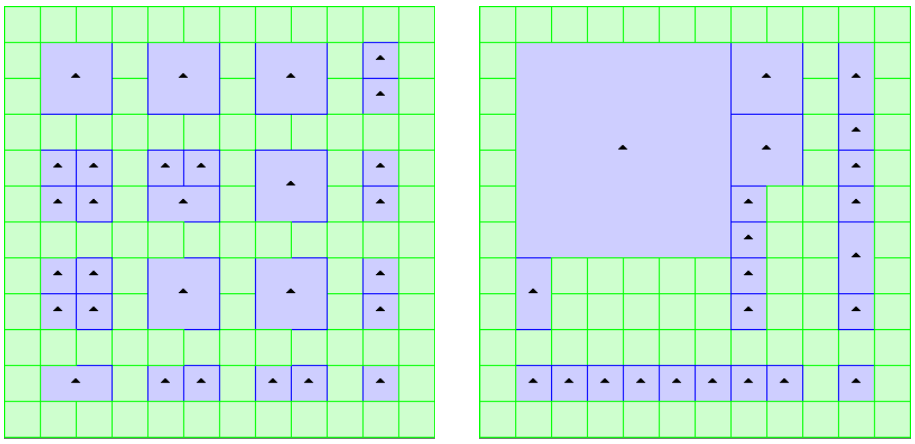
The last trade-off is between user control over the map and matching the user-defined difficulty with the generated map. When a user is allowed to control a parameter of the generation process, this parameter can not be used to adapt the map to the right difficulty. However, allowing a user control means he is able to steer the generation process in order to generate environments that are particularly useful to him. An example of this trade-off is: when using the existing generator in the World Generator tool, the user could change the distance between hallways in the generated map. However, as explained above, variations to this distance can mean that the generated map better matches the desired difficulty. So for the adaptive generator, the user is not able to change that parameter. More generally, the user can not change any parameter of the generation process, except for the difficulty and which rooms are allowed to be placed in the map. However, when the map has been generated, the user is free to change the map by adding, moving or removing rooms and hallways. This does mean that if the user decides to change the map, the difficulty of the resulting environment is not guaranteed.

4 Application

As previously mentioned, the generator is an adaptation of an existing tool for USARSim called World Generator [1]. This tool is written in Java and provides a valuable base for the generator to be build upon. Firstly, the World Generator dictates a grid structure that the generator can build its map on. More importantly, the tool includes processes to convert the grid map to a plain text file that can be converted to a proper 3D environment file, usable by USARSim. The tool also already includes a variety of pre-made rooms that can be placed in the map by the generator. Finally, the World Generator already has a simple map generator, whose rules can be expanded to build an adaptive map generator.

To create the adaptive generator, five adaptations to the existing generator have been made: variable distance between hallways, open area creation, variable doorway probability, adaptive room placement and cycle prevention. All of these changes depend in some way or another on the difficulty that is set by the user before the map generation begins. The user is required to set the difficulty to a value between 0 and 10. There are several reasons for constraining the range of values the difficulty can take. Firstly, by constraining the range, the rooms' difficulty annotations can be in the same range and the annotation can be directly compared to the user-set difficulty. Another reason is that by constraining the range, the difficulty value can be directly used in probability equations, and the probability will never exceed 1. Lastly, by constraining the range of difficulty values, there is a clear minimum and maximum difficulty the generator is capable of generating.

The first of the adaptations is a variable distance between hallways. The distance between hallways was set to an arbitrary value in the unmodified World Generator tool. The adaptation consists of increasing the distance between hallways in both directions (independently), when the difficulty increases. This adaptation is illustrated in Figure 2 with the lowest and highest difficulty, respectively. The distance between hallways is calculated independently for horizontal and vertical hallways using the following equation:
$hallwaySpacing = (0.06 \times difficulty + 0.2) \times size + randomGaussian \times 0.1 \times size$


(a) lowest difficulty (0), where the distance between hallways is small.

(b) highest difficulty (10), where the distance between hallways is large.

Figure 2: A grid-view of maps with varying hallway layouts.

, where $difficulty$ is the user-set difficulty (range: 0-10), $size$ is either the height or width of the map in grid units, depending on if the vertical or horizontal distance between hallways is calculated. *randomGaussian* is a random number drawn from a standard Gaussian distribution. The random deviation is used to ensure variation in maps of the same difficulty, and the deviation is larger with larger maps. *hallwaySpacing* is kept within a range of 3 to $size$, to ensure there is enough space between hallways for the generator to place rooms and there are always at least two hallways per direction.

There are several causes why the difficulty could increase when the distance between hallways increases. The first is that increasing the distance between hallways in, for instance, the horizontal direction, results in less intersections with the vertical hallways. Less intersections on the vertical hallways means there is more distance between intersections, where the robot can build up a larger odometry error. Another cause of increased difficulty as a result of longer distance between hallways, is that bigger rooms can fit in the empty space between hallways. Bigger rooms can have a larger complexity, by containing more objects or longer paths to traverse, for instance when the room is a large maze (see Figure 3). Another increased difficulty is created when large rooms are placed and a large portion of empty space is left after the rest of the column is filled with smaller rooms. This empty space is not filled by rooms because the generator moves to the right a number of columns equal to the largest room that was placed in the current column, when it reaches the bottom of the current column (see rule *roomRule*). Figure 2(b) shows how the second adaptation fills this empty space with hallway pieces (the light-green areas), thus creating large open areas with no distinguishable objects. A robot inside one of these large open areas likely has a problem with localizing itself, due to the walls being on the far end of the sensor range or beyond and the fact that there are no identifiable objects inside the area.

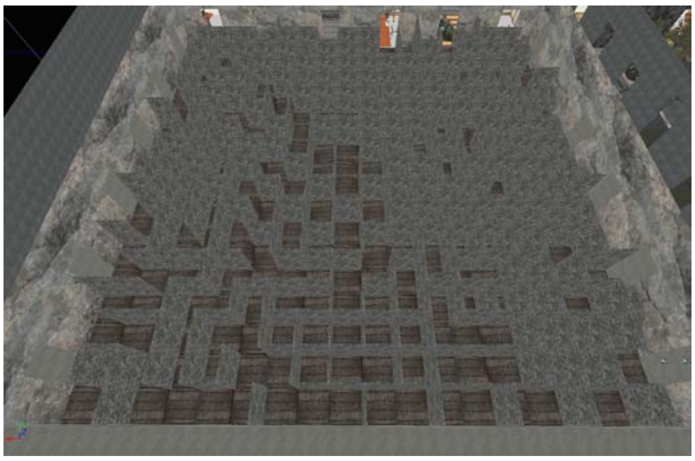


Figure 3: A large (6x6 grid units) pre-made maze room.

The third adaptation is a variable doorway probability. The unmodified World Generator tool already places doorways on the sides of rooms according to a probability and this probability is made adaptive to the difficulty level. The reasoning behind this adaptation is that having more doorways to a room on average, results in more small-scale cycles, where the robot can enter and exit the same room through different doorways. This can result in the same room being mapped from different directions and the two views of the room being misaligned on the map. So, a lower doorway probability is assigned to a map with a lower difficulty level, according to this equation:

$$doorwayProb = \frac{difficulty}{16} + 0.375$$

This equation ensures that the probability of placing doorways is never 0, because that would make every room inaccessible.

The next adaptation changes the random room placement of the unchanged World Generator tool. In order to adapt the map to a difficulty level, the rooms inside the map also need to be of a difficulty corresponding to that level. The adaptation consists of annotating the rooms that can be placed inside the environment, with their own difficulty. The rooms are assessed, ideally by an expert in the field of robot mapping, and given a difficulty value in the same range as the difficulty values the user sets for generated maps. When it comes time to generate a map, a probability of being placed in the map is assigned to each room, which corresponds with how well the room's difficulty matches with the map's difficulty. The equations that calculate this probability are as follows:

$$ISE(room) = \frac{1}{(difficulty - diff(room))^2 + 1}$$

is calculated for each room, where $diff(room)$ is the difficulty annotation of $room$. Then the probability is calculated as:

$$P(room) = \frac{ISE(room)}{\sum_{r}^{rooms} ISE(r)}$$

, where *rooms* is the list of rooms, which can be found in Appendix A, along with the difficulties. The $+1$ in the first equation prevents a possible division by zero and the second equation turns the inverse squared error (ISE) into a probability. For each empty space in the map that requires a room to be placed inside it (see *roomRule* in previous section), each room has a chance of $P(room)$ of being placed in that space. Using probabilities ensures that maps of the same difficulty are still varied, because all rooms have at least some probability of being placed. This method also ensures that on average, the difficulty of the rooms inside an environment is the same as the difficulty of the environment, as set by the user.

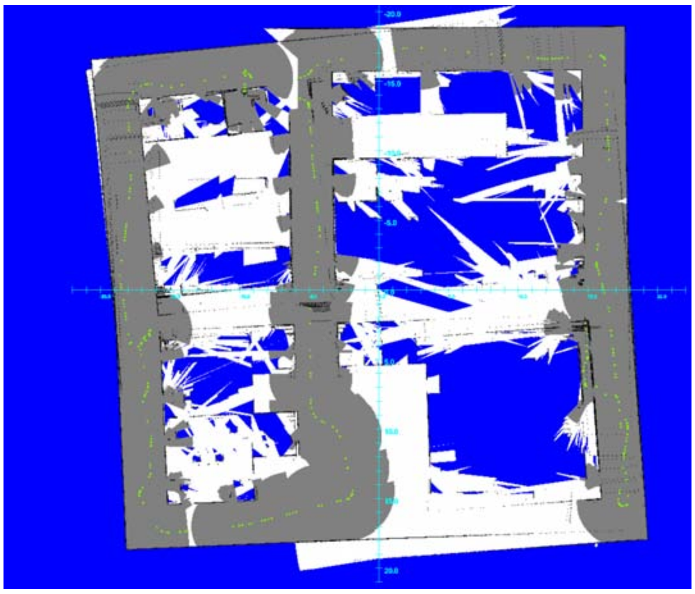


Figure 4: A map created by two robots during an experimental run through a generated environment.

The last adaptation adds a new rule to the generation process, as opposed to changing the existing rules to become adaptive. This rule is designed to prevent cycles in a generated map. A cycle in a hallway can result in a misaligned hallway on the robot's map. The robot builds up an odometry error when moving through the cycle, that is insignificant at a local level, but when the robot completes the cycle the odometry error is large enough that the cycle is misaligned and the map reflects that error [7]. Figure 4 shows such a misalignment, where a robot moved around the left side of the map (the green dots show the path), returning to its starting position and the top horizontal hallway is misaligned as a result of the cycle. The new rule blocks off horizontal hallways at specific positions, so the hallways do not form cycles anymore. This rule is applied with a probability of:

$P = 1 - \frac{difficulty}{10}$

At low difficulties, there is a high probability of cycles being blocked off, and that probability drops linearly with increasing difficulty. The rule itself looks like this in pseudo-code:

```
preventCycles(horizontalSpacing, verticalSpacing)
for every column dividable by horizontalSpacing - 1
    for every row dividable by verticalSpacing
        remove the hallway piece on that grid position
    place back one hallway piece on this column at random
```

This rule is inspired by the *hallwayRule* but instead of placing hallway pieces, it removes them to block of the horizontal hallways. In order to ensure that the entire map is still accessible, one hallway piece is placed back per column. Figure 5 shows the effect of this rule (note the white grid positions), when compared to Figure 2(a) for instance. When this rule is applied it is no longer possible to move in a cycle around the map, when moving through hallways. However, because rooms can have multiple doorways, moving through a room can still result in a cycle. But since the doorway probability is lower at the difficulties that this rule is likely to be applied, it is less likely for rooms to have multiple doorways that connect different hallways together.

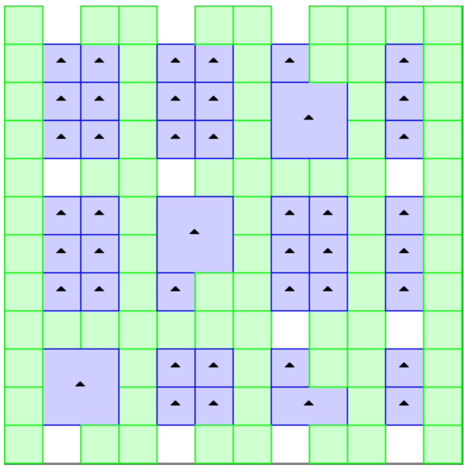
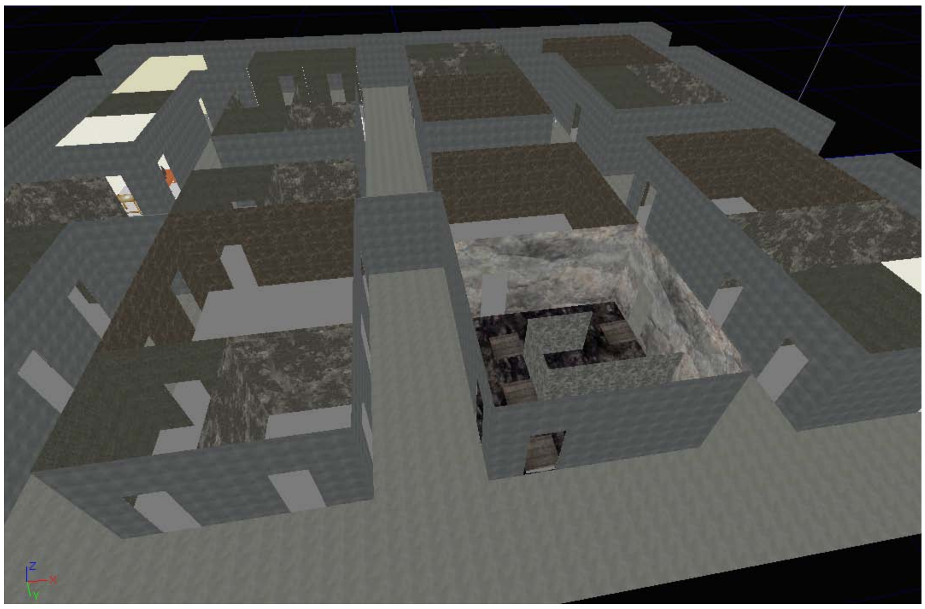


Figure 5: A grid-view of a generated map with 'cycle prevention'.

Together, all the above adaptations of the World Generator tool result in a map generator that adapts to a difficulty that the user sets, without requiring further user intervention. It generates diverse maps with clear differences in geometry between different difficulties. Figure 6 shows just how different a map of a low difficulty is compared to one with a high difficulty, with a clear difference in room complexity, as well as in general layout.


(a) generated with a low difficulty (1).

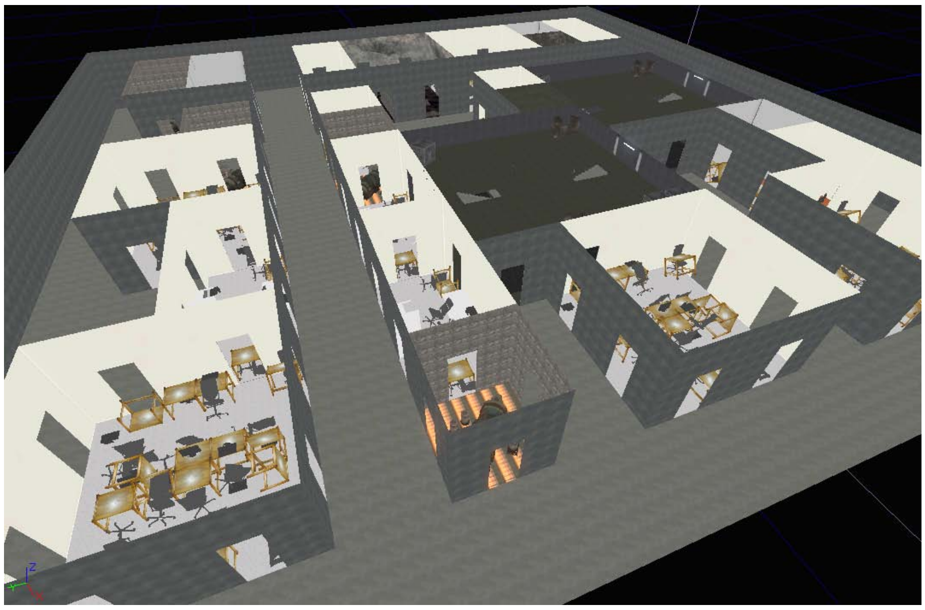

(b) generated with a high difficulty (9).

Figure 6: An eagle-eye view of generated maps with different difficulty settings.

5 Experiments

Ideally, the difficulty of the generated environments is evaluated by having a quantitative measure of the quality of a robot's map, after performing a run through the generated environment. Then, the quality of a map decreases as the difficulty increases, if the generator correctly adapts to the difficulty measure. However, determining the quality of a map, even in a simulated environment where the ground truth of the environment is known, is an open research issue (A. Visser, personal communication, June 16, 2011). And since this issue is beyond the scope of this research, a qualitative assessment of the performance of the map generator is performed.

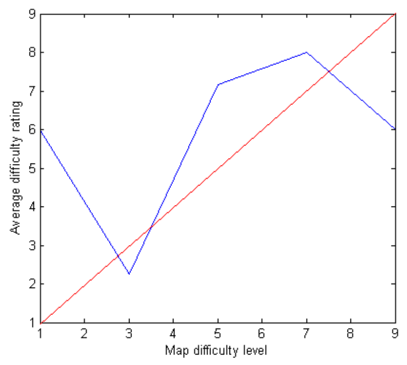
To get a representative assessment of the generated maps, several USARSim operators with different experience levels were asked to run simulations on maps generated with varying difficulty levels. Maps were generated with five difficulties (1, 3, 5, 7 and 9), with two different maps per difficulty, for a total of ten maps. The participating operators were asked to run simulations on three maps with different difficulties. They were then asked for detailed feedback on how difficult they perceived each environment to be, when mapping the environment. They were also asked more specifically what elements in the environment made the mapping either difficult or easy. The feedback form that was used for these experiments can be found in Appendix B. The operators were free to use the operating environment and settings of their choice, as well as choosing how many robots they used to map the environments. One reason for not specifying a simulation configuration, is that the focus is on the environment and the map's specifics can be assessed regardless of the exact simulation configuration. Another reason is that the assessment of the difficulty can vary between operators, so operators are compared to themselves by comparing their feedback on the different maps they evaluated.

If the participants in the experiments rate the maps in the same order of difficulty as the difficulty levels at which they were generated, then the map generator encodes at least some element in the maps that gets more difficult to map as the difficulty rating of the environments themselves increases. If on top of the rating, the operators also note the effects of the adaptable rules to be the cause of the difference in difficulty, then the rules can be considered to correctly adapt to the difficulty measure. If the participants find no distinguishable difference in difficulty between the maps, then the map generator is not capable of adapting to a difficulty measure. If the operators ratings are in reverse order compared to the supposed map difficulties, then an adaptable rules can be reversed to show the behavior at the opposite end of the difficulty scale, in order to make the map's difficulty properly scale. This does however require that the rule that caused the incorrect behavior is identified by the operators.

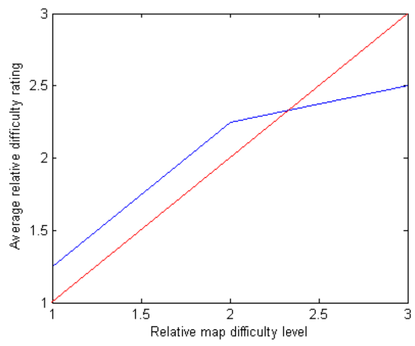
6 Results

Six operators provided feedback on one or three of the ten generated maps each, with experience levels ranging from experimental experience as an operator to experience as an operator at multiple competitions. All operators used the same type of robot for mapping the simulated environments, namely the P3AT robot present in the USARSim package. The P3AT simulated robot is modeled after the real-world Pioneer 3-AT four-wheeled robot, with a laser range scanner for mapping purposes and a camera for operator convenience (so the operator can navigate using the camera image instead of the map). Some operators choose to map the environments using one robot, while others navigated two robots simultaneously through the environment. There was a 20 minute timer for the simulation, but the operators never reached this limit, instead reaching a natural ending when either the robot(s) got stuck or the robot(s) ran out of battery power.

The first of the results is the overall difficulty rating that each operator assigned to each environment they mapped. These difficulty ratings have been plotted both using absolute and relative values, in Figure 7. The absolute graph plots the difficulty with which the maps were generated against the average difficulty that the operators assigned to those maps. For the relative graph, only the feedback of operators that finished runs on three maps were processed. The three maps that each operator evaluated were ordered based on the difficulty that they were generated with and then they were given a value of 1, 2 or 3 based on the order of the difficulty ratings that the operator assigned to these maps. Then the average value of each of the three relative difficulties was plotted.



(a) Absolute difficulty level against difficulty rating.



(b) Relative difficulty level against difficulty rating.

Figure 7: Graphs of the absolute and relative difficulty ratings, against the difficulty level the maps were generated with.

While the difficulty rating is a quantitative value and as a result can be plotted in graphs, it should be noted that the sample size is small (14 samples spread over 5 difficulty levels) and the plots offer no guarantee of statistical significance. They do however give some indication of the performance of the map generator.

The red line in both graphs shows the best possible result, where the difficulty that the operators assigned to the maps equals the difficulty level the maps were generated with exactly. The blue line is the actual result, which in the absolute graph shows that the lowest and highest difficulty levels are not rated as such by the operators.

Maps that were generated with a difficulty of 1 are rated 6 on average, which is significantly higher and it is even higher than the difficulty rating for maps with difficulty 3. An experienced operator that has participated in multiple competitions, notes that the dead ends created by the cycle prevention rule result in mapping difficulties, as written in his feedback form: "The deadends at the corners made the mapping difficult..." and "Only one central crossing, which forces you to repeately come back at the same location...". He rates the difficulty of the map with a 7. However, comparing Figure 8(b) and Figure 8(c) (in Appendix C) for instance, shows that the cycle prevention in the second map does result in fairly straight and correct hallways, while the cycles still present in the first map resulted in a significant offset between the two halves of the top hallway. Another operator with experience at a single competition, notes the lack of doorways in rooms as a difficulty when mapping an environment generated with difficulty 1: "No doors found: need to leave through the door i entered.", is his response to the question which rooms were difficult to map. Turning, especially in a small space, is a time-consuming and difficult job for a robot operator and it results in an odometry error building up, says another operator (O. Formsma, personal communication, June 20, 2011).

On the other end of the spectrum, the maps that were generated with a high difficulty (9) were not rated as being very difficult by the operators, as those maps got an average rating of 6. One operator with experience as an experimental operator, notes in his feedback form that the many doorways to each room, as well as the long hallways, results in an easy to map environment: "Long straight corridors and square rooms with openings on several sides made it easier to map.". The probability for many doorways per room is significantly higher at higher difficulties, because a rule lowers the amount of doorways at low difficulty to avoid cycles in the environment. The hallways are also not cut off at higher difficulties, because the rule that prevents cycles has a very small chance at activating at higher difficulties. The same operator did also note that the objects inside rooms were a primary cause of difficulty in the map and rooms with obstructing objects have a higher probability of being placed on a map with a higher difficulty, due to their higher difficulty annotation (note the office-type rooms in Appendix A).

Between the extreme difficulties however, the ratings by the operators matched the desired difficulty levels to a reasonable degree, as seen in Figure 7(a). This is backed up by the feedback given by the operators, who consistently noted the contents of the rooms as a reason for the maps to be either easy or difficult to map. One operator with experience in a single competition notes that a map with difficulty level 7 is difficult specifically because of the contents of the rooms: "Although a general overview of the map is easily found by following the corridors, exploring the rooms turned out to be exceedingly difficult.". This comment also shows that hallways were a predictable element in the environments and following those gave a very good idea of the overall layout of the entire map, which was also noted by other operators. The same operator also notes that on an easier environment (with difficulty level 3), the rooms are the reason for being easier to map: "Relatively large rooms with little furniture are very easy to navigate.". This comment also shows that besides the contents, the size of the room is also a factor in its difficulty, as larger rooms are noted to be easier to map, likely because the robot can more easily navigate a larger room.

Lastly, the relative difficulty graph in Figure 7(b) shows that on average, the operators rate the easiest map with the lowest difficulty, the second easiest map with a higher difficulty and the hardest map with the highest difficulty. However, the relative ratings of the hardest and second hardest maps (3 and 2 on the graph) are close together, meaning they are often rated the other way around by the operators.

7 Conclusion

In this report, the challenge was to adapt an indoor map generator of USARSim environments to generate maps according to a user-defined difficulty level, and the generator should generate maps that were roughly that difficult to map for a (simulated) robot. The focus was on the following research question: Can the rules used in an algorithm for generating indoor environments be used to make the generator adaptive to a difficulty measure for a robot mapping task? The adaptive map generator was implemented by adding five simple additions to an existing generator, to make the generated maps different, based on the difficulty level. The adaptive rules resulted in significant differences to the layout of the maps of different difficulty levels. However, operators that mapped the generated environments found that the hallways were a predictable structure in most of the generated maps. And while it is usual that hallways have a clear pattern, even in real-world indoor locations, the generator could only generate one type of hallway layout, despite adapting that layout by scaling it based on the difficulty level. This predictable structure resulted in the maps being noticeably easier to map by operators, which likely counteracted the effects of the other rules to make the maps more difficult (when the difficulty level was high).

There were two specific adaptive rules, namely cycle prevention and doorway probability, that were designed to prevent cycles in the environment at low difficulties. And while they did succeed in that respect, they had the unforeseen effect of making the map more difficult, by creating dead ends and rooms with few doorways that required extensive maneuvering by the robot operator to back out of. So while these rules did prevent cycles and the potential mapping errors associated with them, they introduced a navigation difficulty, that had a negative effect on the map. This shows that the rules encoded the abstract notion of difficulty very naively, not taking into account the complexity of difficulty and the many ways it can manifest itself.

The room placement based on the room difficulty annotation did encode difficulty in a reliable way, as nearly all operators noted that easily mapped rooms could be found in maps with a low difficulty and harder to map rooms, mostly due to obstacles and hazards like water, could be found in maps with a high difficulty. However, the rooms were annotated with their own difficulty level by someone with insight, which means the annotations encode the knowledge that this person has about mapping difficulties and the room placement rule simply uses those annotations to place rooms with the correct difficulty in the environment. The generator is not able to somehow derive the difficulty of a room, based on the room's characteristics, but is simply handed the difficulty of each room and places the room with the right difficulty in the map.

In conclusion, the simple adaptive rules are not capable of reliably generating environments with a specific difficulty for robot mapping, as the feedback of the operators proved by not consistently assigning the highest difficulty rating to the map with the highest difficulty level, for instance. The rules do however significantly influence the difficulty of the map, as the operators note differences in difficulty that they attest to the effects of the rules. This influence is not reliable however, as the same rule can both result in mapping difficulties, as well as prevent them.

8 Discussion

A first point to note is that the generator used rules that were derived from known difficulties with robot mapping [7]. By using this knowledge, the generator is build to specifically generate difficult maps for the current generation of mapping algorithms. On top of this, high level, low resolution (room-based) rules were used, which means the generator shows little to no emergent behavior. Together this means that the generator likely does not generalize well to, for instance, future mapping techniques, as the rules are limited in their scope and made to take advantage of the current known issues with mapping. However, if these rules were able to encode difficulty, it would have opened the way for more complex rules to refine the generation process.

Another observation to note is that experienced operators were well able to identify the causes of mapping difficulties, after they had finished a simulated run through the environments. Those observations by the operators could have shaped the adaptive generation rules, if these operators were brought in to the project during the rule designing process, as opposed to being brought in solely to evaluate the generated maps. This suggested approach would bring a potential conflict to the research however, because when the same operators (or similar operators) help design the generation rules and then evaluate the resulting generated maps, they could see their own rules in effect in the generated maps and would not be able to objectively evaluate these maps.

Something that became apparent when reading the feedback on the forms the robot operators filled in, was that some of the operators based their assessment of the difficulty of a generated map on the difficulty to navigate the environment, as opposed to the difficulty to map the environment, while the latter of the two was the difficulty with which the environment was generated. This misunderstanding was likely caused by the way the questions were formulated in the feedback form (Appendix B). While navigation and mapping difficulties are distinctly different, it should be noted that navigation difficulties can cause mapping difficulties, as robots that are not able to navigate an environment properly, are also likely not able to map the environment properly, if only because the environment is not completely accessible for mapping.

Lastly, a question should be posed about the simulated environments and the need for these environments to mirror the real-world. Realism in simulated environments for robot mapping is a good criteria for the quality of these environments, as simulated robots are used to test (and train) algorithms that ultimately should work on real robots in real-world locations. This means that simulated environments should consist of a representative sample of real-world locations and should match the characteristics of these locations as best they can. It is easy to create simulated environments that contain unrealistic challenges for robots when navigating or mapping the environments, but the algorithms should not be trained on unrealistic challenges, as it puts unrealistic demands on the capabilities of the algorithm that could take away from realistic challenges that the algorithm should be able to handle. This means that any adaptive map generator for robot simulation purposes should generate maps within a realistic range, instead of creating artificial challenges that a robot would never face in real life.

9 Future Work

This research introduced the concept of adaptive map generation, to generate functional indoor environments within a robot simulation environment. Future work could build upon this concept and expand it to encompass different functional needs or refine the concept to become more practically usable. First and foremost, the generator could benefit from being expanded with more complex rules. Using L-systems for instance to generate hallways could generate much more diverse and challenging map layouts. L-systems have previously been used to successfully generate diverse city street plans [6], as these systems are very powerful recursive processes.

Using rules with a higher resolution (object-based, as opposed to room-based) could also be a good direction for future work, as a higher resolution gives the designer of adaptive rules the ability to influence the environment at a smaller scale and as a result the designer can also encode difficulty in smaller structures in the environment. An example of this would be to create rules that can rearrange objects inside a room to narrow or widen the paths through the room, to increase or decrease the difficulty of the room.

Another possible direction for future research is to generate different environments and create rules to combine these. For instance, generating both an outdoor and several indoor environments, and then combining them to create a realistic scenario where the robot can move from the outside to the inside environment and can move to different floors inside. This also creates more possibilities to encode difficulty for mapping, for instance by limited the number of ways the inside can be reached from the outside.

Another way to refine the generation rules is to look at specific robots and their capabilities and then designing rules around the specific difficulties the particular robot can have while navigating or mapping. This will make the generated maps specific to a particular robot, but it will likely also make it easier to generate maps with a specific difficulty, because the specific mapping issues of that robot are known. It will also likely result in a more versatile generator, that is able to generate specific types of maps for specific types of robots. The current generator does not generate environments that are properly navigable by flying robot, for example, and a generator that takes into account the characteristics of different types of robots will be able to generate different maps that are navigable by different types of robots.

Another possibility for future work is to research if the map generator creates realistic environments and if rules have an inherent 'realism' property, where the rule itself dictates the generation of realistic or unrealistic structures in a simulated environment. This could result in more insight in how to design a map generator that creates realistic environments, with realistic challenges for a simulated robot. This research could also show whether generating more difficult environments also results in generating more realistic environment, or if more difficult environments create unrealistic challenges for robots.

A practical suggestion for future work is to design more varied rooms and hallway pieces for the current map generator. The current generator will likely benefit from having a larger selection of rooms to place in the environment, because it can then generate more diverse maps as well as match the user-defined difficulty better, because more rooms with varying difficulties are available for placement. If the generator can also place different hallway pieces, for instance with objects blocking the hallway to a certain extent, it would be able to diversify the element of the maps that is currently the most predictable: the hallways. Introducing dynamic objects into the rooms and hallways could also result in more diverse and realistic maps. Dynamic objects like slamming doors, moving people and smoke create a different set of difficulties for the robot, while mapping his surroundings.

Instead of a rule-based generator, a different approach to generation based on simulation could also be researched in the future. Existing literature [4] has already shown that for a simple video game, simulating the player's movement can give the generator the data required to adjust the map to be more or less difficult. While simulation of a robot is not nearly as straightforward as a video game character, it would allow for the generator to use the simulation data to adjust the map to be more or less difficulty, as opposed to someone having to interpret the difficulties a robot has while mapping, in order to design generation rules.

Acknowledgments

I would like to thank my supervisor Arnoud Visser, for his support and assistance. I would also like to thank all the USARSim operators that participated in the experiments for offering their expertise and sacrificing their time: Nick Dijkshoorn, Okke Formsma, Nguyen Nhu Hieu, Julian de Hoog, Sevaztian Soffia Otarola and Arnoud Visser.

References

[1] Brent, T., Carlson, S., and Dutko, J. *USARSim World Generator tool.* http://usarsim.sourceforge.net/wiki/index.php/World_Generator, June 2011.

[2] Carpin, S., Lewis, M., Wang, J., Balakirsky, S., and Scrapper, C. Usarsim: a robot simulator for research and education. In *Proceedings of the IEEE International Conference on Robotics and Automation* (2007), ICRA, pp. 1400–1405.

[3] Carro, R., Breda, A., Castillo, G., and Bajuelos, A. A methodology for developing adaptive educational-game environments. In *Adaptive Hypermedia and Adaptive Web-Based Systems*, P. De Bra, P. Brusilovsky, and R. Conejo, Eds., vol. 2347 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006, pp. 90–99.

[4] Compton, K., and Mateas, M. Procedural level design for platform games. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment International Conference* (2006), AIIDE.

[5] Deussen, O., Hanrahan, P., Lintermann, B., Měch, R., Pharr, M., and Prusinkiewicz, P. Realistic modeling and rendering of plant ecosystems. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 275–286.

[6] Parish, Y., and Müller, P. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 301–308.

[7] Thrun, S., Burgard, W., and Fox, D. *Probabilistic robotics.* Intelligent robotics and autonomous agents. MIT Press, 2005.

[8] Tutenel, T., Bidarra, R., Smelik, R., and de Kraker, K. Rule-based layout solving and its application to procedural interior generation. In *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation* (Amsterdam, NL, 2009), 3AMIGAS, pp. 15–24.

Appendices

A Room List & Difficulties

The table below lists the names and sizes of all pre-made rooms that can be placed inside a generated map by the map generator. Also shown is the difficulty of each room, as well as a short explanation of the reasoning behind the difficulty rating.

Room name (scale in grid units)	**Difficulty**	**Elaboration**
dark (1x1)	0.5	A tiny empty room with no mapping difficulties.
empty (1x1)	0.5	See dark (1x1).
bisquare (2x2)	1.5	An average-sized empty room with little mapping difficulties.
maze (2x2)	2.5	Simple maze with some uneven ground.
Maze (smaller) (2x2)	4.0	Small dense maze with uneven ground.
Boiler Room (1x1)	6.8	The boiler takes up just enough room to block the robot's path.
Dragon Room (2x2)	5.5	Some obstructing objects, but still some space to move left.
Warehouse (2x2)	4.0	Plenty of room to move with some containers taking up space.
Futuristic Reception (2x2)	3.5	Reception in the middle of the room with plenty of space around it.
Water Room (3x3)	9.0	Hazardous water with narrow walkways and stairs blocking the view.
Maze2 (6x6)	9.5	Very large maze with narrow paths and uneven ground.
Computer Lab (2x2)	5.0	Desks and chairs make navigating trickier, but not impossible.
Single Office (1x1)	5.25	Same as Computer Lab, but with less space to move.
Multi Office (2x1)	5.75	Desks and chairs block movement slightly and no walls.
CompLab_unlit_destroyed (2x2)	7.3	Fallen chairs, desks and computer screens make navigating hard.
Cubicle_unlit (4x4)	6.25	Large room with cubicles but also reasonably sized walkways.
Cubicle_unlit_destroyed (4x4)	8.0	Slanted cubicle walls and fallen objects make navigation very hard.
Single_office_north_destroyed (1x1)	7.0	Fallen desks and chairs make navigation hard.
MultiOffice_tight_unlit_dest (2x1)	7.2	Fallen objects split the room in two, allowing no robot through.
victims (1x2)	2.0	Empty space, with room for victims

B Feedback Form

The form below was used to poll the operators that participated in the evaluation of the map generator, about their opinion on the difficulty of the map.

USARSim Robot Mapping Feedback Form

Name:

Experience level as Operator (none, experimental, single competition, multiple competitions):

Filename of the map/environment:

Software package used for running the simulated mapping (e.g. Amsterdam Oxford Joint Rescue Forces' USARCommander Rev. 2215):

Connection model (Direct, PropegationModel, DistanceOnly, ObstaclePropegationModel):

Number and type of robots (e.g. 1x P3AT):

Overall, how would you rate the difficulty of the environment in terms of mapping (scale 0-10):
NOTE: 0 difficulty means the environment does not include any aspects that 'force' a mapping error on the robot. 10 difficulty means the map resulting from the mapping process can not be properly navigated, due to excessive mapping errors.

What aspect(s) of the map contributed most to the difficulty rating given above:

Were there one or more rooms in the environment that proved to be particularly difficult, and if so, what do you think made the room(s) difficult to map:

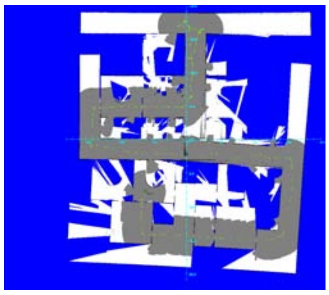
Were there one or more rooms in the environment that proved to be particularly easy, and if so, what do you think made the room(s) easy to map:

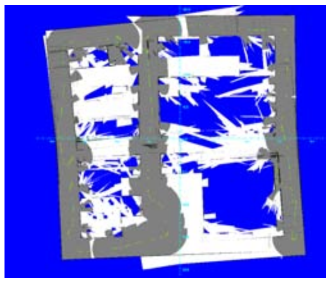
What patterns/regularities did you notice in the way the map was laid out (e.g. hallways/dead ends/doorways/etc.), and what effect did each of these patterns have on the difficulty of the map, in your opinion:

Are there any other aspects of the environment that, for whatever reason, stood out, that you wish to note?

C Operator Maps

The maps below are all the maps that the operators created while evaluating the generated environments, including information about the operator, difficulty level of the map and number of robots used. The maps are in the order the operators performed them in. All operators used the four-wheeled P3AT robot for their simulations and operated the robot(s) using Amsterdam Oxford Joint Rescue Forces' USARCommander rev. 2215 tool, with connection model: Direct.


(a) map765 with difficulty 3 mapped using 1 robot


(b) map354 with difficulty 7 mapped using 2 robots

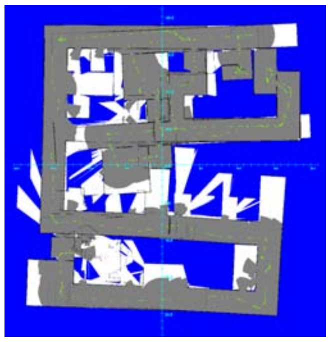
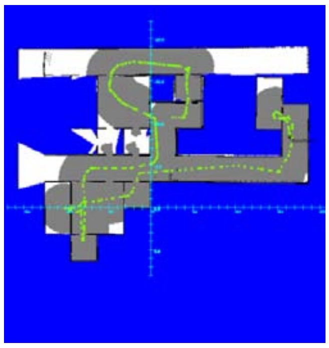
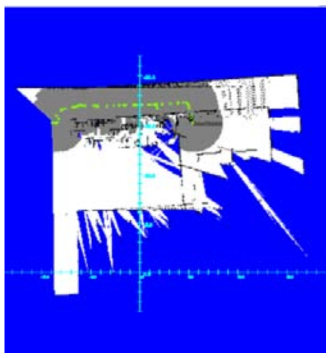

(c) map674 with difficulty 1 mapped using 2 robots

Figure 8: Operator: Okke Formsma. Experience: Single Competition


(a) map674 with difficulty 1 mapped using 1 robot


(b) map238 with difficulty 7 mapped using 1 robot

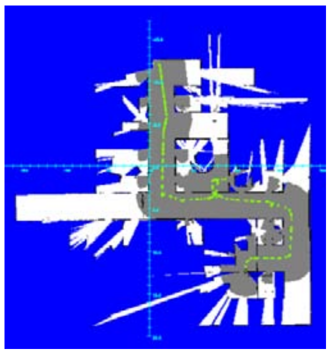
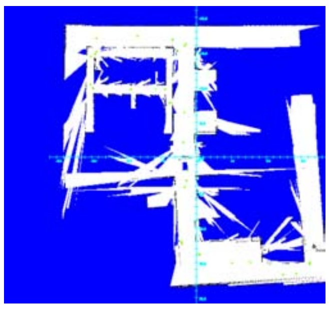
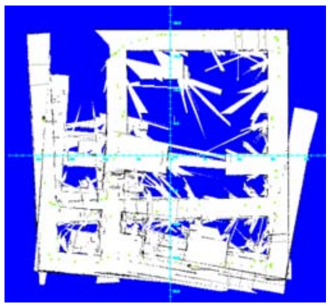

(c) map165 with difficulty 5 mapped using 1 robot

Figure 9: Operator: Nick Dijkshoorn. Experience: Single Competition


(a) map364 with difficulty 5 mapped using 1 robot


(b) map104 with difficulty 9 mapped using 2 robots

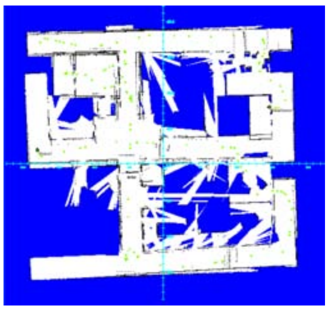
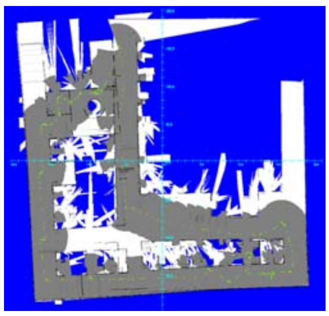
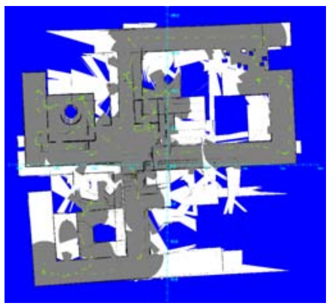

(c) map673 with difficulty 3 mapped using 2 robots

Figure 10: Operator: Julian de Hoog. Experience: Multiple Competitions


(a) map245 with difficulty 9 mapped using 1 robot


(b) map673 with difficulty 3 mapped using 2 robots

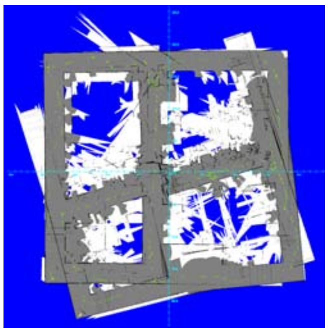

(c) map364 with difficulty 5 mapped using 2 robots

Figure 11: Operator: Sevaztian Soffia Otarola. Experience: Experimental

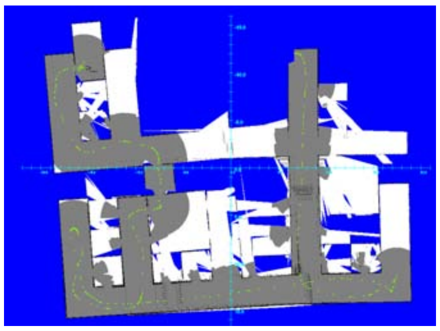


Figure 12: map347 with difficulty 1 mapped by Arnoud Visser (experience: multiple competitions) using 1 robot

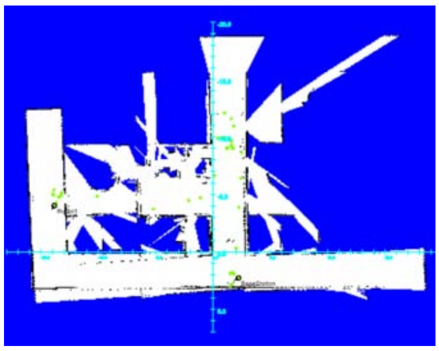


Figure 13: map765 with difficulty 3 mapped by Nguyen Nhu Hieu (experience: experimental) using 2 robots