National Polytechnic School

Control systems and Automation Department

logo-ENP.png

Seminar

Visual localization and obstacle avoidance
With application using a px4flow camera in a quadrotor

Written by :

SLIMOUNE Mohamed El Amine

BOUDJOGHRA Mohamed El Amine

DAIMELLAH Sofiane Sid Ali

Contents

Part I
Localisation

1 Feature based algorithms used for localization and mapping:

1.1 Camera localization using equipolar geometry:

To localize a camera we need to compute the fundamental matrix which defines the relationship between two views of a particular scene, given two matched pixels $X = \begin{pmatrix} x & y & 1 \end{pmatrix}$ and $X' = \begin{pmatrix} x' & y' & 1 \end{pmatrix}$, a constraint is defined as following

$$X'^T F X = 0$$

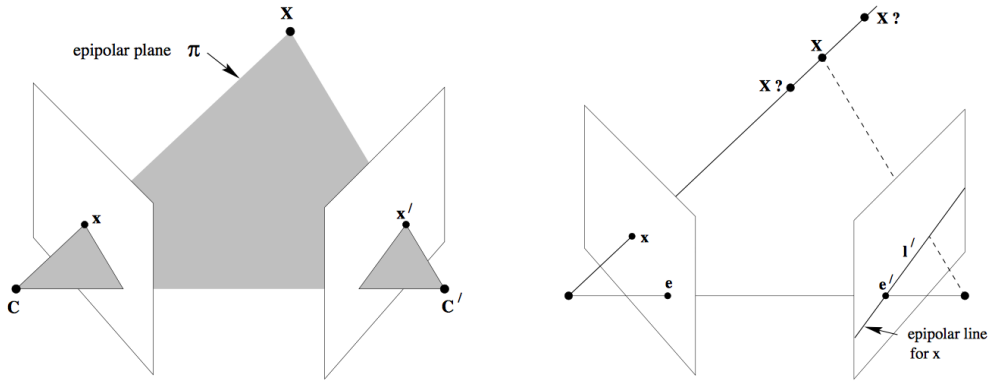


Figure 1: Illustration of two matched pixels

this constraint is used to construct a linear system $AF = 0$, where A is a matrix defined as, the number of the matched pixels has to be greater than 8 to define the matrix, thus this method is called the **eight-point method**

$$A = \begin{pmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ x'_2x_2 & x'_2y_2 & x'_2 & y'_2x_2 & y'_2y_2 & y'_2 & x_2 & y_2 & 1 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{pmatrix}$$

the fundemantal matrix F is defined as a 9 elements vector noted f

$$f = \begin{pmatrix} F_{11} & F_{12} & F_{13} & F_{21} & F_{22} & F_{23} & F_{31} & F_{32} & F_{33} \end{pmatrix}^T$$

If the number of matched pairs is greater than 8, we can get more robust result by minimizing $|Af|$

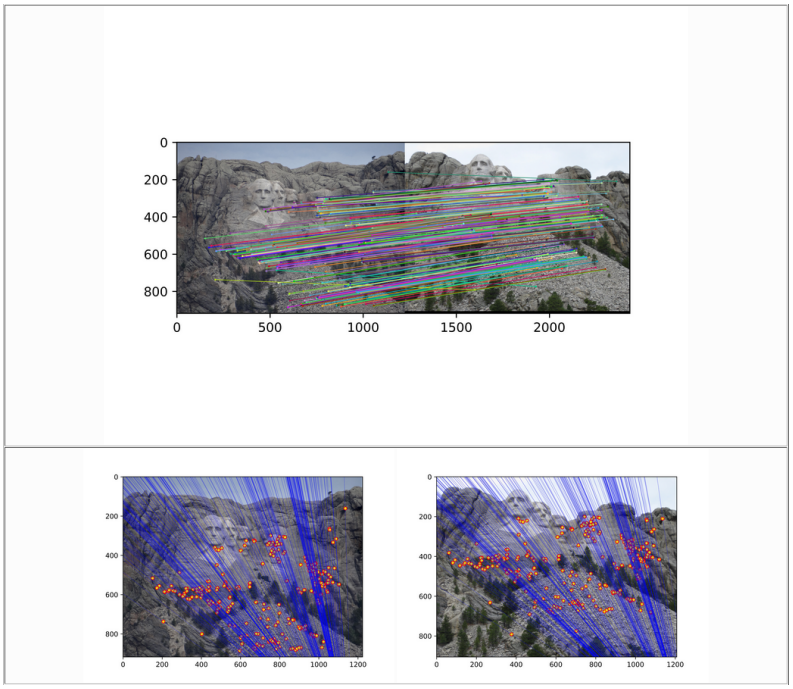


Figure 2: Matching two views and constructing the epipolar lines

the figure above illustrates different matched pixels in two views of the same scene depicted in the upper image which will be utilized next to estimate the fundamental matrix, the two images bellow show the epipolar lines that passe through the pixels to the other camera's optical center. All the epipolar lines create an epipolar plane.

1.1.1 Next step using RANSAC algorithm:

this algortihm is used to compute the fundamental matrix, firstly it uses all combinations of 8 pixels from the image frame to compute several fundamental matrices candidates using the **eight-point method**, **however there is only one to choose**.
in order to decide what matrix is the best, the number of inliers that verify the following constraint is computed

$$x'^{T} F x < threshold \ .....(1)$$

after that the fundamental matrix with the maximum number of inliers is chosen.
in the figure bellow the blue pixels will be tested whether they respect the constraint(1) or not, meanwhile the number of those respecting it will be counted.

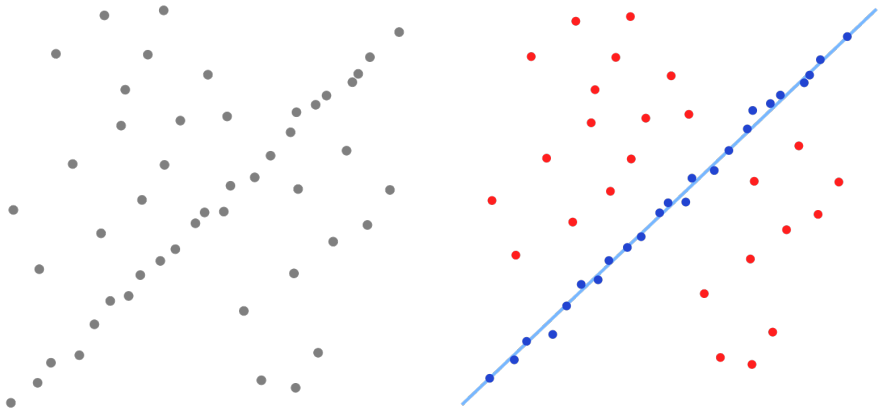


Figure 3: Fixing the inliers to be tested

1.1.2 Finding the essential matrix and the camera's location:

the next step to localize the camera is to compute the essential matrix, knowing that

$$E = K'^{T} F K$$

where K' in the intrinsic matrix of the first camera, and K is that of the second one.
after computing E we can deduce the rotation and the translation of the camera using the formula that relates them with the essential matrix

$$E = [t]_{\mathrm{x}} R$$

1.2 Depth estimation using dense tracking and mapping:

This method allows us to construct a depth model based on an RGB image, the used technique utilizes a cost function constructed by summing the photometric error of a pixel in different image frames. Thereafter, we seek the minimum of it to obtain the information about the depth of a given pixel in the image frame.

1.2.1 Theory of the method:

W refer to the projection matrix as T_{cw}, which describes the location of the camera with respect to the world coordinates.

$$T_{cw} = \begin{pmatrix} R_{cw} & c_w \\ 0 & 1 \end{pmatrix}$$

We refer to x_c as the position of the camera with respect to the camera's frame, and x_w the coordinates of a 3D point with respect to the world frame.

$$x_w = T_{cw} x_c$$

In order to get an estimate to the depth of a pixel the following cost function has to be minimized to find the corresponding depth d, the reason is that the intensity of a pixel is allays the same no

matter how much the image frame changes, called the brightness constancy assumption.

$$C_r(u,d) = \frac{1}{\mathcal{I}(r)} \sum_{m \in \mathcal{I}(r)} ||\rho(I_m, u, d)||$$

where $\rho(I_m, u, d)$ is photometric loss and describes the intensity difference of a pixel between the reference frame I_r and the frame at an instant t I_m, it is described with the following formula

$$\rho(I_m, u, d) = I_r(u) - I_m(\pi(KT_{mr}\pi^{-1}(u, d)))$$

the intensity I_m is of the pixel that matches the same point in the real world which means that it can be obtained by a transformation of the point $x = \pi^{-1}(u, d) = \frac{1}{d}K \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$, where K is the intrinsic matrix of the camera.

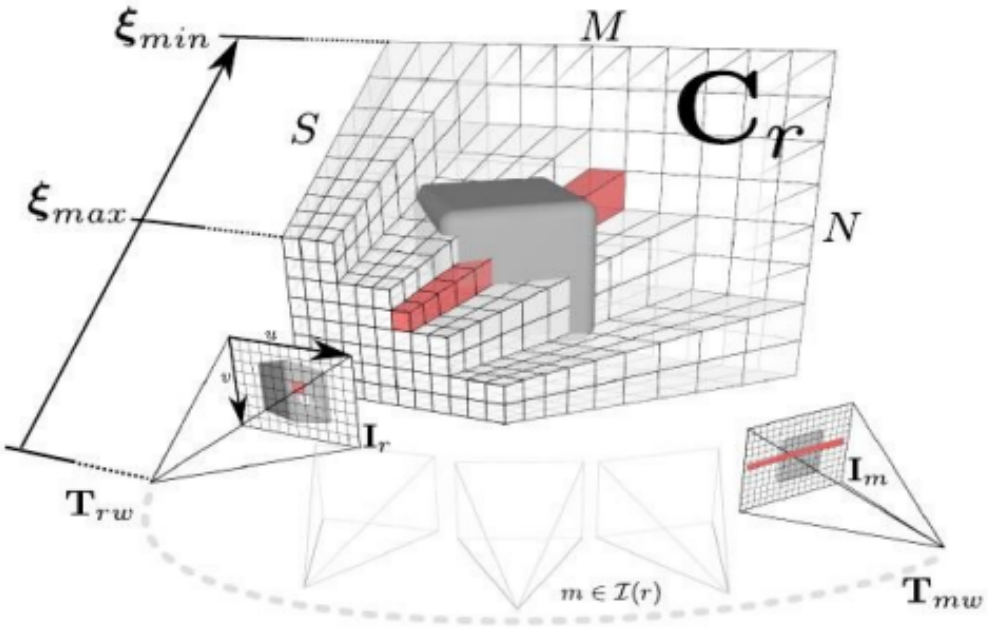


Figure 4: DTAM method

After plotting the variation of loss function $C_r(u, d)$ with the depth we can located the minimum depth of the three pixels presented extracted from the image bellow.

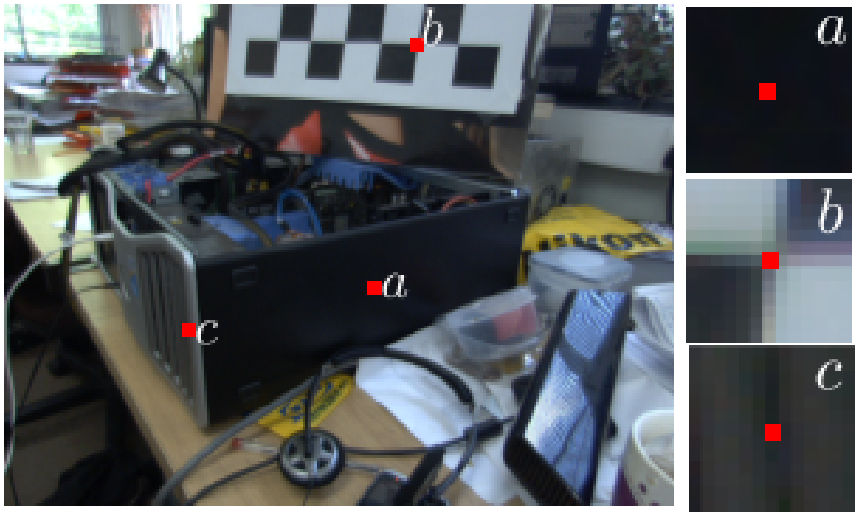


Figure 5: The used pixels

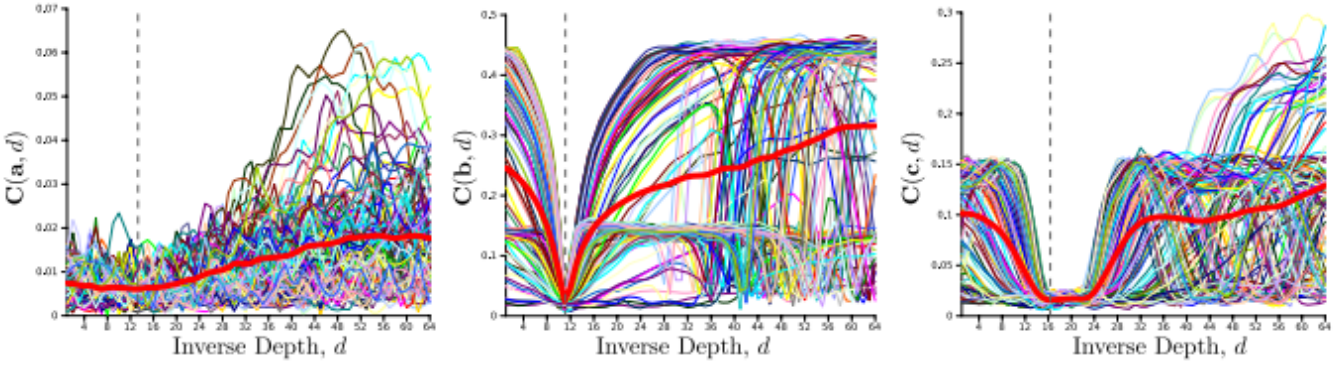


Figure 6: The found results

False minima are depicted by the photometric cost volume of pixel a in figure 3, which means that the information about the depth of the pixel remains unclear, if all pixels were subject to this problem then the final depth map will be similar to the image bellow.

Figure 7: false minima

To solve that problem a regularised cost is suggested.

1.2.2 regularised cost

The huber norm is described as following, it will be used next in the regulariser

$$||x||_\epsilon = \begin{cases} \frac{||x||_2^2}{2\epsilon} \text{ If } ||x||_2 < \epsilon \\ ||x||_1 - \frac{\epsilon}{2} \text{ Otherwise} \end{cases}$$

The regulariser used is consisted of a weighted huber norm of the inverse depth cost $\mathbf{g}(\mathbf{u})||\nabla\xi(\mathbf{u})||_\epsilon$.We also assume that the inverse depth solution consists of regions that vary smoothly with discontinuities due to occlusion boundaries.
moreover we desire to preserve edges and discontinuities.

Figure 8: Depth map filtering

The introduced regularised cost

$$E_\xi = \int_\Omega \left\{ g(u)||\nabla\xi(u)||_\epsilon + \lambda C(u, \xi(u)) \right\} du$$

- The first term ($g(u)$): regularization constraint, g is defined as 0 for image gradients and 1 for uniform regions, so the gradient on depth is penalized for uniform regions.

- Second term ($C(u, \xi(u))$): defined by the photometric error.
- Third term ($||\nabla\xi(u)||_\epsilon$): defferentiable replacement to L_1 that better preserve discontinuities than L_2

2 Application using KITTI dataset:

Part II
Path planning and obstacle avoidance

3 Introduction

One of the hottest subjects in autonomous navigation and mobile robotics is path planning. As a matter of fact, giving the ability to autonomously and safely navigate to our mobile robots provides us with an infinite range of applications in various fields such as entertainment, medicine, mining, rescuing, education, military, space, agriculture and many more.

In this part of the work, we will be focusing on different path planning algorithms, and do a comparison between them in terms of efficiency, and computation cost. But first, we need to go through some definitions.

3.1 Motion planning vs Path planning

The path **planning problem** is a sub-problem of the general **motion planning** problem. Path planning is the purely geometric problem of finding a collision-free path $q(s), s \in [0, 1]$, from a start configuration q_0 to a goal configuration q_f [1]. Thus, path planners focus on finding obstacle-free paths between two points, whereas motion planners attempt to construct a full trajectory with time and dynamics involved.

3.2 Online vs offline motion planning

Depending on the application and the requirements, one can be asked to implement either an **online planner**, when the environment is uncertain and obstacles are moving, or an **offline planner**, when the environment and the obstacles are static.

3.3 Optimal vs satisficing

In addition of reaching the goal state, we might want to look for the optimal way of doing so. By minimizing either the path length, or the effort required to follow that path. An example of a cost that can be minimized in an optimisation problem is :

$$J = \int_0^T L(x(t), u(t))\, dt$$

For example, minimizing with L = 1 yields a **time-optimal** motion while minimizing with $L = u^T(t)u(t)$ yields a **minimum-effort** motion.

3.4 Motion planner properties

There are plenty of path planners out there that can be distinguished through a determined set of properties, that are introduced bellow :

3.4.1 Multiple-query vs single-query planning

Multiple-query planners spend a considerable amount of time and energy trying to best represent the C_{free} space in order to solve multiple path planning problems. This kind of path planners is best suited to unchanging and well-known environments. On the other hand, Single-query planners solve each new problem from scratch. This kind of path planners are less computationally expensive, and are used in changing and unknown environments.

3.4.2 Completeness

A motion planner is said to be complete if it is guaranteed to find a solution in finite time if one exists, and to report failure if there is no feasible motion plan.

3.4.3 Resolution completeness

Resolution completeness is a weaker statement of motion planners' completeness. A planner is resolution complete if it is guaranteed to find a solution if one exists at the resolution of a discretized representation of the problem, such as the resolution of a grid representation of C_{free}.

3.4.4 Probabilistic completeness

Meaning that the probability of finding a solution, if one exists, tends to 1 as the planning time goes to infinity[1].

3.4.5 Computational complexity

The computational complexity refers to characterizations of the amount of time the planner takes to run or the amount of memory it requires. These are measured in terms of the description of the planning problem, such as the dimension of the C-space or the number of vertices in the representation of the robot and obstacles. For example, the time for a planner to run may be exponential in n, the dimension of the C-space. The computational complexity may be expressed in terms of the average case or the worst case [1].

4 Necessary foundation

4.1 Configuration space

The configuration space or the C_{space} for short, is defined such that every point in the C_{space} corresponds to a unique configuration q of the robot, and every configuration of the robot can be represented as a point in the C_{space} [1]. For example, the configuration of a robot arm with n joints can be represented as a list of n joint positions, q = $[\theta_1, ..., \theta_n]$. The free C_{space}, C_{free} consists of the configurations where the robot neither penetrates an obstacle nor violates a joint limit.

$$C_{space} = C_{free} \cup C_{obs}$$

The explicit mathematical representation of a $C_{obstacles}$ can be exceedingly complex, and for that reason $C_{obstacles}$ are rarely represented exactly. Despite this, the concept of $C_{obstacles}$ is very important for understanding motion planning algorithms. The ideas are best illustrated by examples.

Consider the 2R planar robot arm given bellow :

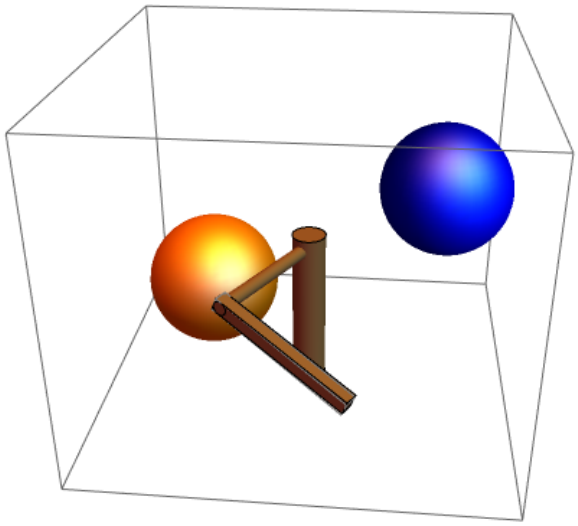


Figure 9: 2R robot arm in presence of obstacles

The exact configuration of this robot arm in its configuration space is represented as follows :

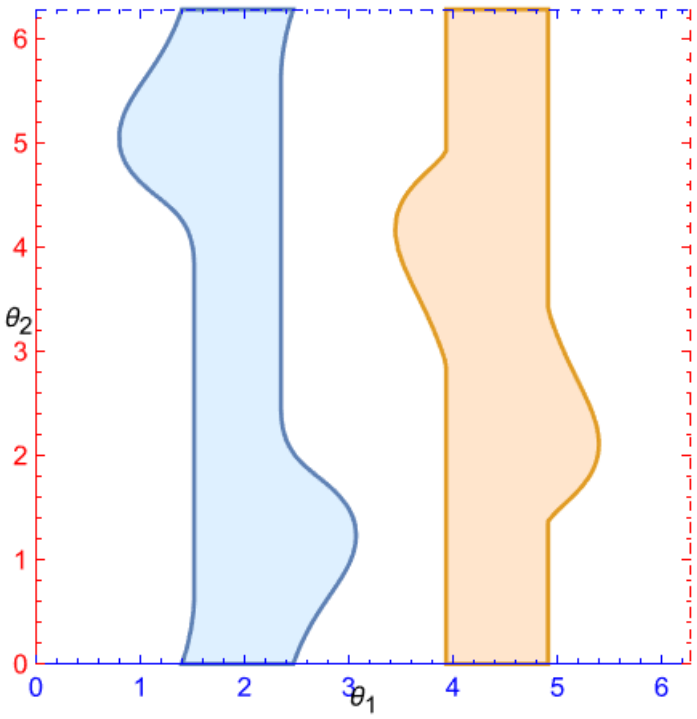


Figure 10: Configuration space of a 2R planar robot arm

Where the white domain represents the C_{free}, and the blue and orange domains represent the C_{obs} due to the blue and orange balls respectively. This configuration space representation is derived by slicing the original representation of the C_{space} (which is a torus, since the topology of

the C_{space} of a 2R planar robot arm is a torus) since the vertical edge at $\theta_1 = 0$ is connected to the vertical edge at $\theta_1 = 2\pi$, and the horizontal edge at $\theta_2 = 0$ is connected to the horizontal edge at $\theta_2 = 2\pi$. Thus, if we slice the torus twice and lay it flat on the plane, we obtain the above representation.

We can notice that if we have a starting state that lays between the two obstacles in the configuration space representation above, there is no way for it to reach a goal state that is outside of that region. We say that the two points are located in two different **connected components**. We define a connected component as the set of reachable points in the configuration space from a starting configuration, without penetrating any obstacle, and taking into consideration joints and actuators limits.

In this example the configuration space is subdivided into two connected components as illustrated bellow :

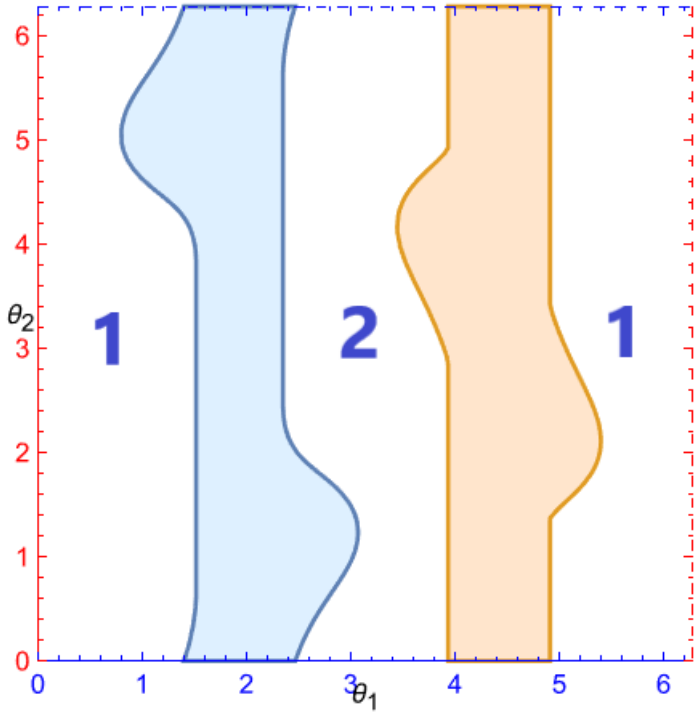


Figure 11: Configuration space of a 2R planar robot arm

Important note : The importance of such representation will be demonstrated when different path planners will be introduced.

4.2 Distance to Obstacles and Collision Detection

Given a configuration q, let $d(q, C_{obs})$ be the shortest distance between the robot and the obstacle. We can then derive the following rules :

$$d(q, C_{obs}) > 0 \longrightarrow \text{No contact with the obstacle.}$$
$$d(q, C_{obs}) = 0 \longrightarrow \text{Contact.}$$
$$d(q, C_{obs}) < 0 \longrightarrow \text{Penetration.}$$

A distance-measurement algorithm is one that determines $d(q, C_{obs})$. A collision–detection routine determines whether $d(q, Bi) \leq 0$ for any obstacle in the C_{space} and return a binary result.

One simple approach of collision detection is to approximate the robot and the obstacles with the union of overlapping spheres. The approximation has to be **conservative**, meaning that the bodies need to be completely covered with the spheres, so that if the collision detection routine indicates a collision free path, it is guaranteed that the latter is free of obstacles. To best represent the objects in the C_{space} we need more spheres, however this results in more distance checks and can become computationally expensive.

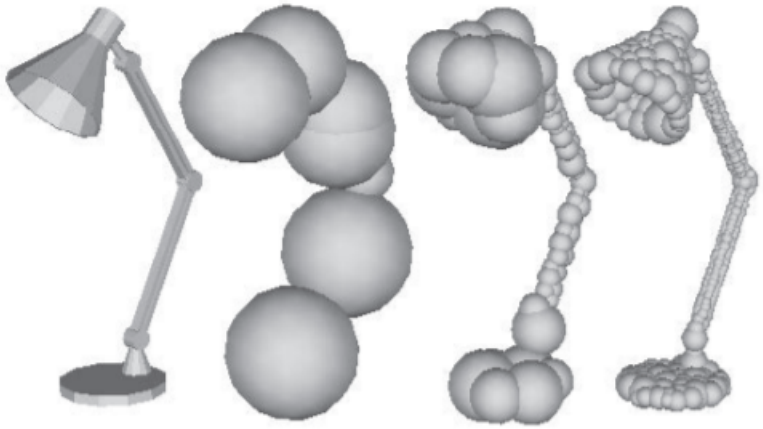


Figure 12: Sphere representation of a lamp [2]

4.3 Graphs and Trees

Many motion planners rely on the representation of the C_{space} as a **graph** or a **tree**. A graph is defined by a set of nodes N and a set of edges E where the nodes represent a specific configuration in the C_{space} and the edges are the free paths connecting the nodes. A graph can be **directed** or **undirected**. A graph is said to be undirected when each edge is bidirectional, meaning that for each node n_i and n_j, both paths a possible (from n_i to n_j or from n_j to n_i) through the edge e_{ij}, and is directed when only way is possible through the edges.

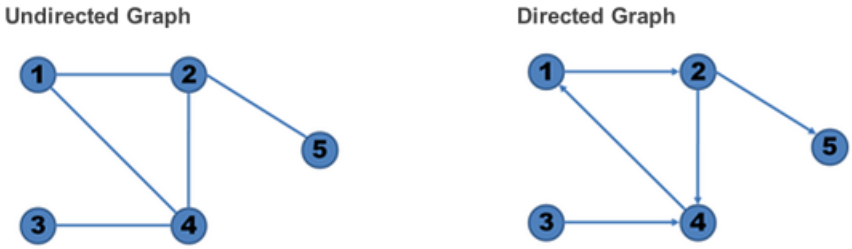


Figure 13: Directed vs Undirected graphs

Graphs can also be **weighted** or **unweighted**, they are weighted when the cost moving from

a node to another a different and determined by the edges, whereas unweighted graphs are characterized by the fact that every edge has a weight of 1.

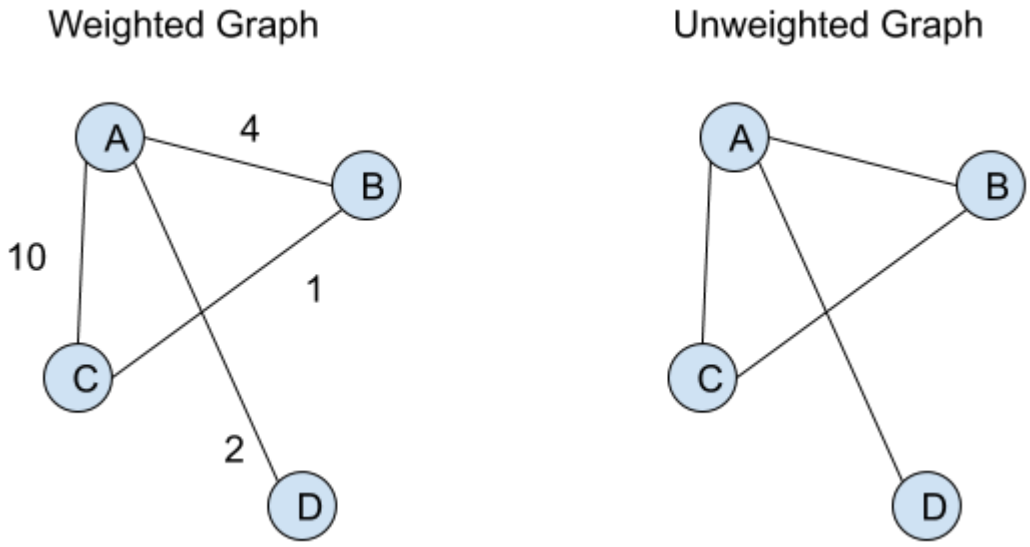


Figure 14: Weighted vs Unweighted graphs

A tree is a simply directed graph in which there are no cycles and each node has at most one parent node. It also has one **root** node with no parents and a number of **leaf** nodes with no child. And finally, a tree can also be weighted or unweighted.

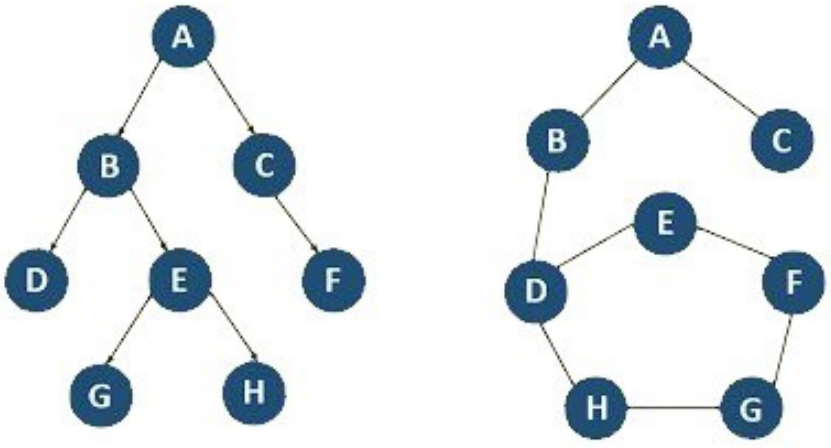


Figure 15: Graphs vs Trees

5 Path planning algorithms

For several decades, various researchers and scientists have provided numerous methodologies on navigational approaches. Thus, there is no single planner applicable to all motion planning problems. In this work, we will be discussing some of the many planners available in the literature, and make comparisons when appropriate.

6 Grid-based methods

Grid-based path planners proceed by **discretizing** the C_{space} into a grid and **connect** the grid cells in order to obtain a **graph** that will be searched using a **graph-search** algorithm (which we will introduce later in this section) in order to find a free path from a starting point q_{start} to the goal q_{goal}. These planners are easy to implement and can return optimal paths at the level of discretisation. Hence, their solutions are **resolution complete**.

The idea is to derive a graph from the discretized grid representing the C_{space} and then apply the above-mentioned algorithms. To do so, we need to determine whether the robot constrained to move in axis-aligned directions in configuration space or can it move in multiple dimensions simultaneously. If we consider a 2D C_{space} for demonstration purposes, the problem is reduced to choosing whether neighbour cells are 4-connected (south, north, east, west), or 8-connected, meaning that diagonals are also allowed.

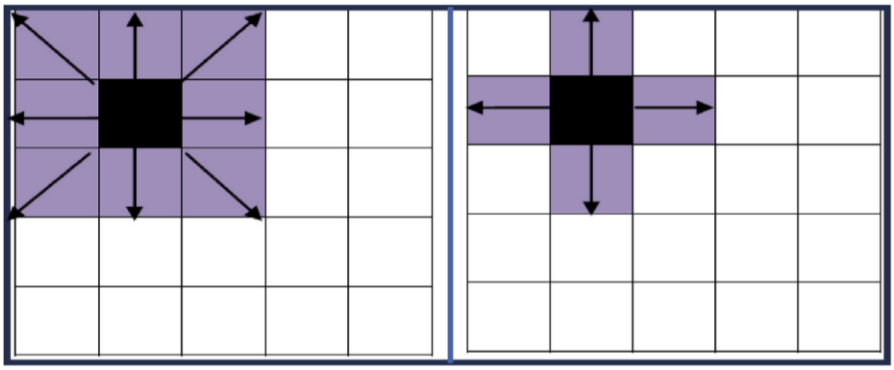


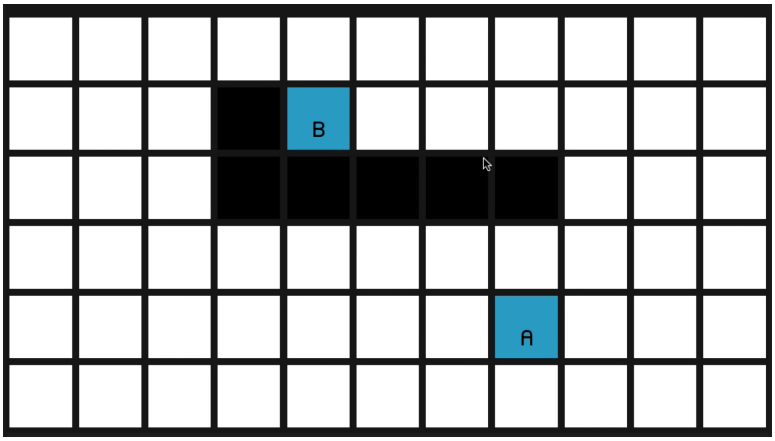
Figure 16: 4-connected cells vs 8-connected cells [3]

After defining the cells' connection, we should end up with a graph accurately describing the free space in which the robot will navigate. We then apply one of the many graph-search algorithms.

6.1 Graph-search algorithms

6.1.1 A* algorithm

A* algorithm is one of the most popular and most efficient graph-search algorithms out there. It can efficiently find a minimum-cost path on a graph when the cost of the path is simply the sum of the positive edge costs along the path.
To understand the algorithm, let's consider an example :

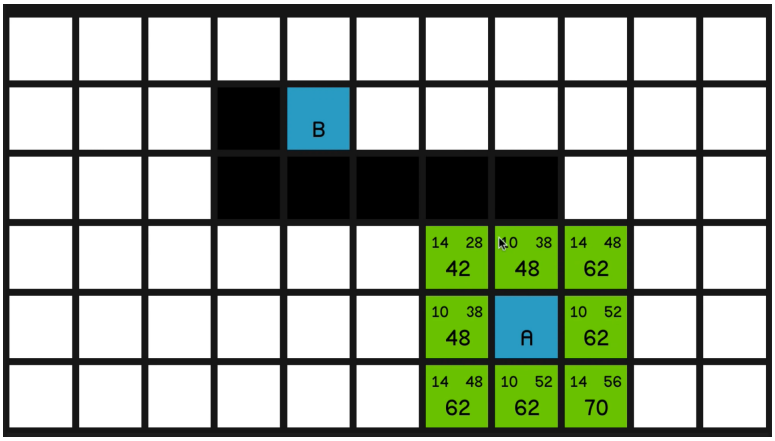


Where the white cells represent the C_{free} the black ones represent the C_{obs}, whereas the cells A and B represent respectively the initial state configuration and the goal configuration.

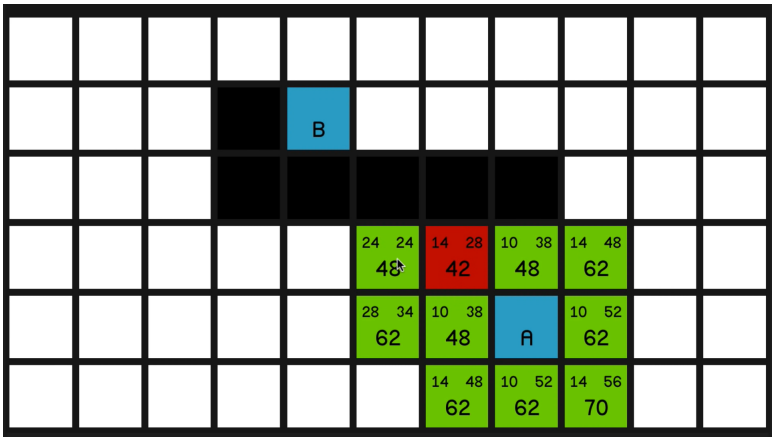
The A* algorithm uses three important parameters to guide the search :

- **Gcost:** The distance between the current node and the starting node.
- **Hcost:** An optimistic estimation of the distance between the current node and the goal node that needs to be smaller than the actual cost to go (generally, we use the euclidean distance to represent the Hcost).
- **Fcost:** The sum of the Gcost and the Hcost.

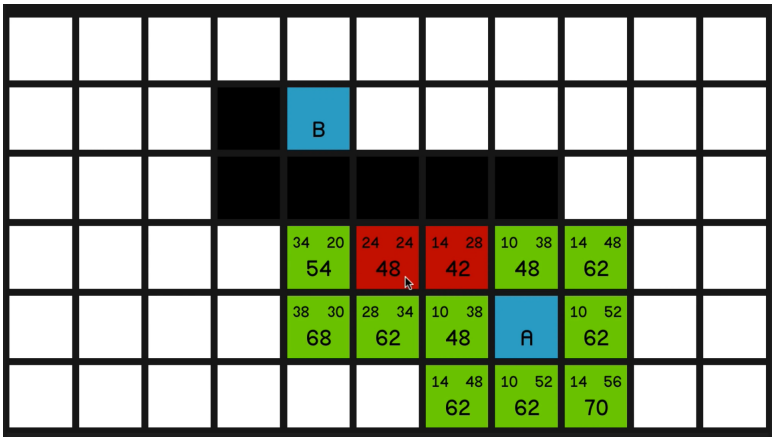
To start the algorithm, we start exploring the neighbours of the initial configuration by putting them in a list called OPEN (the open cells are represented in green).



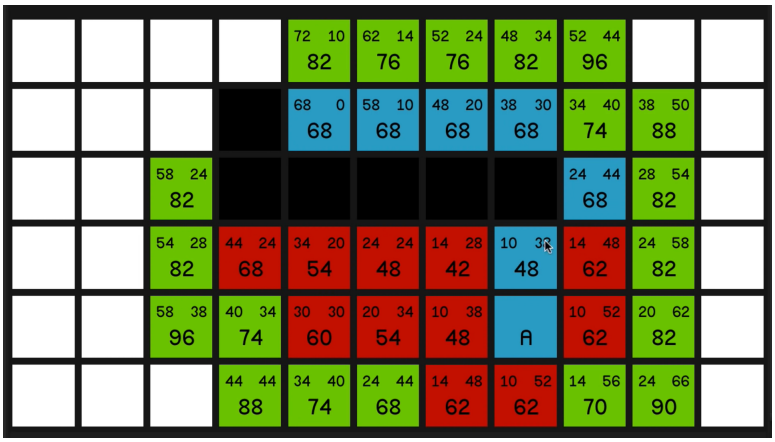
In the top left corner of each cell is the Gcost where the horizontal and vertical neighbours have a value of 10 and the diagonal ones a value of 14 (since 14 is an approximation of $10 \times \sqrt{2}$). We find the Hcost in the to right corner of each cell and the Fcost in the center.
Then, the algorithm is always going to look for the open node with the lowest Fcost to explore first. In our case, it is the node with an Fcost of 42 on the top left corner:



It will put the node in the CLOSED list (in red), and explore again. In case of equal Fcosts, the algorithm looks for the lowest Hcost:



And the process is repeated until the algorithm finds the optimal path connecting the starting point to the goal point (the optimal path is showed in blue).



An implementation of this algorithm for solving different path planning problems in different environments has been done, and gave the following results :

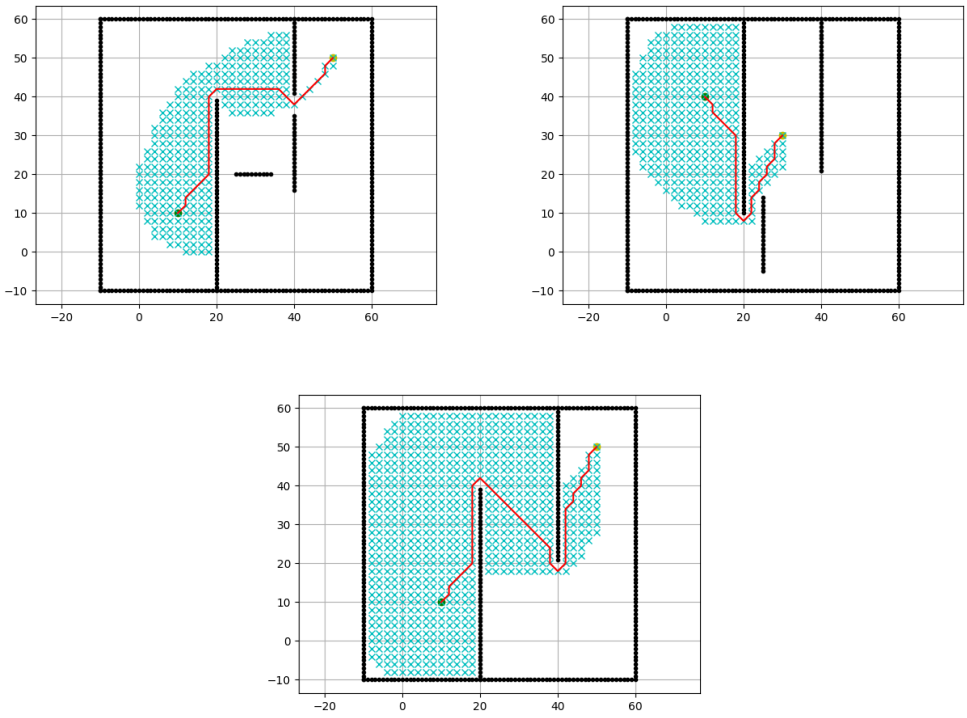


Figure 17: Simulation results for A* search

6.1.2 Other Search Algorithms

- **Dijkstra's algorithm:** If the heuristic cost-to-go (Hcost) is always estimated as zero then A* always explores from the OPEN node that has been reached with minimum past cost. This variant is called Dijkstra's algorithm, which preceded A* historically. Dijkstra's algorithm is also guaranteed to find a minimum-cost path but on many problems it runs more slowly than A* owing to the lack of a heuristic look-ahead function to help guide the search. Here's a simulation to compare the performance of the two algorithms :

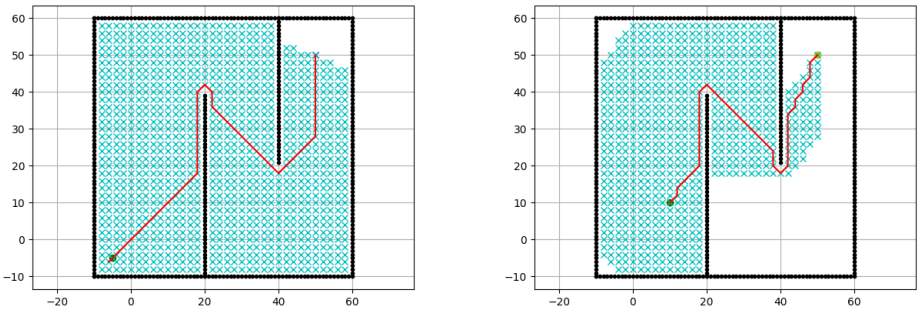


Figure 18: Djikstra vs A*

Notice how Djikstra's algorithm explores more cells when looking for the optimal path.

- **Breadth-first search:** If each edge in E has the same cost, Dijkstra's algorithm reduces to breadth-first search. All nodes one edge away from the start node are considered first, then all nodes two edges away, etc. The first solution found is therefore a minimum-cost path.

6.2 Conclusion

As a conclusion, we can derive the following characteristics of grid-based planners :

- Their solutions are not complete, but resolution complete.
- Their solutions are optimal at the level of discretization using A* algorithm.
- Not practical for high dimensional configuration spaces as the number of cells grows exponentially with the dimension of the configuration space (k^n with n the dimension of the C_{space} and k the resolution).

7 Sampling-based methods

Unlike grid-based methods, sampling-based methods are designed to find free paths even in high dimensional state spaces. A generic sampling-based method relies on a function to choose a sample from the C-space, a function to detect whether the sample is in C_{free}, a function to determine the closest previous free sample and a local planner to try to connect the two or move towards the latest sample. Sampling-based planners are **probabilistic complete**, their solutions are satisticing and not optimal, and they can be either multiple-query planners or single-query planners.

7.1 Rapidly-exploring random trees (RRT)

Rapidly-exploring random trees are single-query planners, meaning that they always compute every new query from scratch. They are based on the rapid exploration of the C_{space} in order to construct a tree that covers as much space in the environment as possible, with as few nodes as possible.

To find a free path, the algorithm goes through the following steps :

- Choose the first random sample node.
- Check whether the sample node is in C_{free}
- Connect the starting point to the first sample (or to a closer node in the direction of the sample node depending on the allowable distance to travel) using a local planner.
- Choose another sampling point and check that it is in C_{free}
- Connect the sample node to the closest node in the tree.
- Repeat from step 4 until a goal region is found.

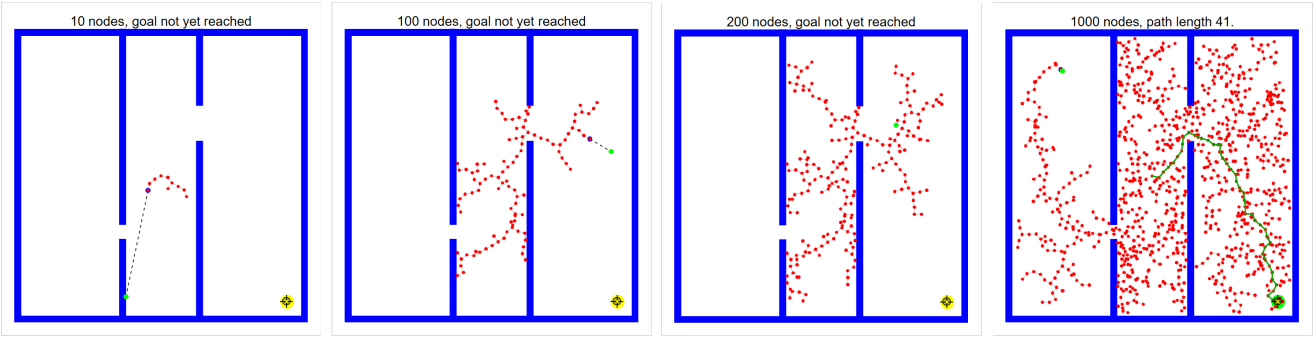


Figure 19: Rapidly-exploring random trees in action

We can see how the tree is rapidly growing to explore more and more of the environment until it finds a free path from the starting point to a goal region. However, one of the most considerable drawbacks of the RRT algorithm, is that it never attempts to find a better path once it finds one for the first time. To fix this, researchers introduced RRT* algorithm, which works in much the same way as RRT, but differs in where to connect new sample nodes to the tree. Instead of connecting the new node to the closest node in the tree, it checks for other nodes within some specified **search radius**, and determine if we can reconnect these local nodes in a way that maintains the tree structure, but also minimized the total path length. This addition to the algorithm equips it with a very powerful property, that is the asymptotic optimality of its solution. Meaning that the planner will find the optimal solution as the number of nodes in the tree tends to infinity.

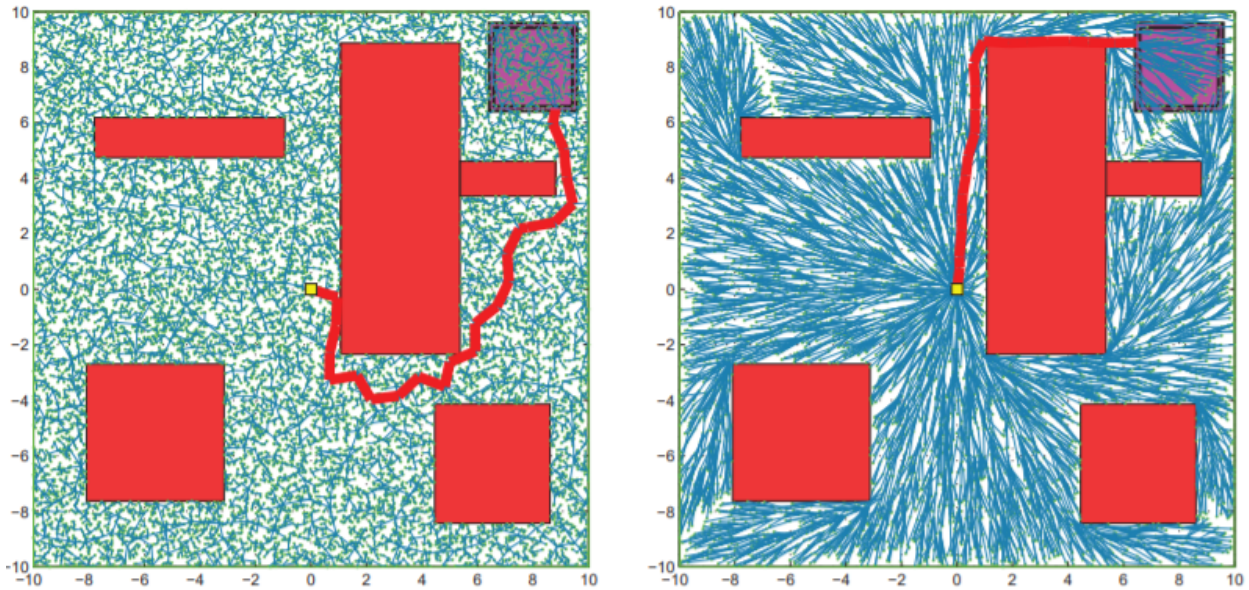


Figure 20: RRT vs RRT* [5]

7.2 Probabilistic roadmaps (PRM)

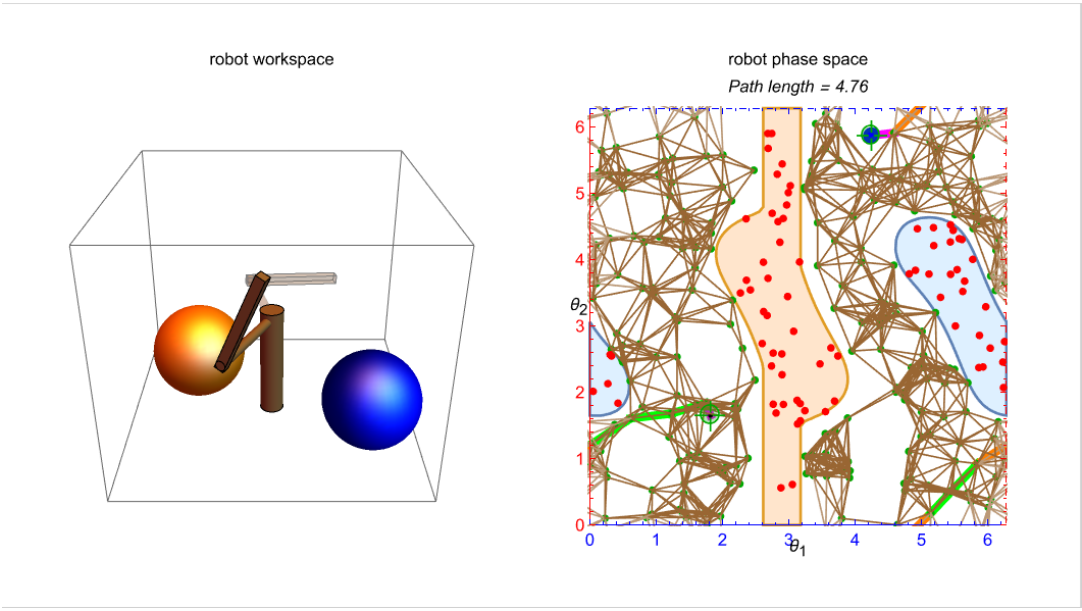
Since it is difficult to analytically calculate a true roadmap, we look for methods to approximately construct a roadmap. One type of approximate roadmap is the probabilistic roadmap, or the PRM for short. The PRM is constructed from a set of configurations sampled from the C_{space}, and it can be called a probabilistic roadmap because, as the number of samples tends to infinity,

the likelihood that the graph is a true roadmap goes to 100%. The main advantage of a PRM graph over a grid-based graph is that the structure of the free C_{space} is generally captured by the PRM with many fewer nodes than with a grid graph. Thus, PRMs are more interesting to use in high dimensional $C_{space}s$ compared to the grid-based method introduced earlier.

To contruct a PRM we go through the following steps :

- Generate N samples of the free C_{space}, whether by uniformly randomly sampling the C_{space} and only keeping the sample if it is collision-free, or by going with a non-uniform sampling strategy that can be used to increase the likelihood that the PRM is able to represent narrow passageways in the C_{space} with a smaller number of samples.
- For each node (sample generated earlier), find a set of K nearby nodes and try to connect them with the original node using a simple local path planner which does not attempt to avoid obstacles (straight line for instance).
- Check if the paths proposed by the local planner are collision free and add and edge between each two nodes that are connected with a collision free straight line.

At the end of the above algorithm, we should end up with a graph that approximately represents the free C_{space} depending on the number of sample N, the number of neighbours K, the sample algorithm and the local path planner. Once we have pre-processed the C_{space} by generating the PRM, we can solve different path planning problems by connecting different start and goal configurations to the PRM and use a grid search algorithm like A* to solve the problem. Thus, the PRM planner is usually thought of as a multiple-query planner. We invest time to have a good representation of the C_{space} to then be able to accurately solve multiple planning problems.



7.3 Conclusion

- Sampling-based planners attempt to solve planning problems based on an approximate representation of the C_{space} rather than using the actual C_{space} which can be expensive to compute.

- Sampling-based planners are probabilistic complete, and their solutions are not optimal.
- Sampling-based methods are designed to solve high dimensional path planning problems.
- Modifying the local path planner and the sampling algorithm gives a lot of flexibility to customize the original sampling-based methods.

References

[1] Kevin M. Lynch and Frank C. Park. *MODERN ROBOTICS MECHANICS, PLANNING, AND CONTROL.* December 30, 2019

[2] P. M. Hubbard. *Approximating polyhedra with spheres for time-critical collision detection.* ACM Transactions on Graphics, July 1996

[3] B.K. Patle, Ganesh Babu L, Anish Pandey, D.R.K. Parhi, A. Jagadeesh. *A review: On path planning strategies for navigation of mobile robot.* 27 April 2019

[4] Steven M. LaValle and James J. Kuffner, Jr. *Randomized Kinodynamic Planning.* May 1, 2001

[5] Sertac Karaman and Emilio Frazzoli. *Sampling-based algorithms for optimal motion planning.* Jun 22, 2011

[6] Lydia E. Kavralu, Petr Svestka, Jean-Claude Latombe, and Mark H. Overmars. *Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces.* AUGUST 1996