Dynamic path planning of initially unknown environments using an RGB-D camera

Sara Gustafzelius

***Abstract*—In this paper an RGB-D camera was used with the goal to perform dynamic path planning in an initially unknown environment. Depth data from an RGB-D camera together with a discretizising algorithm is continuously used for maintaining an obstacle map of the environment which within the path planning algorithm D* Lite [S. Koening, 2005] is performed on the flight.**

Experiments were conducted on a Gantry Tau robot at the Robot Lab of the Department of Automatic Control, LTH. For discretization purposes we compare the use of Box Approximation and Signed Distance Function (SDF) for creating the obstacle map.

***Keywords*—Dynamic path planning, D*Lite, RGB-D camera, discretization, Box Approximation, obstacle map, SDF, re-planning.**

I. Introduction

There are currently many fields of applications where traversing non human friendly environments with unmanned vehicles can be highly useful. Further, as the use of, for example, Unmanned Aerial Vehicles (UAVs) increases it is not hard to see the benefit of not only unmanned but also automated unmanned vehicles.

In this article we demonstrate how path-planning and automatic re-planning can be performed using an RGB-D camera. We compare two different approches, Signed Distance Function (SDF) and Box Approximation, to efficiently discretisize depth data. Depth data from an RGB-D camera is used to create and continuously update an obstacle map of the environment. The obstacle map contains information of which coordinates can be traveled and not traveled. When a new obstacle is detected by the RGB-D camera the location of the obstacle is calculated and marked as untraversable in the obstacle map. The path planning algorithm is initially given a start and goal position and a first path is calculated using the obstacle map. When a new obstacle occurs on the obstacle map the path planning algorithm dynamically re-plans the local path part affected by the obstacle. The system flow can be seen in Figure 1.

We choose to use the path-planning algorithm D* Lite [S. Koening, 2005] that builds on Lifelong planning A* (LPA*) [S. Koening, 2002]. D* Lite plans the same path as the more complex algorithm Focused Dynamic A* (D*) [Stentz, 1995] by Stentz but is algorithmically different and easier to overview. The main idea of D* Lite is to do an initial search identical to the A* [P. E. Hart, 1968] algoritm and to start move along the calculated path. When an obstacle that affects the planned route is detected the subsequent search uses information from the previous searches to locally re-plan the path. D* Lite uses a priority heap to sort the vertices with decreasing travel cost.

A. Robertsson is with the Department of Automatic Control, LTH, Lund, Sweden.

E. Bylow is with Department of Mathematics, LTH, Sweden.

S. Yngve at Combine Control Systems AB, Lund, Sweden.

M. Stenmark is with the Department of Computer Science, LTH, Lund, Sweden.

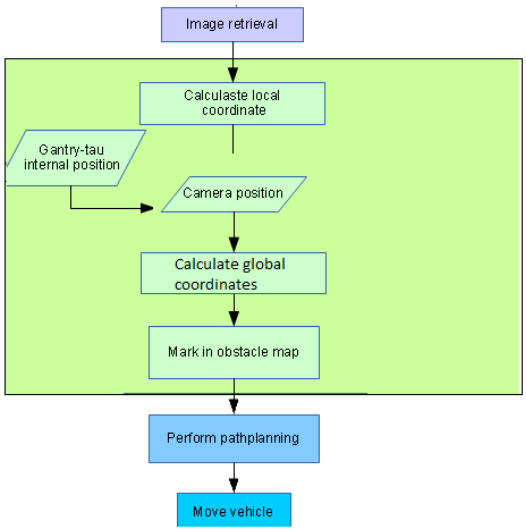

Fig. 1. Flowchart over the system blocks.

II. Updating the obstacle map

The main idea of the obstacle map is to avoid building a graph for the path planning and instead use a structure that both the discretization algorithms and the path planning algorithm can use. The obstacle map will be represented as a three-dimensional binary matrix.

In Figure 2 we see an environment in two dimensions seen from above. The environment contains two obstacles. We apply a grid in order to discretize the environment. The result can be seen in Figure 3 where every square in the grid is an index in the obstacle map and will be referred to as nodes in the path planning.

If a square contains an obstacle the corresponding index in the matrix will be set to one. This means that this node will be untraversable when applying the path planning algorithm. If a square is free from obstacles the corresponding index will be set to zero. For the path planning algorithm this means that this node will be traversable. The result of this an obstacle map filled with ones and zeros as in Figure 4.

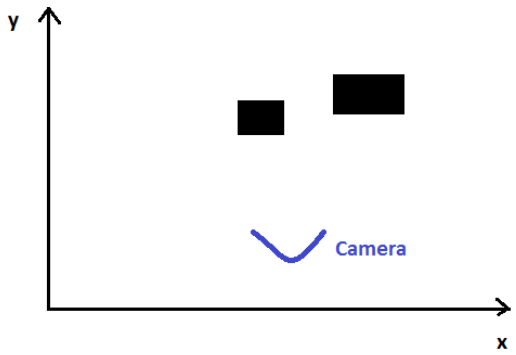

Fig. 2. An environment in two dimensions seen from above.

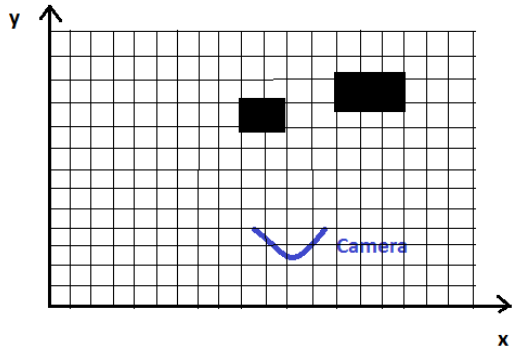

Fig. 3. The environment seen in Figure 2 when a grid is applied.

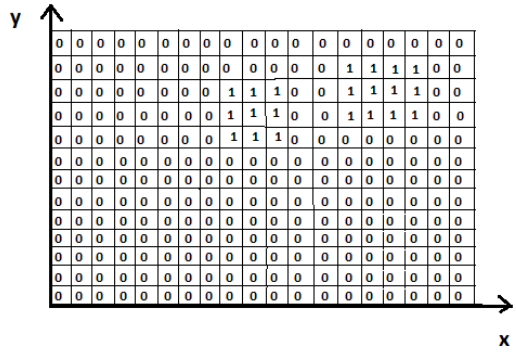

Fig. 4. The obstacle map corresponding to the environment in Figure 2.

A. Transforming depth data into global coordinates

The depth image data received by the RGB-D camera is transformed into local coordinates using the pinhole camera model.

$$X = (x, y, z)^T = (\frac{(i - c_x)z}{f_x}, \frac{(j - c_y)z}{f_y}, z)^T,$$

where $(i, j) \in I_d$, $z = I_d(i, j)$ and f_x, f_y, c_x and c_y are intrinsic camera parameters.

To be able to use data from different camera positions we need to transform the local coordinates into global coordinates. We use,

$$X_G = C \cdot X_L,$$

where X_L are the local coordinates, X_G are the global coordinates, and C is the camera matrix i.e., the position and rotation of the camera.

B. Discretization methods

1) Signed Distance Function: SDF is a function which gives the signed distance d between a point $X \in \mathbb{R}^3$ and the closest point X_s on a surface S. For two dimensions this can be illustrated as in Figure 5.

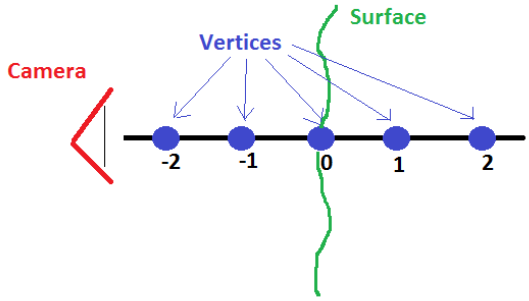

Fig. 5. SDF example in two dimensions.

We will use a voxel grid (see Figure 6) to symbolize the signed distances we get from the SDF. We want to try to project each voxel in the voxel grid onto the image plane using,

$$I_d(i, j) = (\frac{f_x x}{z} + c_x, \frac{f_y y}{z} + c_y),$$

where f_x, f_y, c_x and c_y are intrinsic camera parameters.

If the signed distance is more than or equal to 0 it means that there is a surface on or between the global voxel and the camera, and therefore we set the value of the global voxel to 1 [Bylow, 2012, p. 17]. If the signed distance is less than 0 it mean that there are no surfaces on or between the camera and the global voxel, and therefore we set the global vertex to 0 [Bylow, 2012, p. 17].

2) Box Approximation: In our other approach we will use a box grid (see Figure 7) to symbolize if a space in the room is free or not. If $D(i, j)$ *i.e.,* the local z value is greater than zero, this means that the local 3D coordinate X_L is located in front of the image plane. We want to try to project each pixel in the image with $D(i, j) > 0$ into the box grid using,

$$X = (x, y, z) = (\frac{(i - c_x)z}{f_x}, \frac{(j - c_y)z}{f_y}, z)^T,$$

where $(i, j) \in I_d, z = I_d$ and f_x, f_y, c_x and c_y are intrinsic camera parameters. Then we have to calculate in which box

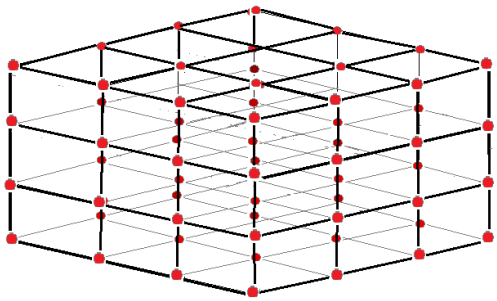

Fig. 6. Example of how a voxel grid can look like where each red dot is a voxel.

the 3D coordinate is located. Since we know the resolution and the size of the space in all dimensions we also know the exact boundaries of each box so that the box grid can be easily indexed to find the correct box using the global 3D coordinates. Finally, we just need to mark the correct box as occupied.

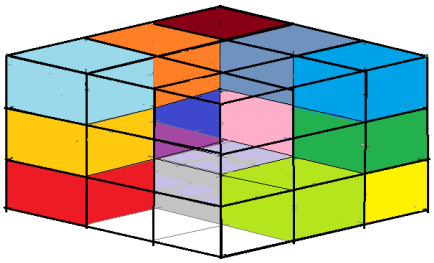

Fig. 7. Example of a box grid where every filled box is occupied and every empty box is free.

C. Path planning using the obstacle map

We implemented the D* Lite algorithm in three dimensions using an obstacle map.

Every index in the matrix represents a global coordinate and is considered a node. If a node is blocked and untraversable it has the value 1 and if it is free and traversable it has the value 0. If a node is blocked there is no edges to or from that node. If a node is free if got edges to all surrounding free nodes, at most eight edges.

Since every node represents a global coordinate the weight of the edges are only effected by how many dimensions it covers, this is showed in Figure 8. The cost of an edge along one dimension is 1. The cost of traveling in two dimensions is the cost of traveling along the hypotenuse i.e $\sqrt{2} \approx 1.41$. Finally the cost of traveling in three dimensions is the euclidean distance between the nodes $\sqrt{3} \approx 1.73$. For simplicity reasons we choose to scale the cost and use the values 10 for one dimension, 14 for two dimensions and 17 for three dimensions.

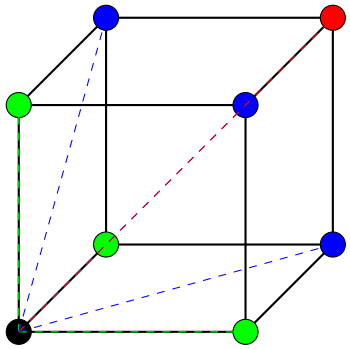

Fig. 8. This figure shows the edge costs for traveling from the black node to its neighbors. The cost of traveling from black to green is 1, black to blue are $\sqrt{2}$ and from black to red are $\sqrt{3}$

III. Experimental setup

A. Overview

The Asus Xtion Pro Live RGB-D camera was mounted on the Gantry-Tau robot, as can be seen in Figure 10, and connected to a stationary computer running the Simulink model. A new labcomm module, making the communication between our stationary computer and the Gantry-Tau robot possible, was implemented and wrapped to work with our Simulink model, this is shown in Figure 9. When the model enters the *moving phase* it sends the global coordinates (x, y, z) relative to the initial position along with the four quaternion elements $q1, q2, q3, q4$ to the robot using the labcomm module. The quaternion elements will for simplicity never be changed letting the robot keep the same rotation matrix along the experiment.

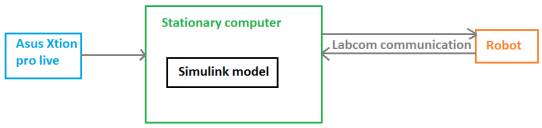

Fig. 9. Simulink to robot communication flow

B. Labcomm

Labcomm is a binary protocol developed at LTH. Labcomm only requires one way communication and keeps the communication to a minimum. It consists of a protocol specification and a compiler that generates C or Java code for needed methods such as encode/decode [*Labcomm* 2014].

The Labcomm module sets up a TCP IP socket to the robot computer. The stationary computer running the Simulink model is server and the robot is the client. After the server is started the client can connect to the server. The Labcomm module also initializes and registers a reader and a writer that will handle the stream input/output on the server side.

The method used by the server side in the moving phase can be seen in Listing 1.

Listing 1. Server side method.

```
float setTarget(float x, float y, float z,
    float q1, float q2, float q3, float
    q4)
```

It encodes and sends the robot target and then calls the decoder that locks execution until an acknowledgement is sent from the client application. The client code that controls the robot is written in ABB's robot programming language RAPID [Berlin, 2012].

C. Gantry Tau

The Gantry Tau structure was invented by Torgny Broghårdh and developed in the SMErobot [*SMErobot* 2009] and MONROE [*MONROE* 2012] projects. In 2000 a new family of parallel kinematic 3 DOF robots developed at ABB was presented by Broghårdh where the six carbon fiber links where clustered in a 3-2-1 configuration [Dressler, 2012]. The robot has three prismatic joints implemented as carts on linear guide ways connected to the end-effector plate [Dressler, 2012].

Fig. 10. L2 Gantry Tao Robot with mounted Xtion Asus Pro Live RGB-D camera.

In this experiment we tested the system's overall performance as well as the path planning implementation. The Gantry-Tau robot was set into a well suited starting position with a goal position in front of it. Then an obstacle was placed between the Gantry-Tau robot and the goal position. The system starts with an empty obstacle map. After the first iteration the obstacle map starts filling with what the camera sees. After some iterations when the Gantry-Tau robot approaches the obstacle it appears on the camera and is marked in the obstacle map. The obstacle will then be located on the planned path and the algorithm is forced to re-plan and travel around the obstacle.

IV. Results

A. Updating the Obstacle map

In Figure 11 we see the execution time of discretization using SDF, Box Approximation using every pixel and Box Approximation using every fifth row and every fifth column (every 25th pixel). Notice that the CPU time measured can only be used for relative time performance between the different discretization algorithms. The time measurements are done on the *freiburg1_teddy* sequence [*Freiburg1 Teddy sequence* 2011]. The statistics are presented in Table I.

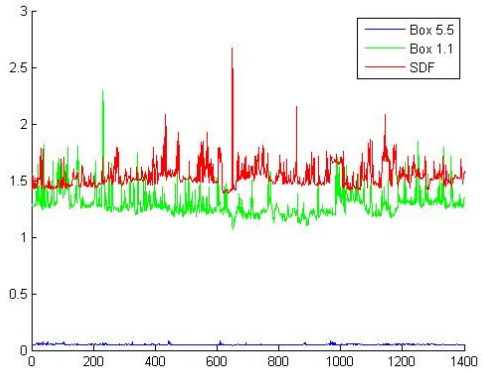

Fig. 11. Executions time for discretization using SDF, Box Approximation using every pixel and Box Approximation using every fifth pixel over 1400 samples.

CPU time	SDF / s	Box 1.1 / s	Box 5.5 /s
Mean	1.51	1.307	5.29×10^{-2}
Median	1.50	1.305	5.28×10^{-2}
Max	2.28	2.002	8.40×10^{-2}
Min	1.39	1.127	4.53×10^{-2}
Std	8.92×10^{-2}	6.78×10^{-2}	0.30×10^{-2}

TABLE I. Executions time for discretization using SDF, Box Approximation using every pixel and Box Approximation using every fifth pixel over 1000 samples.

B. Path planning

In this experiment a paper cylinder was placed about 30cm in front of and about 40cm under the robot (since the camera is pointed downwards). The start position of the robot is the upper blue dot marked in Figure 12. Then the robot was commanded to move to a position behind the cylinder, marked as the lower yellow dot in Figure 12. The shortest path from the start to the goal position would be straight through the cylinder but as the discrete map is created the path planning needs to re-plan to avoid the obstacle resulting in a path traveling around the cylinder. The environment can be seen in Figure 12 where the discrete map of each camera is painted with the same color as the camera dot. We clearly see how more and more of the environment is discovered as the robot moves.

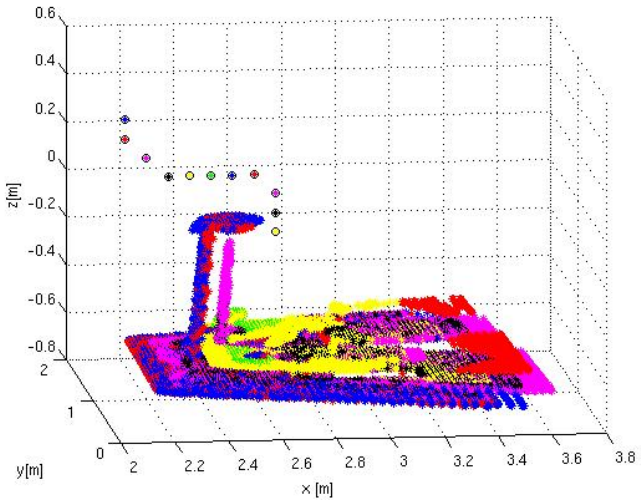

Fig. 12. This figure shows camera position (circles) and obstacle map while running path planning. The left blue camera is the start position and the lower right yellow camera is the goal position.

V. Conclusions

A. Updating the Obstacle map

Results show that Box Approximation using every pixel is faster than SDF and that Box Approximation using every fifth pixel is much faster than SDF. Performance wise SDF is better suited for image reconstruction with a higher need of precision but for graph generation Box Approximation is to prefer. SDF expects the area behind known objects to be untraversable until it is proven to be clear of obstacles and Box Approximation expects all areas to be traversable until an obstacle is spotted making it more compatible with the D* Lite algorithm which expects all paths to be free until proven otherwise. The main motivation for using Box Approximation in this application is that since SDF operates on the voxels a small object, for example a thin pole or hanging string, may lie in between two voxels and will not be represented in the discretization. Box Approximation on the other hand that operates on the pixels will instead fill the box even if just a small item is located there. This can of course lead the algorithm to believe that areas that could in fact be traveled are untraversable. Considering the nature of the application a rather diminished area avaliable for travel is better than a larger area with hidden and unrepresented obstacles.

B. Path planning

The D* Lite performs well. When a new obstacle occurs it re-plans the path and manages to avoid the obstacle.

Acknowledgment

First of all I want to thank my company supervisor Simon Yngve who has guided and supported me all through my thesis. I also want to thank my university supervisor Prof. Anders Robertsson from the Department of Automatic Control as well as my second university supervisor Erik Bylow from the Department of Mathematics for all help they have given me.

I also want to thank Maj Stenmark from the Department of Computer Science for her help with the Labcomm robot communication setup as well as her help with RobotStudio.

Finally I want to thank my family.

References

Berlin, H., Program Manager (2012). *Text based robot programming made easy*. Accessed 22 May 2015. RobotStudio. URL: http://www.abb.com/blog/gad00540/1DDE6.aspx?tag=RAPID%20programming.

Bylow, E. (2012). "Camera Tracking using a Dence 3D Model". Master's Thesis. Lund University, Faculty of Engineering, Center for Mathematical Sciences, Mathematics, Lund , Sweden.

Dressler, I. (2012). "Modeling and Control of Stiff Robots for Flexible Manufacturing". PhD thesis. Department of Automatic Control, Lund University, Sweden.

Freiburg1 Teddy sequence (2011). Last modified: 30 Sep 2011, 15:16. TUM. URL: http://vision.in.tum.de/data/datasets/rgbd-dataset/download#freiburg1_teddy.

Labcomm (2014). URL: http://wiki.cs.lth.se/moin/LabComm.

MONROE (2012). Accessed 7 May 2015. Hyper-Modular Open Networked RObot systems with Excellent performance. URL: http://www.echord.info/wikis/website/monroe.

P. E. Hart N. J. Nilsson, B. R. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics* **4**, pp. 100–107.

S. Koening, M. L. (2002). "Incremental A*". In: *Advances in Neural Information Processing Systems (NIPS)*. Georgia Institute of Technology, College of Computing, Atlanta, pp. 1539–1546.

– (2005). "Fast replanning for navigation in unknown terrain". *Transactions on Robotics* **21**:3, pp. 354–363.

SMErobot (2009). Accessed 7 May 2015. The European Robot Initiative for Strengthening the Competitiveness of SMEs in Manufacturing. URL: http://www.smerobot.org/.

Stentz, A. (1995). "The Focussed D* Algorithm for Real-Time Replanning". In: *In Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1652–1659.