CS302: Data Structures – Bonus Assignment 2018

Theme: Robot Navigation

Dr. Kostas Alexis
Autonomous Robots Lab
www.autonomousrobotslab.com

Document Version 1.0 (October 1 2018)

Table of Contents:

Section 1: Introduction to the Research Theme & this Assignment

Autonomous navigation of robotic systems is and will remain at the core of research activities. It is only when navigation of robots is absolutely autonomous, robust and resilient, that we can integrate them in complex and important tasks and applications. This bonus assignment provides an avenue on how to use your knowledge on data structures to solve (simplified) robot navigation problems.

More specifically, in this assignment you will deal with two major questions, namely:

1. How to represent the environment in a manner that facilitates efficient collision-free path planning for autonomous robots.
2. How to utilize this representation to conduct autonomous path planning.

As you execute this assignment you will simultaneously have the opportunity to get pointers to important topics in relation to robotics research. By all means, grasp the opportunity and move forward towards being a researcher/engineer/scientist in robotics.

Highlight: This assignment has a lot of theory in it! This is an opportunity to study a bit about robotics and get motivated for this field. You do not really need all this theory to execute the assignment. But we feel it is better to offer the background rather than just focusing on the simple questions of a course assignment only. The real world is quite complex.

Section 2: Where am I in the Environment?

One of the most fundamental questions for robotic systems is that of knowing where it is in its environment. Acquiring GPS signals is a simple way to answer this question. But what about when an environment is GPS-denied or GPS-degraded?

Modern robotics research has formally answered this question through a technique called Simultaneous Localization And Mapping (SLAM). SLAM solutions typically rely on the combination of certain exteroceptive and proprioceptive sensors such as cameras/LiDARs and Inertial Measurement Units (accelerometers, gyroscopes) respectively. In this assignment you will not deal with how to develop a SLAM algorithm as this would way beyond its scope. But we start by mentioning the initial problem and provide few good references in case you develop personal interest:

- Scaramuzza, D. and Fraundorfer, F., 2011. Visual odometry [tutorial]. IEEE robotics & automation magazine, 18(4), pp.80-92.
- Durrant-Whyte, H. and Bailey, T., 2006. Simultaneous localization and mapping: part I. IEEE robotics & automation magazine, 13(2), pp.99-110.
- Durrant-Whyte, H. and Bailey, T., 2006. Simultaneous localization and mapping: part I. IEEE robotics & automation magazine, 13(2), pp.99-110.

When a robot is able to simultaneously answer the question, what is my pose (where am I), and what is a map of the environment, then it also has to answer the question how to efficiently store this map and use it to plan its actions. A common way to represent maps are Point Clouds, a collection of <x,y,z> points representing "samples" from the environment surfaces that may (or may not be) annotated with color. In case you want to experiment with Point Clouds, there is a well-established library: http://pointclouds.org/

Figure 1 represents a visualization of a point cloud.

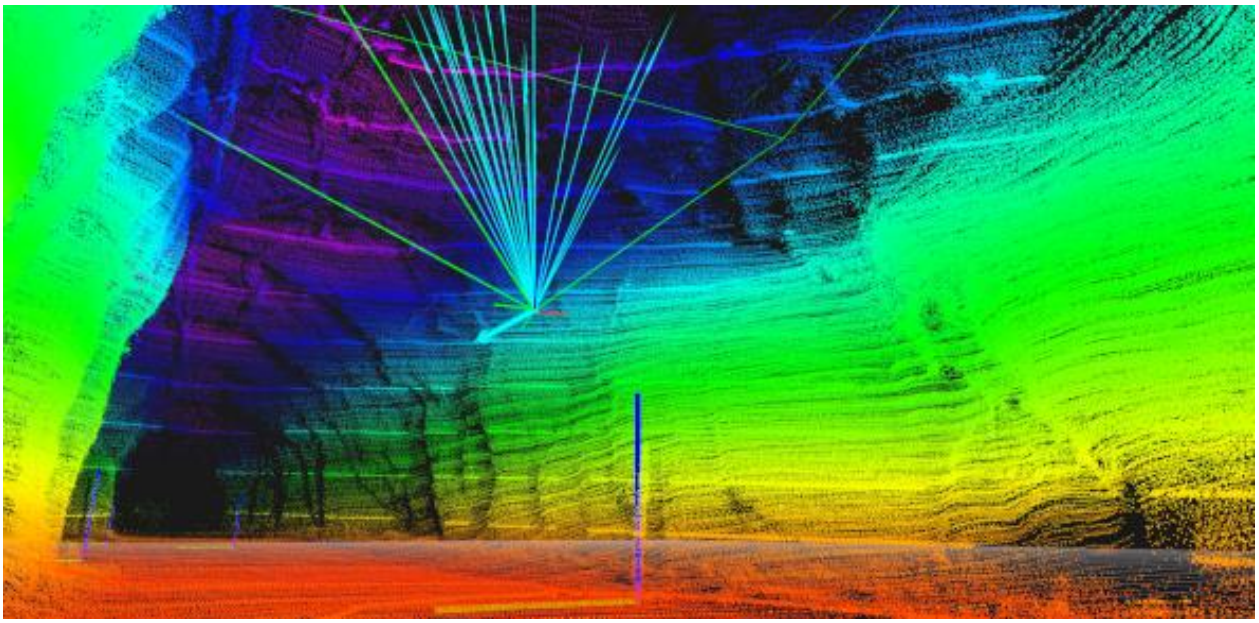


Figure 1. A point cloud of an underground setting autonomously reconstructed by a flying robot.

This representation may look visually appealing but it is in fact very computationally expensive to process and manipulate. This implies that although we do want robots to provide such high-quality results to their users, it may not be the best map representation to facilitate efficient path planning and decision making. In response to this fact, the robotics community has developed a set of alternative methods for robot-internal map representations that are optimized for efficiency. One of them is called "Occupancy Grid Maps".

Section 3: Occupancy Grid Mapping

Occupancy Grid Mapping refers to a set of probabilistic approaches to the problem of robot mapping that purely focuses on how sensor observations lead to inferring that certain parts of the environment are free and available for navigation or occupied. When we conduct occupancy grid mapping, we assume a SLAM solution – or any other way to directly get the robot pose – is available and runs onboard.

The basic idea of the occupancy grid is to represent a map of the environment as an evenly spaced field of binary random variables each representing the presence of an obstacle at that location in the environment. Occupancy grid algorithms compute approximate posterior estimates for these random variables.

3.1 Introduction to Occupancy Grid Mapping

In further detail, the occupancy mapping algorithm estimates the posterior probability over maps given the data $p(m|z_{1:t}, x_{1:t})$ where m is the map, $z_{1:t}$ is the set of measurements from time 1 to t, and $x_{1:t}$ is the set of robot poses from time 1 to t.

Occupancy grid algorithms represent the map m as a fine-grained grid over the continuous space of locations in the environment. The most common type of occupancy grid maps are 2D maps that describe a slide of the 3D world. Nowadays, we tend to use 3D grid maps and you can find two good references at the end of this section.

If we let m_i denote the grid cell with index i (often in 2D maps, two indices $< i, j >$ are used to represent the two dimensions), then the notation $p(m_i)$ represens the probability that cell i is occupied by an obstacle of the environment. The computational problem with estimating the posterior probability $p(m|z_{1:t}, x_{1:t})$ is the dimensionality of the problem: if a map contains N grid cells, then the number of possible maps that can be represented by tis gridding is 2^N. Now note that for average environments and reasonable resolutions we need millions of grid cells. Therefore, calculating a posterior probability for all such maps is not tractable.

The standard approach to resolve this problem in an efficient way, is to break down the problem into a set of smaller problems of estimating $p(m_i|z_{1:t}, x_{1:t})$ for all grid cells m_i. Each of these estimation problems is then a binary problem. This breakdown is convenient but does of course lose some of the structure of the problem, since it does not enable modeling dependencies among neighboring cells. Instead, the posterior of a map is approximated by factoring it into $p(m|z_{1:t}, x_{1:t}) = \prod_i p(m_i| z_{1:t}, x_{1:t})$. Due to this factorization, a binary Bayes Filter can be used to estimate the occupancy probability for each grid cell. For convergence properties, it is common to use a log-odds representation of the probability.

3.2 Pit Stop: A bit of introduction to the Bayes Filter

Bayes Filter is a probabilistic approach for estimating an unknown probability density function recursively over time using incoming measurements and a process model.

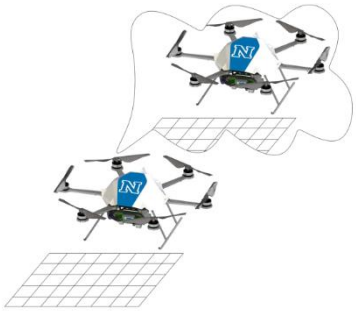


Figure 2. Robot belief of a true state.

Before we write down the basic equations of this algorithm, one needs to understand a bit deeper why do we truly need it. For that it is important to stress that in robotics we almost never have access to the true "world state" (the real state of the robot in the world). We typically end up trying to estimate the "belief state" (our best estimate of the world state) in a manner that is close and converging to the "true" robot state. For example, we try to make robots "estimate" their position in a manner that is truly close to their true location in the world.

In principle, consider that you have two ways to acquire information for your robot, namely:

- Through sensor observations – for example a camera, a gyroscope, barometer, GPS
- Through a model of your robot dynamics – its equations of motion

A sensor model can have a generic form $z = h(x)$ where x is the true robot state. Now one would think that we can just invert this equation and get x from z but unfortunately this in general does not hold (for reasons we will not explain in this document). Also note that sensor measurements are inaccurate, noisy and biased.

Similarly, a process model (or the model of the robot dynamics) can be represented by some sort of equation $x' = g(x, u)$ where u are the control actions to the robot (e.g., its motor commands). The problem here is that our model of the robot dynamics g is typically not accurate and trustworthy only within very short durations of time. So in general for robots we estimate probability over sensor observations $p(z|x)$ and a probability over motion models probability over motion models $p(x'|x, u)$.

All that may or may not sound relevant with CS302 but in fact, programming for advanced engineering requires a good relationship with math. So we will in fact continue but try to keep it straight and simple. So instead of a full textbook reference to the Bayes Rule (which is what we need to introduce to continue this discussion) let us directly outline a very simple example.

Example to understand the Bayes Rule: Assume you have a very simple quadrotor that its only purpose in life is to find its landing zone. The landing zone is marked with many bright lamps and this quadrotor has a light sensor onboard. Now consider the following:

- The robot is equipped with this light sensor and its output is binary, in other terms $Z \in \{bright, not\_bright\}$

- The world state (what we try to infer) is also binary, or $X \in \{home, not\_home\}$
- Our sensor is not truly good and therefore the probability of it reporting bright when the robot is actually on top of the landing zone is only 60% ($P(Z = bright|X = home) = 0.6$), while the probability of a false positive detection when the robot is not at home is 30% ($P(Z = bright|X = not_{home}) = 0.3$).
- We do not really know where the robot is initially so we assume that it may be either at home or elsewhere with equal probability, therefore $P(X = home) = 0.5$

Now you are tasked to find what is the probability, the robot's confidence, that it is actually at home after a single observation of the sensor reporting bright!

The mathematical rule to find an answer to this question is called the Bayes rule and as quickly as possibly is summarized in the following equation:

Observation likelihood — Prior on world state

$$P(x|z) = \frac{P(z|x)P(x)}{P(z)}$$

Prior on sensor observations

Let's apply it directly to our problem. We know in fact all terms other than the denominator. For that we use the trick of "marginalization" of probabilities. The result is:

$$P(X = home|Z = bright) = \frac{P(bright|home)P(home)}{P(bright|home)P(home) + P(bright|\bar{home})P(\bar{home})} = \frac{0.6 \cdot 0.5}{0.6 \cdot 0.5 + 0.3 \cdot 0.5} = 0.67$$

So the robot that didn't know where it is really, after a single reading of an inaccurate sensor already increased its confidence that it is above the landing zone by 17%! This is the first message to get for robotics: they are probabilistic systems. For now, we will dive into this further. This was motivation for you to study the underlying math topics.

3.3. Back to the Bayes Filter

A Bayes filter is an algorithm used for calculating the probabilities of multiple beliefs to allow a robot to infer information for its state and/or the map. Essentially, Bayes filters allow robots to continuously update their most likely position within a coordinate system, based on the most recently acquired sensor data. This is a recursive algorithm. It consists of two parts, namely "prediction" and "correction".

The true state x is assumed to be an unobserved Markov Process, and the measurements z are the observed states of a Hidden Markov Model (HMM). If we also have robot inputs u then the following diagram holds

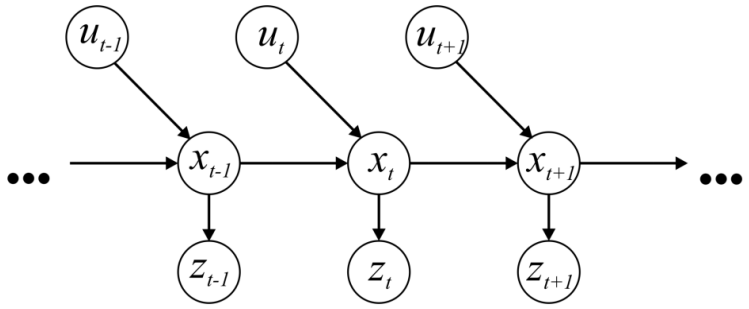


Figure 3. Visualization of a Markov Process

In the Markov Assumption the of sensor observations only depend on the current state

$$P(z_t|x_{0:t}, z_{1:t-1}, u_{1:t}) = P(z_t|x_t)$$

Similarly, the current state depends only on previous state and current action.

$$P(x_t|x_{0:t}, z_{1:t}, u_{1:t}) = P(x_t|x_{t-1}, u_t)$$

This fact gives rise to a recursive process according to which one can estimate the robot state by separating the so-called process update (by knowledge of the prior robot state belief and current control actions) and correction update (by knowledge of sensor readings). We will not dive any further. But a good reference for you is available at: https://www.autonomousrobotslab.com/badgerworks.html

These concepts of recursive estimation find application in fundamental algorithms of robotics such as the Kalman Filter and the Occupancy Grid Maps.

3.4. Occupancy Grid Mapping Algorithm Process

Let's jump into the specifics of the problem for this assignment. Consider you have a robot that is equipped with a sensor providing range readings as depicted below.

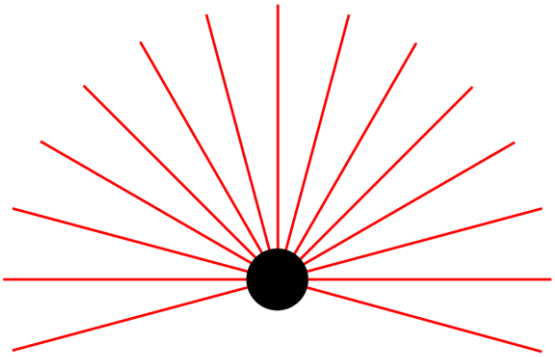


Figure 4. A visualization of a range sensor.

Each one of the rays of the sensor tells you if there is a surface ahead and at what distance. Now we want to estimate an occupancy map that is a grid of cells, each of them annotated with a probability of being occupied. Remember a 2D occupancy map looks like the following:

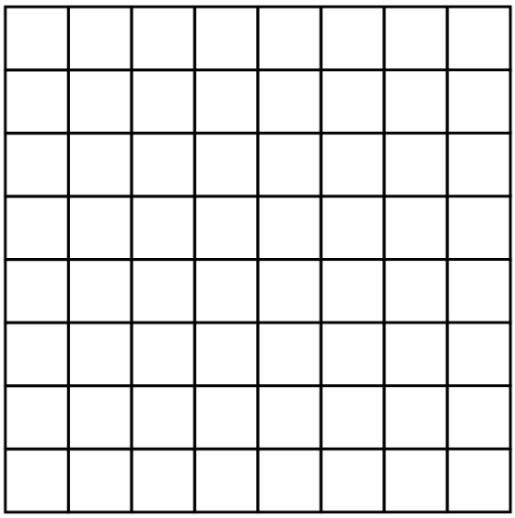


Figure 5. An occupancy grid map with no cell occupied.

With each $< i, j >$ corresponding to a location $< x, y >$ and being either free or occupied. In other terms $m_{x,y}$: $\{free, occupied\} \rightarrow \{0,1\}$. The challenge is how to probabilistically build the confidence that any of these cells is either free or occupied given noisy and imperfect sensor readings. Essentially we will build each of the probabilities $p(m_{x,y})$. We build this based on range measurements.

When we receive a range measurement, it provides us knowledge of which map cells that it went through are free, and which is the first occupied one it intersected with as visualized with blue and yellow in the following Figure. Therefore, a single range measurement is $z \sim \{0,1\}$ (zero meaning free and one meaning occupied).

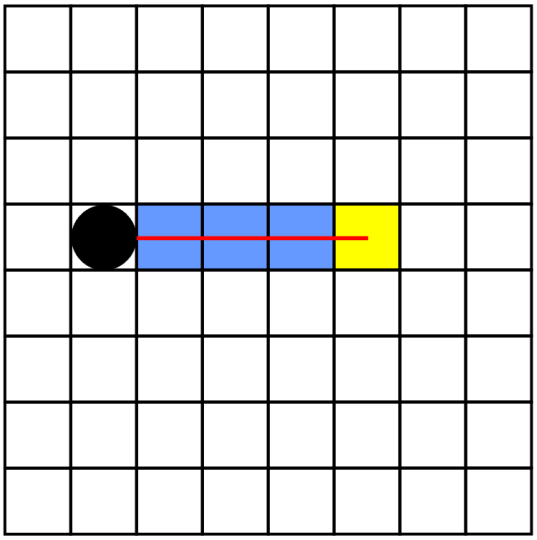


Figure 6. A range measurement on an occupancy map.

Now as you understand that we see same cells from different perspectives and therefore from new measurements, we can exploit that for accurate mapping. Let us write down the probabilities for each cell with respect to its sensor reading. In other terms let us write about the conditional probability $p(z|m_{x,y})$. More specifically:

- $p(z = 1|m_{x,y} = 1)$: True occupied measurement
- $p(z = 0|m_{x,y} = 1)$: False free measurement
- $p(z = 1|m_{x,y} = 0)$: False occupied measurement
- $p(z = 0|m_{x,y} = 0)$: True free measurement

As you understand the reason that we have false measurements is because we model sensor errors. Using a bit of rules applying to conditional probabilities let us re-write:

- $p(z = 1|m_{x,y} = 1)$
- $p(z = 0|m_{x,y} = 1) = 1 - p(z = 1|m_{x,y} = 1)$
- $p(z = 1|m_{x,y} = 0)$:
- $p(z = 0|m_{x,y} = 0) = 1 - p(z = 1|m_{x,y} = 0)$

Now we have the understanding we need. We understand that sensors provide measurements that are inaccurate, and we understand that we can have certain cases of probabilities for the occupancy map. We remind ourselves the Bayes Rule, and therefore we understand that our goal for every of the grid map cells is to find the following posterior probability:

$$p(m_{x,y}|z) = \frac{p(z|m_{x,y})p(m_{x,y})}{p(z)}$$

where:

- $p(m_{x,y}|z)$ is the posterior probability for cell $m_{x,y}$
- $p(z|m_{x,y})$ is the observation likelihood for this cell

- $p(m_{x,y})$ is the prior probability for this cell
- $p(z)$ is the sensor/evidence confidence value

To find this we introduce a trick called "odds". The "odds" of an event happening is

$$Odd := \frac{(X\ happens)}{(X\ not\ happens)} = \frac{p(X)}{p(X^c)}$$

More conveniently when we use "Odd"

$$Odd\left((m_{x,y}=1)given\ z\right) = \frac{p(m_{x,y}=1|z)}{p(m_{x,y}=0|z)}$$

Applying the Bayes Rule we can re-write the odd to include the sensor model term, and prior term. After some algebraic manipulation we have:

$$Odd: \quad \frac{p(m_{x,y}=1|z)}{p(m_{x,y}=0|z)} = \frac{p(z|m_{x,y}=1)p(m_{x,y}=1)}{p(z|m_{x,y}=0)p(m_{x,y}=0)}$$

And this equation becomes much more convenient if we apply a log function on it:

$$Log-Odd: \quad log\frac{p(m_{x,y}=1|z)}{p(m_{x,y}=0|z)} = log\frac{p(z|m_{x,y}=1)p(m_{x,y}=1)}{p(z|m_{x,y}=0)p(m_{x,y}=0)}$$

After some manipulation:

$$Log-Odd: log\frac{p(m_{x,y}=1|z)}{p(m_{x,y}=0|z)} = log\frac{p(z|m_{x,y}=1)p(m_{x,y}=1)}{p(z|m_{x,y}=0)p(m_{x,y}=0)} = log\frac{p(z|m_{x,y}=1)}{p(z|m_{x,y}=0)} + log\frac{p(m_{x,y}=1)}{p(m_{x,y}=0)}$$

To indicate the fact that we update our new belief based on prior and the confidence of new sensor readings for a certain event happening (e.g., a cell being occupied) we write:

$$\text{logodd}^{+} = \text{logodd\_meas} + \text{logodd}^{-}$$

Therefore, the map stores the log-odds of each cell and the measurement model is also modeled as a log-odd. There are two things you need to remember when you apply this rule:

1. The update step is only executed for cells for which you got measurements
2. The updated probabilities become priors for the next iteration of the algorithm in future time steps

Therefore, the update becomes recursive. The update rule is logodd += logodd_meas and as you understand it refers to a recursive formulation.

Let us see how the update step takes place in further detail. The measurement model in log-odd form is written as:

$$log\frac{p(z|m_{x,y}=1)}{p(z|m_{x,y}=0)}$$

We can receive two types of measurements (if for a certain cell, a measurement is received), namely:
1. Cells with $z = 1$
2. Cells with $z = 0$

For the "occupied" measurements we write:

$$logodd_{\downarrow}occ = log\frac{p(z=1|m_{x,y}=1)}{p(z=1|m_{x,y}=0)}$$

For the "free" measurements we write

$$logodd_{\uparrow}occ = log\frac{p(z=0|m_{x,y}=0)}{p(z=0|m_{x,y}=1)}$$

Alright, we are ready for an example. Let us consider the following Constant Measurement Model:
- $logodd_{\downarrow}occ = 0.9$
- $logodd_{\downarrow}free = 0.7$

An remember the update rule: $logodd+ = logodd_{\downarrow}meas$, while at the initial condition (t0) for all (x,y) cells of the map it holds that $logodd = 0$.

First of all, we have to understand that the abovementioned initialization is equivalent to setting $p(m_{x,y}=1) = p(m_{x,y}=0) = 0.5$ (same probability for each cell being occupied or free).

Now let us assume we receive a new measurement from our range sensor which for the moment let us also assume it is a single-ray one as shown below. Blue cells are measured as empty and the yellow as occupied.

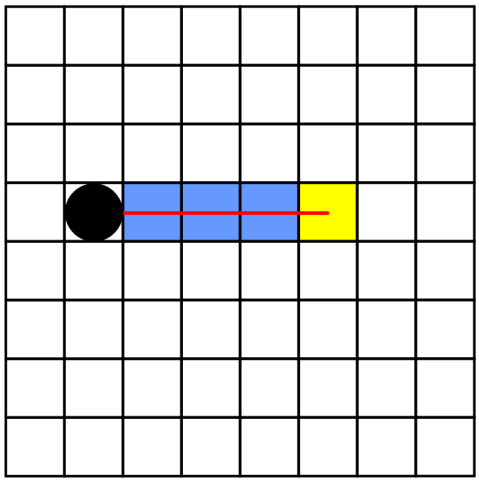


Figure 7. Our first measurement on an occupancy map

We can now proceed to execute the update step.

- For the cells for which we receive an "occupied" measurement we update the logodd respectively:

$$logodd \leftarrow 0 + logodd_{\downarrow}occ(= 0.9)$$

- Similarly, for the cells for which we receive a "free" measurement we update:

$$logodd \leftarrow 0 - logodd_{\downarrow}free(= 0.7)$$

Note that we write $0 + logodd$ ... and $0 - logodd$ ... as initially we had no sensor readings for any of the cells.

Now that we executed this first step let's see what the map looks like:

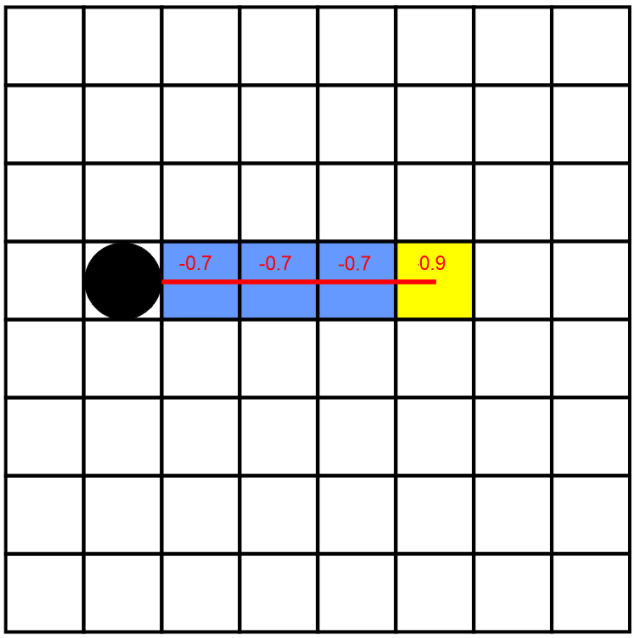


Figure 8. Updated probabilities for measured cells. All "white" cells are with zero initial log-odd value.

And then let's further assume that the robot rotated and got a new measurement:

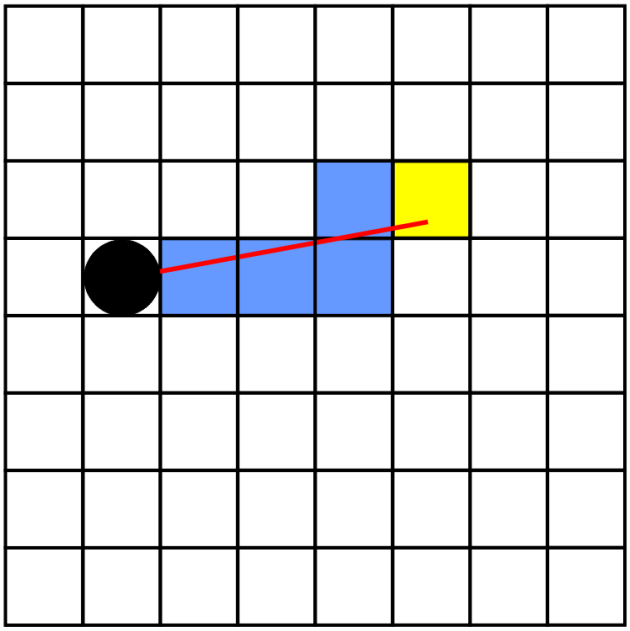


Figure 9. A second measurement with the sensor rotated compared to before.

Remember that we consider previous updates as new prior. This means we apply again the same rule as before. If you re-do the following steps but now starting from you will see that for the cells you have two measurements and they are free, their confidence of being free will become even more certain, while you will also assign a 0.9 probability on the new detected as occupied cell.

Section 4: Occupancy Grid Mapping Assignment

Your assignment is as follows. Extend the above example by considering a sensor that has multiple beams as shown below. Each sensor beam measurements up to 50 cells away and they are placed with a radial step size of 15 degrees (as depicted).

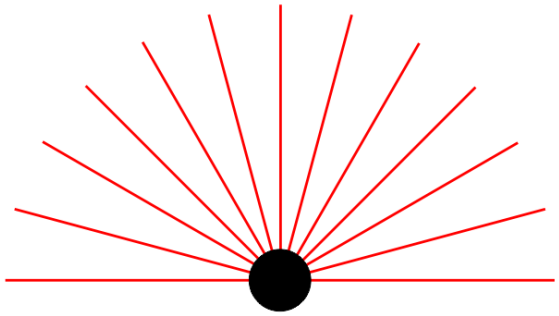


Figure 10. Visualization of the range sensor used. It provides 13 range measurements, each of them measuring up to 50 cells away.

Also consider that you are operating in a 1000x1000 grid map with each position on x and y being from 1 to 1000 with 1 stepsize increments. Zero-Zero location is upper left corner. Orientation zero for this robot means facing up north exactly as the above Figure. The sensor detects all free cells it goes through but only the first occupied cell it intersects with. In other terms, it does not see "behind" surfaces. In this environment, there is a square obstacle in all cells with x index equal from 250 to 500 and y index from 250 to 500. The robot has to detect the obstacle to navigate safely.

Now consider that you have a perfect estimate of your robot position and you know that your robot moved through the points presenting in the following Figure:

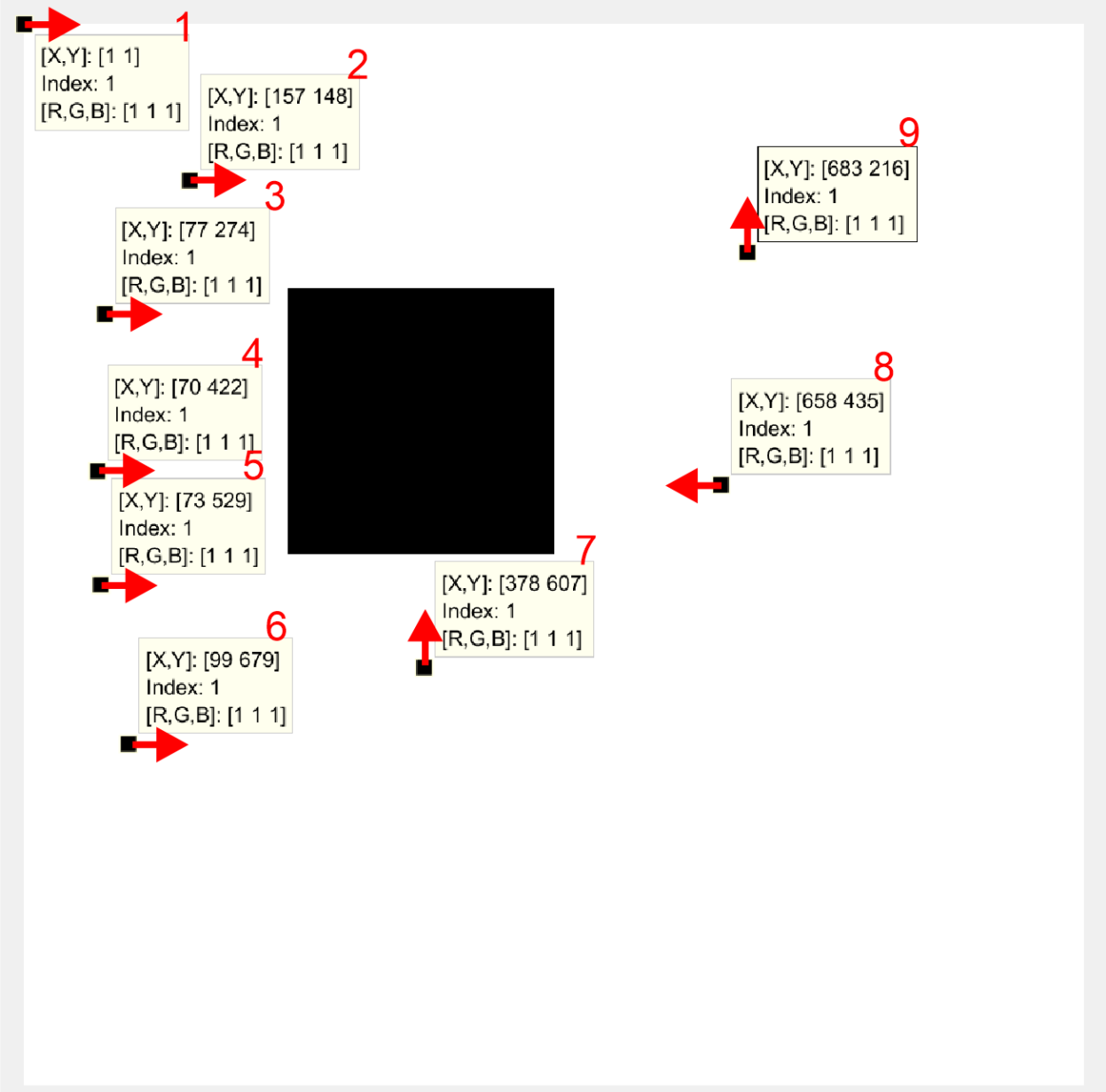


Figure 11. The robot locations and orientations fron which measurements were taken. You are expected to use a 1000x1000 grid map.

Also note that the orientation with which the robot was facing, which also implies a respective rotation of the sensor is also presented with the arrow.

Your assignment is about finding the resulting map after each of the aforementioned steps of the robot.

To execute this assignment, you will need to address the following challenges:

Task 1: Define a Data Structure to save the Occupancy Map. Argue about which you selected in terms of computational efficiency and memory needs.

Task 2: Do ray casting to find the cells through each sensor beam, at each location and orientation, goes through and therefore find "free" and "occupied" measurements. Was the selected data structure the ideal one for this operation?

Task 3: Introduce an additional 10^{th} measurement and further update the map. This 10^{th} measurement can be at any free area of the map and with any robot orientation.

Task 4: Derive the algorithm efficiency/complexity analysis for the following steps: a) Conducting ray casting to find which cells perceived by each of the sensor beams is free or occupied, and b) updating all the required probabilities.

Section 5: Path Planning for Collision-free Navigation

Assuming that the ability to estimate the robot pose and the map of its environment is facilitated, then another major challenge is that of planning paths for collision-free navigation. The literature consists of a series of approaches to the problem, including using exact models, following a purely reactive paradigm, hybrid approaches, probabilistic approaches utilizing sampling-based methods ore reinforcement learning schemes. The applicability of each of these methods relates to its own strengths and weaknesses, alongside the particular kinematic and dynamic configuration of the robot at hand. In principle, one can visualize the role of collision-free path planning through the following block diagram.

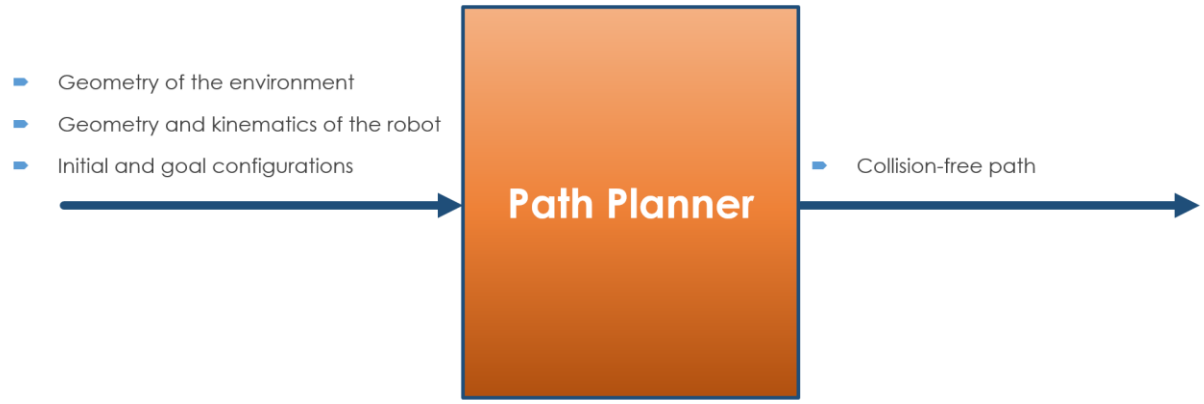


Figure 12. Block-diagram visualization of the path planner module receiving the current map of the environment, the pose of the robot and its possible constraints and deriving a collision-free path to be followed by the vehicle.

Generally, the goal of a collision-free path planning module is to compute a continuous sequence of collision-free robot configurations connecting the initial and goal configurations.

More formally: If W denotes the robot's workspace, and WO_i denotes the i-th obstacle, then the robot's free space, W_{free}, is defined as: $W_{free} = W - (\cup WO_i)$ and a path c is $c: [0,1] \rightarrow W_{free}$, where $c(0)$ is the starting configuration q_{start} and $c(1)$ is the goal configuration q_{goal}.

As you understand, knowledge of the free and occupied parts of the environment comes from the mapping step. Furthermore, if we assume that our system is modeled with a set of differential equations, then the collision-free motion planning problem is defined as:

Consider a dynamical control system defined by an ODE of the form:

$$\frac{dx}{dt} = f(x,u),\ x(0) = x_{init}$$

where is x the state, u is the control. Given an obstacle set X_{obs}, and a goal set X_{goal}, the objective of the motion planning problem is to find, if it exists, a control signal u such that the solution of (1) satisfies $x(t) \notin X_{obs}$ for all $t \in R^+$, and $x(t) \in X_{goal}$ for all $t > T$, for some finite $T \geq 0$. Return failure if no such control signal exists.

Section 6: Motion Planning using Rapidly-exploring Random Trees

Out of the multiple ways to conduct collision-free motion planning we will focus on sampling-based algorithms. In sampling-based algorithms, solutions are computed based on samples drawn from some distribution. Sampling-based algorithms retain some form of completeness, e.g., probabilistic completeness or resolution completeness.

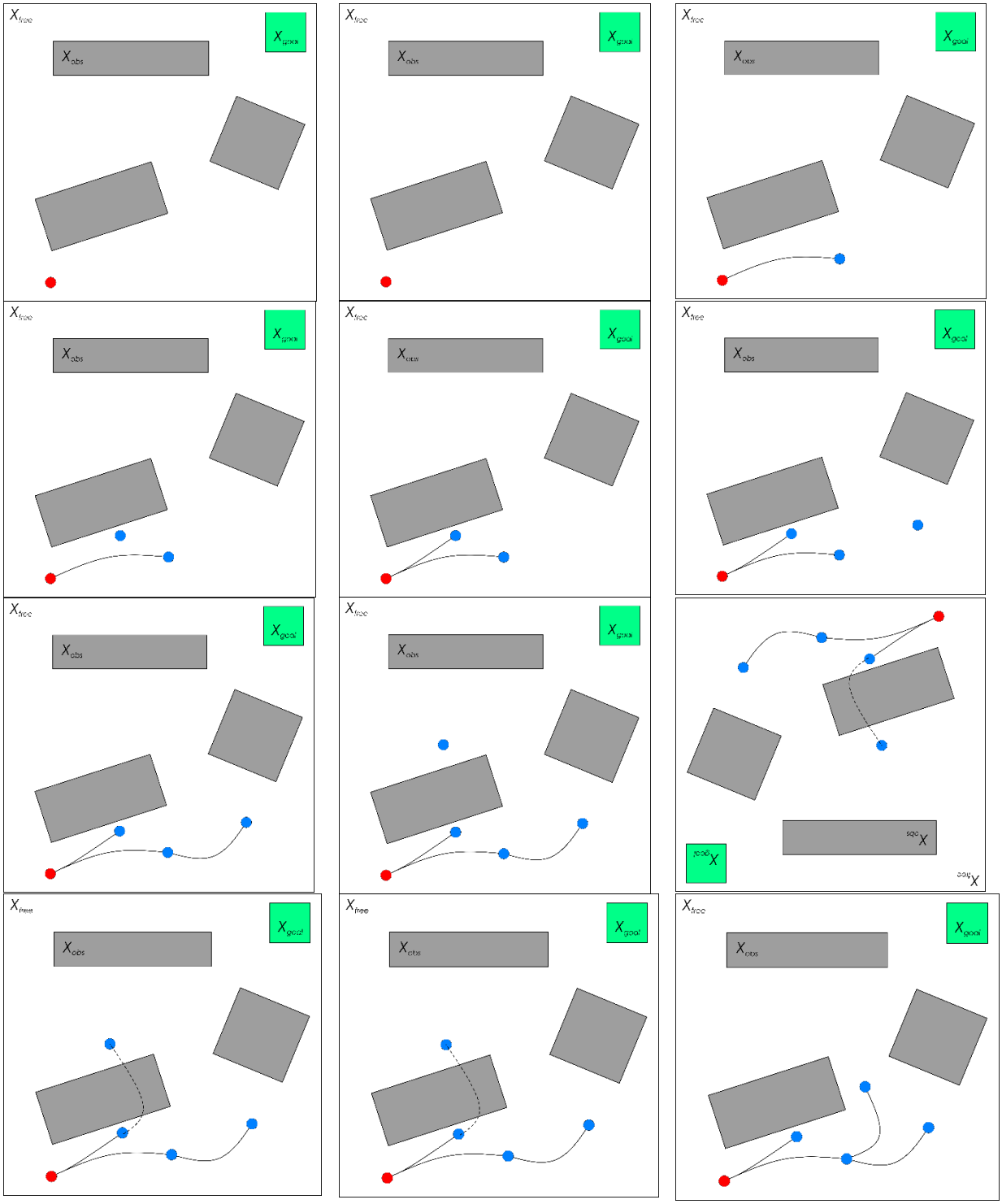
A particular sampling-based motion planner is that of using the so-called Rapidly-exploring Random Trees (RRTs). RRTs are appropriate for single-query planning problems. The idea is to build online a tree of robot configurations (e.g., robot locations and orientations) exploring the region of the state space that can be reached from the initial condition. At each step sample one point from X_{free} and try to connect it to the closest vertex in the tree. The methodology is very efficient, although it suffers from an effect called "Voronoi Bias" (search for it! And also search for the solution – the RRT*).

In pseudo-code, the RRT algorithm has as follows:

$\boldsymbol{V \leftarrow \{x_{init}\}; E \leftarrow 0;}$
for i=1,…,N do:
    $\boldsymbol{x_{rand} \leftarrow SampleFree;}$
    $\boldsymbol{x_{nearest} \leftarrow Nearest(G = (V, E), x_{rand});}$
    $\boldsymbol{x_{new} \leftarrow Steer(x_{nearest}, x_{rand});}$
    if $\boldsymbol{ObstacleFree(x_{nearest}, x_{new})}$ **then:**
        $\boldsymbol{V \leftarrow V \cup \{x_{new}\}; E \leftarrow E \cup \{(x_{nearest}, x_{new})\};}$
return $\boldsymbol{G = (V, E)}$;

where N is the number of iterations, "SampleFree" samples a random <x,y> and heading configuration within X_{free}, "Nearest" identifies the nearest previously sampled vertex (typically in a Euclidean Distance sense), "Steer" finds the connection path (which for the case of a quadrotor can be a straight edge between x_{rand} and $x_{nearest}$ and therefore in this case $x_{new} = x_{rand}$), the "ObstacleFree" function checks if the connection between x_{rand} and $x_{nearest}$ intersects with an obstacle (therefore you need the map!), while if this is not the case then we add the vertex to the set of vertices V and the new edge to set of edges E. This algorithm builds the tree and our only last goal is to identify when a certain admissible path arrives close (relevant to a fixed distance radius r defined by the user) to the goal configuration. If we find one, then we have a solution. We keep

running the algorithm if we have time and resources available in order to find alternative paths that may have smaller cost, i.e. they also arrive to the goal configuration but their length is shorter. A visualization of the execution of the RRT algorithm is shown in the images below.



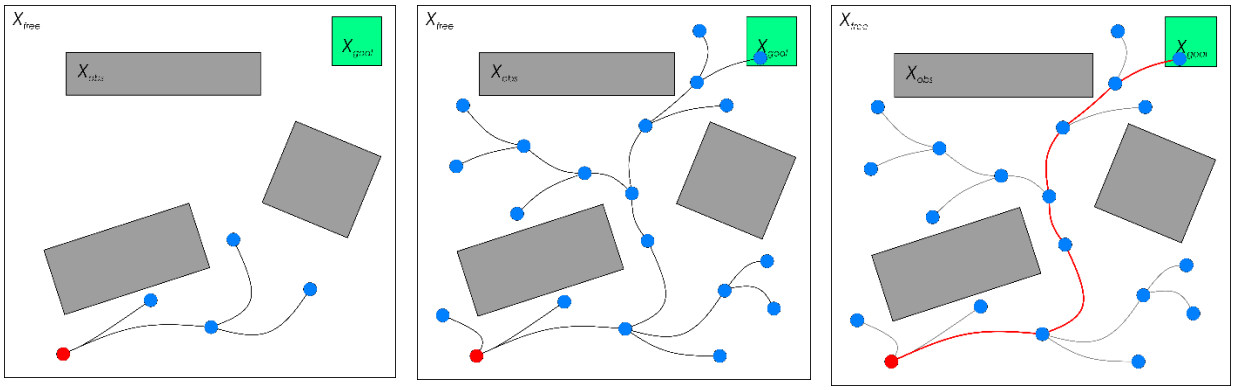


Figure 13. Steps of the execution of the RRT algorithm.

Two good references for the RRT algorithm are the following:

1. LaValle, S.M., 1998. Rapidly-exploring random trees: A new tool for path planning.
2. LaValle, S.M., 2006. Planning algorithms. Cambridge university press.

Section 7: Collision-free Path Planning Assignment

For your assignment you are called to implement RRT to derive collision free paths for a robot that has no kinematic constraints. Your robot operates on a 2D occupancy map identical to the one discussed in Section 4. You can sample only on <x,y> robot configurations and neglect heading orientation. You are requested to implement RRT such that you derive a collision-free path that connections the following initial and goal robot configurations:

- Initial Robot Configuration: <x,y> = <70,422>
- Final Robot Configuration: <x,y> = <658,435>

For your report include images of both the tree and the derived path in a manner analogous to the screenshot shown below.

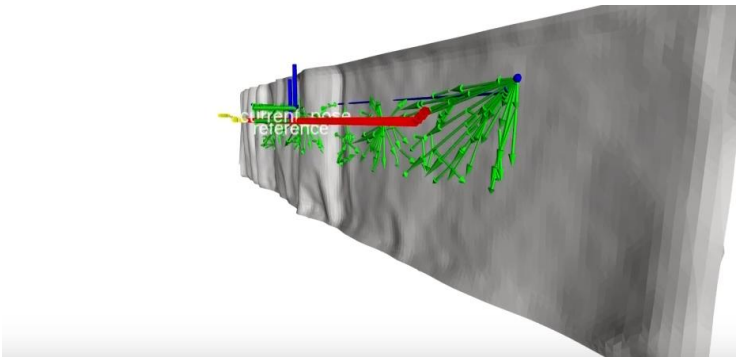


Figure 14. Sampling-based motion planning for in-contact navigation of aerial robots.

Section 8: What now?

If you liked this assignment, or if you found it too easy and want to explore more, please do one of the following:

- Contact us at kalexis@unr.edu or by talking to any other member of the Autonomous Robots Lab (www.autonomousrobotslab.com)
- Find out projects we propose for students here https://www.autonomousrobotslab.com/student-projects2.html or design one on your own and let us know for your ideas.
- Build a community of undergrads in robotics – join "BadgerWorks": https://www.autonomousrobotslab.com/badgerworks.html

Want motivation? Watch videos of our work at:

- https://www.autonomousrobotslab.com/autonomous-navigation-and-exploration.html
- https://www.autonomousrobotslab.com/fixed-wing-uavs.html
- https://www.autonomousrobotslab.com/agile-and-physical-interaction-control.html

And other links available at: https://www.autonomousrobotslab.com/

References

1. Thrun, S., Burgard, W. and Fox, D., 2005. Probabilistic robotics. MIT press.
2. Siegwart, R., Nourbakhsh, I.R., Scaramuzza, D. and Arkin, R.C., 2011. Introduction to autonomous mobile robots. MIT press.
3. Hornung, A., Wurm, K.M., Bennewitz, M., Stachniss, C. and Burgard, W., 2013. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. Autonomous Robots, 34(3), pp.189-206.
4. Oleynikova, H., Taylor, Z., Fehr, M., Siegwart, R. and Nieto, J., 2017, September. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on (pp. 1366-1373). IEEE.
5. Scaramuzza, D. and Fraundorfer, F., 2011. Visual odometry [tutorial]. IEEE robotics & automation magazine, 18(4), pp.80-92.
6. Durrant-Whyte, H. and Bailey, T., 2006. Simultaneous localization and mapping: part I. IEEE robotics & automation magazine, 13(2), pp.99-110.
7. Durrant-Whyte, H. and Bailey, T., 2006. Simultaneous localization and mapping: part I. IEEE robotics & automation magazine, 13(2), pp.99-110.
8. LaValle, S.M., 1998. Rapidly-exploring random trees: A new tool for path planning.
9. LaValle, S.M., 2006. Planning algorithms. Cambridge university press.
10. Bircher, A., Kamel, M., Alexis, K., Oleynikova, H. and Siegwart, R., 2016, May. Receding horizon" next-best-view" planner for 3d exploration. In Robotics and Automation (ICRA), 2016 IEEE International Conference on (pp. 1462-1468). IEEE.