Autonomous and Mobile Robotics

FP6. Enhancing kinodynamic RRT using CBF-based steering

G.Giunta E.Nicotra A.Paggetti

Dipartimento di Ingegneria Informatica Automatica e Gestionale
Antonio Ruberti
Control engineering
Sapienza Università di Roma

Academic year 2020/2021

Professor G. Oriolo
Supervisor P. Ferrari

Contents

1 Introduction

When dealing with autonomous mobile robots, motion planning is one of the main issue to handle. It is even more problematic in case of non-holonomic constraints, which need to be satisfied in addition with the trajectory requirements. Among the various types of probabilistic planning algorithms, Rapidly-exploring Random Tree (**RRT**) is one of the most powerful at the moment. It requires to be able to perform a Collision Checking (**CC**) in the workspace in particular points, to make sure that the resulting path is collision free.

An alternative procedure which is being used lately is with Control Barrier Functions (**CBF**). This approach requires the use of a particular function which allows to compute a control input producing a collision free path through the resolution of an optimization problem. Instead of executing the **CC**, this method will solve an optimization problem.

The performance in solving a motion planning problem of both methods are evaluated. The environment has been built with an increasing number of obstacles to increase the complexity of the problem.

In this project we consider the motion palnning problem for the class of mobile Robots subject to non-holonomic constraints whose kinematic model can be written as a non linear affine system.

$$\dot{x} = f(x) + g(x)u \tag{1}$$

In particular we consider the case of unicycle. The motion planning problem is formally described as follows.

The robot is placed in an environment with obstacles. The task of the robot is to reach a planar goal region G of radius r_G, avoiding the obstacles. Then the problem consists on finding a configuration space path $\boldsymbol{q(s)}$, $s \in [0, 1]$ with associated history of control inputs $\boldsymbol{u(s)}$, $s \in [0, 1]$ such that:

- $\boldsymbol{q(s)} \in \mathscr{C}_{free} \;\; \forall s \in [0, 1]$
- $\boldsymbol{u} \in [\boldsymbol{u_{min}}, \boldsymbol{u_{max}}] \;\; \forall s \in [0, 1]$
- $\boldsymbol{q(1)} \in G$

2 Background on control barrier functions

It is fundamental to focus on safety and reliability of motion planning algorithms. In our context safety is defined as a minimum clearance from obstacles, so it is possible to define a safe set $\mathscr{C}_s$ that contains all the configurations that satisfies the minimum clearance. $\mathscr{C}_s \in \mathscr{C}_{free}$, that is the subset of robot configurations that do not cause collision with the obstacles. Once safety is defined, then it can be maintained by defining safe invariant sets. In general an invariant set is defined as such if all the evolutions starting on it remain on

it.
Consider the simple dynamical system

$$\dot{x} = f(x) \tag{2}$$

Where $\boldsymbol{x} \in \mathbb{R}^n$ and $f : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ vector field.
Given a smooth function $h(\boldsymbol{x}) : \mathbb{R}^n \longrightarrow \mathbb{R}$, $\mathscr{C}_s$ is defined as follows:

$$\mathscr{C}_s \coloneqq \{\boldsymbol{x} \in \mathbb{R}^n : h(\boldsymbol{x}) \geq 0\} \tag{3}$$

The necessary and sufficient conditions for invariance are given by the Nagumo's theorem:

$$\mathscr{C}_s \text{ is invariant } \iff \dot{h}(\boldsymbol{x}) \geq 0 \quad \forall \boldsymbol{x} \in \partial\mathscr{C}_s \tag{4}$$

The intuition behind these conditions is that, if the system is on the boundary of the safe set $\mathscr{C}_s$, $h(\boldsymbol{x})$ needs to increase in order to come back inside of it, and so its derivative needs to be greater than zero.
The conditions can be extended to the controlled dynamical system

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u} \tag{5}$$

Where the controller $\boldsymbol{u} \in \mathbb{R}^m$ can be exploited to achieve invariance of a certain set. In particular the condition become

$$\exists \quad \boldsymbol{u} \quad s.t. \quad \dot{h}(\boldsymbol{x}, \boldsymbol{u}) \geq 0 \implies \mathscr{C}_s \text{ is invariant} \tag{6}$$

It can be noted that the conditions given above are defined only on the boundary of $\mathscr{C}_s$ and in the case of control dynamical system are only sufficient.
However, these conditions are stronger than necessary. Then in order to extend the necessary and sufficient conditions to the entire set $\mathscr{C}_s$ it is possible to enhance it by using the class-$\mathscr{K}_\infty$ functions.

Def. A continuous function $\alpha : [0, a] \longrightarrow [0, \infty]$ is said to belong to class-$\mathscr{K}$ if it is strictly increasing and is s.t. $\alpha(0) = 0$

Def. A continuous function $\alpha : [0, \infty) \longrightarrow [0, \infty]$ is said to belong to class-$\mathscr{K}_\infty$ if it belongs to class-$\mathscr{K}$ and is s.t. $\lim_{r \to \infty} \alpha(r) = \infty$.

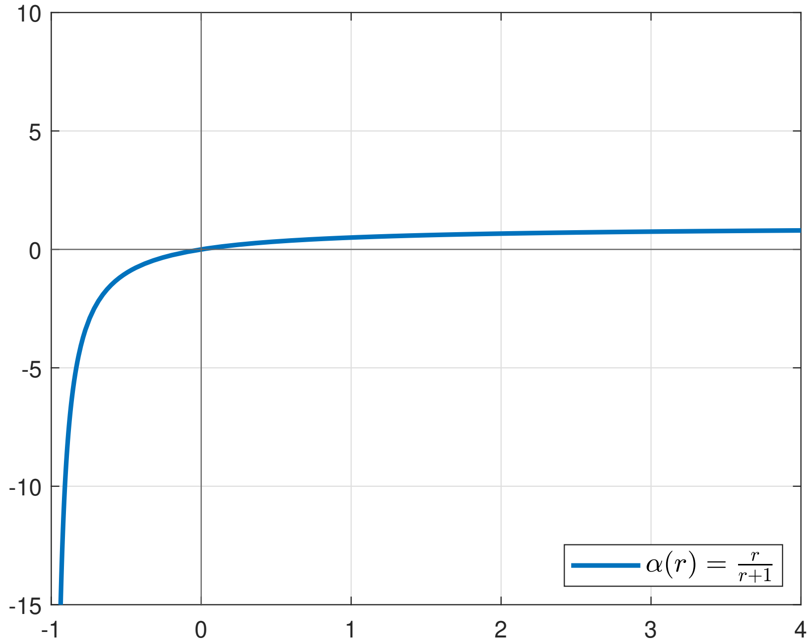


Figure 1: Example of Class-$\mathscr{K}$ function

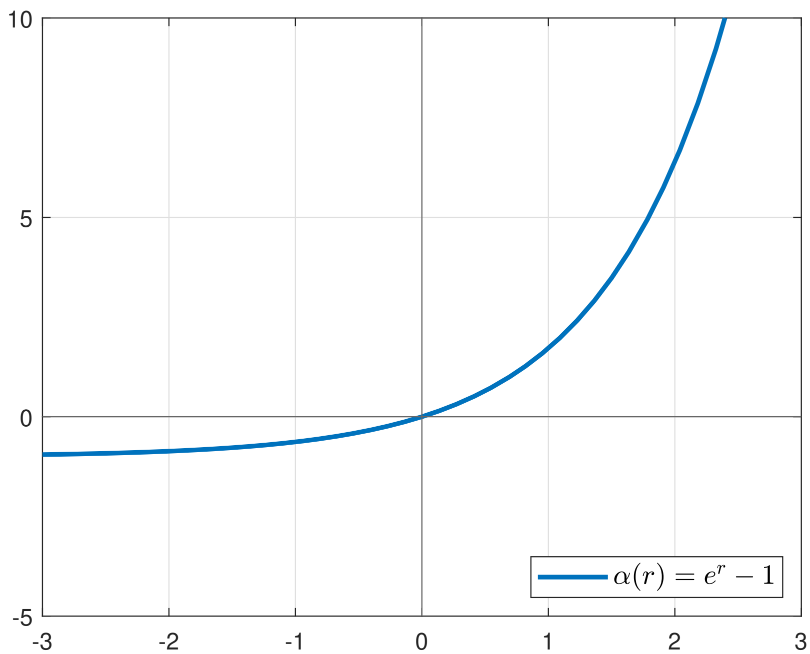


Figure 2: Example of Class-$\mathscr{K}_\infty$ function

The necessary and sufficient conditions are re-written as follows:

$$\exists \quad \boldsymbol{u} \quad s.t. \quad \dot{h}(\boldsymbol{x},\boldsymbol{u}) \geq -\alpha(h(\boldsymbol{x})) \iff \mathscr{C}_s \text{ is invariant} \tag{7}$$

Where $\alpha(\cdot)$ is an extended class-$\mathscr{K}_\infty$ function.

At this point we can define the control barrier functions.

Def: Let $\mathscr{C} \subset \mathscr{D} \subset \mathbb{R}^n$ be the superlevel set of a continuously differentiable function h : $\mathscr{D} \rightarrow \mathbb{R}$, then h is a control barrier function (**CBF**) if there exists an extended class-$\mathscr{K}_\infty$ function α such that for the control system (2):

$$\sup_{u \in \mathbb{R}^m} [L_f h(\boldsymbol{x}) + L_g h(\boldsymbol{x})\boldsymbol{u}] \geq -\alpha(h(\boldsymbol{x})) \tag{8}$$

$\forall \boldsymbol{x} \in \mathscr{D}$.

It is important to underline the fact that the **CBF** not only guarantees invariance with respect to $\mathscr{C}_s$ but also convergence to this set. As a matter of fact it might happen, due to external disturbances, that the system leaves the safe set, however thanks to a **CBF** based controller it will be driven again in $\mathscr{C}_s$. Given a reference control input u_{ref}, it is possible to compute the control input u, that is as close as possible to u_{ref} while respecting the safety requirement embodied by the CBF by solving the following QP problem:

$$\begin{cases} \boldsymbol{u}(\boldsymbol{x}) = \underset{\boldsymbol{u} \in \mathbb{R}^m}{\operatorname{argmin}} \dfrac{1}{2} \|\boldsymbol{u} - \boldsymbol{u_{ref}}\|^2 \\ s.t \quad L_f h(\boldsymbol{x}) + L_g h(\boldsymbol{x})\boldsymbol{u} \geq -\alpha(h(\boldsymbol{x})) \end{cases} \tag{9}$$

3 Considered motion planners

3.1 Classic RRT

RRT is one of the most powerful algorithm that nowadays allows to compute collision free path in a workspace with obstacles. It is a single-query probabilistic method and so it does not generate a roadmap of the free configuration space $\mathscr{C}_{free}$ but only a subset of it, needed to solve the problem.
We suppose to know in advance the entire workspace and the exact position, dimension and shape of the obstacles.

The RRT algorithm runs off-line, and generates as output a feasible collision free path to the goal region. RRT is a motion planning method which does not require the computation of $\mathscr{C}_{obs}$ (i.e. the image of the obstacles in the Configuration space) which is a very expensive procedure, but in its classic form requires a Collision-Checking algorithm. RRT is a probabilistic method, which belongs to the family of sampling-based methods and extracts its samples randomly from the Configuration Space, with a uniform probability distribution.

Using RRT we can build a tree composed by collision free nodes, which are linked by a collision free path. This tree has its root on the initial configuration of the robot. RRT keeps building the tree until it generates a configuration in the goal region. At this point it simply extracts the collision free path from initial configuration to goal configuration from its nodes.

The RRT algorithm for non-holonomic robots is a slightly adapted version of the general one.
Since not all the subpath in the configuration space are admissible from a certain configuration, a group of inputs $U = \{u_1 ... u_n\}$ are defined, to generate feasible behaviours, called Motion Primitives.

The adapted-RRT algorithm is composed by the following steps:

1. set the tree root as the starting configuration
2. extract randomly a configuration $\boldsymbol{q_{rand}}$ from the Configuration Space, with Uniform Probability Distribution
3. search in the tree the closest node ($\boldsymbol{q_{near}}$) to $\boldsymbol{q_{rand}}$
4. find the new configuration $\boldsymbol{q_{new}}$ by applying one of the primitive $u \in U$ to the robot, starting from the configuration $\boldsymbol{q_{near}}$ and integrating the system under the chosen u for an interval of integration of ΔS
5. check if there is collision between the robot and the obstacles in $\boldsymbol{q_{new}}$ and also in the path traveled from $\boldsymbol{q_{near}}$ to $\boldsymbol{q_{new}}$.
6. if there is collision discard $\boldsymbol{q_{new}}$, otherwise add it to the tree
7. return to (1)

It is clear that the main difference with respect to the standard RRT is that, because of the non-holonomic constraint, it is not possible to connect $\boldsymbol{q_{near}}$ to $\boldsymbol{q_{rand}}$ (and so to generate $\boldsymbol{q_{new}}$) directly by using a rectilinear path but instead it is necessary to use the motion primitives in order to not violate the constraint.

The algorithm stops if the maximum number of iterations are reached or if the goal region is reached.

The Pseudo-code of the algorithm is given below.

Algorithm 1 RRT

procedure INITIALIZATION
    $r_G \leftarrow 0.2$ ▷ *radius of the circular goal region*
    $found \leftarrow false$ ▷ *whether solution is found or not*
    $U \leftarrow \{u_1...u_n\}$ ▷ *set of primitives*
    $i \leftarrow 0$ ▷ *current iteration*
    $i_{max} \leftarrow 30000$
    $\boldsymbol{q_{start}} \leftarrow$ *'starting robot configuration'*
    $T \leftarrow \boldsymbol{q_{start}}$ ▷ *tree structure*
    $\boldsymbol{q_{goal}} \leftarrow null$ ▷ *it gets a value only when a solution is found*

procedure MAINLOOP
    while $(i \leq i_{max})$ & $\boldsymbol{!}found$ **do**
        $\boldsymbol{q_{rand}} \leftarrow getRandomConfiguration()$
        $\boldsymbol{q_{near}} \leftarrow getNearestNode(T, \boldsymbol{q_{rand}})$
        $\boldsymbol{u_{ref}} \leftarrow choosePrimitiveRandomly(U)$
        $\boldsymbol{q_{new}} \leftarrow computeNewConfiguration(\boldsymbol{q_{near}}, \boldsymbol{u_{ref}})$
        if $isCollisFree(\boldsymbol{q_{near}}, \boldsymbol{q_{new}})$ **then**
            $T.Add(\boldsymbol{q_{new}})$
            if $\boldsymbol{q_{new}} \in G$ **then**
                $found \leftarrow true$
                $\boldsymbol{q_{goal}} = \boldsymbol{q_{new}}$

procedure BUILDPATHTOGOALREGION
    $\boldsymbol{q_{curr}} \leftarrow \boldsymbol{q_{goal}}$
    $path.Add(\boldsymbol{q_{curr}})$
    while $\boldsymbol{q_{curr}} \; != \; \boldsymbol{q_{start}}$ **do**
        $\boldsymbol{q_{curr}} \leftarrow \boldsymbol{q_{curr}.parent}$
        $path.Add(\boldsymbol{q_{curr}})$
    $path.reverse()$

In the real implementation we kept track not only of the configurations $\boldsymbol{q}$ of the robot but also of the inputs that we applied to reach that configuration.

Some comments about the trivial functions presented on **Algorithm** 1.

The function *getRandomConfiguration()* simply picks randoms numbers with uniform probability distribution from the Configuration Space.

The function *getNearestNode(tree,* $\boldsymbol{q_{rand}}$*)* goes through the tree and compute the cartesian distance from each $\boldsymbol{q}$ of the tree to $\boldsymbol{q_{rand}}$. It returns the $\boldsymbol{q}$ belonging to the tree which is the closest to $\boldsymbol{q_{rand}}$.

The function $computeNewConfiguration(\boldsymbol{q_{near}}, \boldsymbol{u_{ref}})$ returns the new reached configuration $\boldsymbol{q_{new}}$ when we apply the inputs $\boldsymbol{u_{ref}}$ to the unicycle.

The function $isCollisFree(\boldsymbol{q_{near}}, \boldsymbol{q_{new}})$ checks if the path from $\boldsymbol{q_{near}}$ to $\boldsymbol{q_{new}}$ collide with an obstacle. This is done through the collision checking routine of V-rep considering the entire path divided into 50 sub-intervals and performing the check on the end point of each sub-intervals.

Between the various steps it is possible to recognise that one of the most expensive operation is the Collision-Checking, for this reason an alternative will be discussed in the following. The alternative is to remove the CC procedure and use instead another method to avoid collisions, which involves the use of Control Barrier Functions (CBFs).

3.2 CBF-based RRT

3.2.1 Main Idea

In order to enhance the RRT algorithm and avoid collision checking, one possibility consists in computing the control inputs by using CBF to modify the provided primitives.

Instead of discarding colliding nodes generated through Motion Primitive $\boldsymbol{u_{ref}}$, the **RRT-CBF** algorithm computes the closest control input $\boldsymbol{u}$ to $\boldsymbol{u_{ref}}$ in (9). So the collision checking is no longer needed. In general the resolution of **QP** problem will increase the computational time making the **RRT-CBF** slower, however it will generate collision-free trajectories every time that the problem is feasible.

Since constraints are present, such as the **CBF** or actuator constraints due to actuators saturation, the problem may be unfeasible and so this is the only case in which this method fails on find a control input. Actually, we have also set some other constraints such as a minimum value of the linear velocity, to avoid, that too many nodes are composed by only rotations in the place.

Experimental results shows that, even if all these constraints are considered, the optimization problem is still feasible.

3.2.2 Robot Model

We developed the CBF-based RRT planner with reference to the unycicle, whose kinematic model is:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega \tag{10}$$

where $(x, y) \in \mathbb{R}^2$ corresponds to the Cartesian coordinates of the contact point of the wheel with the ground and θ is the orientation of the wheel with respect to the x axis.

Clearly the input vector in this case is $\boldsymbol{u} = [v, \omega]^T$ and the dynamics can be gathered as in equation (5).

Note that the unicycle is a driftless system, in fact $f(\boldsymbol{x}) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ and $g(\boldsymbol{x}) = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix}$

3.2.3 Choice of CBF

The safe set is defined as follows:

$$\mathscr{C}_s \coloneqq \left\{ \boldsymbol{x} \in \mathbb{R}^2 : h(\boldsymbol{x}) \geq \tau \right\} \tag{11}$$

Where $h(\boldsymbol{x})$ is defined as the distance between the robot representative point and the nearest obstacle.

$$h(\boldsymbol{x}) = \sqrt{(\boldsymbol{x} - \boldsymbol{x_{obs}})^2 + (\boldsymbol{y} - \boldsymbol{y_{obs}})^2} \tag{12}$$

The coordinates $(\boldsymbol{x_{obs}}, \boldsymbol{y_{obs}})$ indicates the point of the obstacle nearest to the robot.
Being $h(\boldsymbol{x}) - \tau > 0 \, \forall x \in \mathscr{C}_s$, the simplest choice for the class-$\mathscr{K}_\infty$ function that makes $h(\boldsymbol{x})$ a **CBF**, according to (8), is a constant.

τ is a non-negative safe distance, including the unicycle footprint size on it. In particular, we included on the safe distance, the radius of the circle surrounding the unicycle.

So the **CBF** is chosen as $h(\boldsymbol{x}) - \tau$.

In conclusion, taking into account the choice of α, the invariance condition is reduced to:

$$\sup_{\boldsymbol{u} \in \mathbb{R}^m} [L_g h(\boldsymbol{x})\boldsymbol{u}] \geq -\alpha h(\boldsymbol{x}) \tag{13}$$

3.2.4 Optimization problem

In order to solve the optimization problem in (9) with the *qpOASES* solver, it is necessary to rewrite it in the following form:

$$\begin{cases} \min\limits_{\boldsymbol{u} \in U} & \dfrac{1}{2}\boldsymbol{u}^T H \boldsymbol{u} - \boldsymbol{u}^T R \\ s.t & l_b A \leq A\boldsymbol{u} \leq u_b A \\ & \boldsymbol{l_b} \leq \boldsymbol{u} \leq \boldsymbol{u_b} \end{cases} \tag{14}$$

For sake of completeness the passages leading from form (9) to (14) are reported below.

$$\begin{aligned}\frac{1}{2}\|\boldsymbol{u}-\boldsymbol{u_{ref}}\|^2 &= \frac{1}{2}((v-v_{ref})^2+(\omega-\omega_{ref})^2 \\ &= \frac{1}{2}(v^2-2vv_{ref}+v_{ref}^2+\omega^2-2\omega\omega_{ref}+\omega_{ref}^2) \\ &= \frac{1}{2}(v^2-2vv_{ref}-2\omega\omega_{ref}+constant) \\ &= \frac{1}{2}\boldsymbol{u}^T H\boldsymbol{u}-\boldsymbol{u}^T R\end{aligned} \tag{15}$$

Where $H = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$ and $R = \begin{bmatrix}v_{ref}\\ \omega_{ref}\end{bmatrix}$.

$\boldsymbol{u} \in U$ is bounded by a maximal and minimal values taken from the maximum and minimum value of the primitives.

The constraints need to be expressed as lower and upper bounds

$$l_b A \leq A\boldsymbol{u} \leq u_b A \tag{16}$$

Starting from (13):

$$\begin{aligned}h(\boldsymbol{x}) &= \sqrt{(\boldsymbol{x}-\boldsymbol{x_{obs}})^2+(\boldsymbol{y}-\boldsymbol{y_{obs}})^2}-\tau \\ \dot{h}(\boldsymbol{x}) &= \begin{bmatrix}\frac{\partial h(\boldsymbol{x})}{\partial x} & \frac{\partial h(\boldsymbol{x})}{\partial y} & \frac{\partial h(\boldsymbol{x})}{\partial \theta}\end{bmatrix}\begin{bmatrix}\cos\theta & 0\\ \sin\theta & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix}\end{aligned} \tag{17}$$

With:

$$\begin{aligned}\frac{\partial h(x)}{\partial x} &= \frac{1}{\sqrt{(x-x_{obs})^2+(y-y_{obs})^2}}(x-x_{obs}) \\ \frac{\partial h(x)}{\partial y} &= \frac{1}{\sqrt{(x-x_{obs})^2+(y-y_{obs})^2}}(y-y_{obs}) \\ \frac{\partial h(x)}{\partial \theta} &= 0\end{aligned} \tag{18}$$

As it can be seen $h(x)$ does not depend on θ so matrix A has the second column composed by zeros only. This implies that the angular velocity ω does not appear in the equations at all, and for this reason the optimization problem would not consider it as a variable.

In order to include the input ω in the optimization problem instead of computing the distance between the center of rotation of the robot and the obstacle a small offset is applied along the sagittal axis as in figure (3).

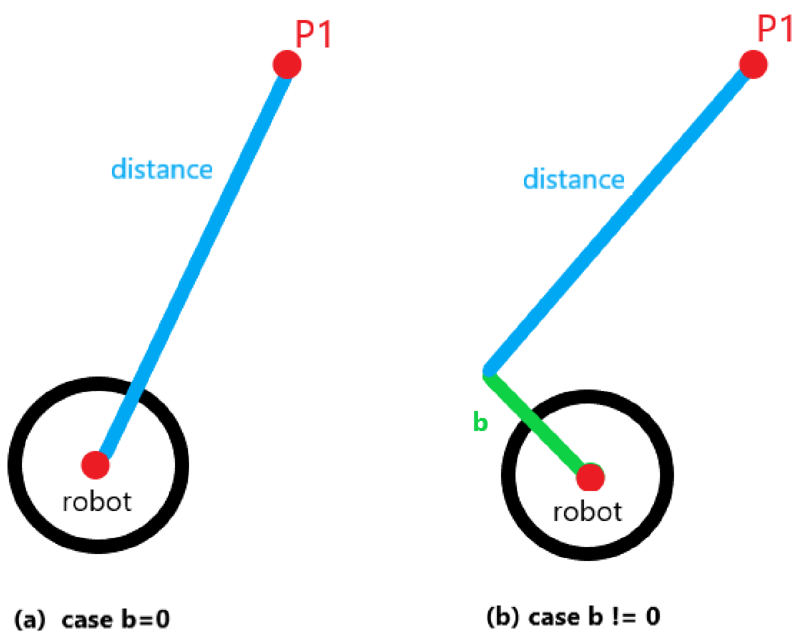


Figure 3: Choice of b

Intuitively, from (Fig.3-a) it is possible to see that considering the representative point coincident with the center of rotation, implies that distance from any point does not change depending on the orientation. Instead if we consider (Fig.3-b), we immediately see that the distance from the robot to the point is also influenced by the orientation of the robot.

For this reason we moved the representative point along the sagittal axis in the forward direction of a quantity $|b|$.
The new considered point is represented by:

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} x + b\cos\theta \\ y + b\sin\theta \end{bmatrix} \tag{19}$$

yielding:

$$
\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & -b\sin\theta \\ \sin\theta & b\cos\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{20}
$$

According to these new coordinates the derivative of the distance $\dot{h}$ become:

$$
\dot{h}(x) = \begin{bmatrix} \frac{\partial h(x)}{\partial x} & \frac{\partial h(x)}{\partial y} & \frac{\partial h(x)}{\partial \theta} \end{bmatrix} \begin{bmatrix} \cos\theta & -b\sin\theta \\ \sin\theta & b\cos\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{21}
$$

It can be noted that this time the matrix A depends also on θ, allowing the optimization problem to compute its optimal value

$$
\begin{aligned}
\frac{\partial h}{\partial x} &= \frac{x - x_{obs} + b\cos(\theta)}{\sqrt{(x - x_0 + b\cos(\theta))^2 + (y - y_0 + b\sin(\theta))^2}} \\
\frac{\partial h}{\partial y} &= \frac{y - y_{obs} + b\sin(\theta)}{\sqrt{(x - x_{obs} + b\cos(\theta))^2 + (y - y_{obs} + b\sin(\theta))^2}} \\
\frac{\partial h}{\partial \theta} &= \frac{b\,(y\cos(\theta) - y_{obs}\cos(\theta) - x\sin(\theta) + x_{obs}\sin(\theta))}{\sqrt{(x - x_{obs} + b\cos(\theta))^2 + (y - y_{obs} + b\sin(\theta))^2}}
\end{aligned} \tag{22}
$$

Then $\quad A = \begin{bmatrix} \frac{\partial h(x)}{\partial x} & \frac{\partial h(x)}{\partial y} & \frac{\partial h(x)}{\partial \theta} \end{bmatrix} \begin{bmatrix} \cos\theta & -b\sin\theta \\ \sin\theta & b\cos\theta \\ 0 & 1 \end{bmatrix} \quad$ and $\quad l_b A = -\alpha(h(x) - \tau)$

Giving H, R, A and $l_b A$ as inputs to the *qpOASES* solver it will return the optimal control motion for each step.

Anyway, we have to point out that the point $[x_{obs}, y_{obs}]$ is taken only once and kept constant in the optimization problem. But, in a real situation, the following fact could occur:

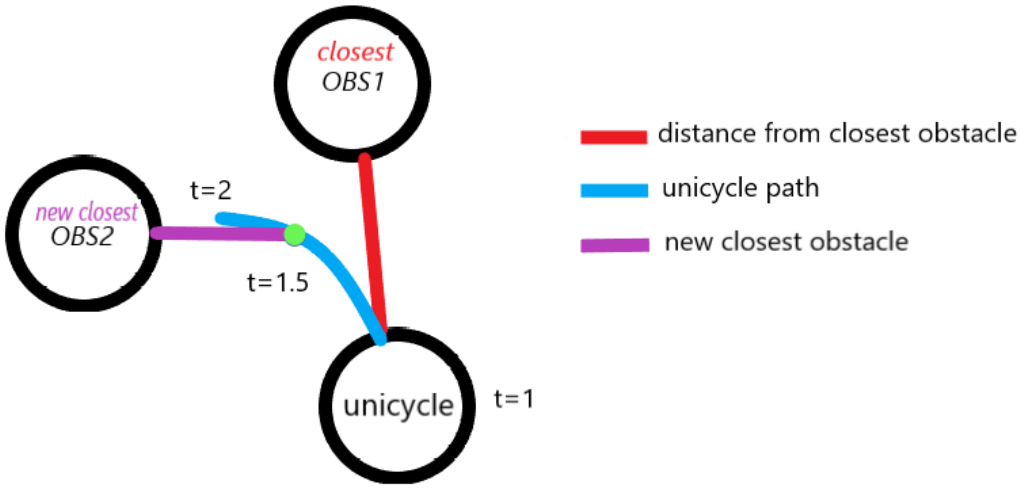


Figure 4: CBF - example of bad workspace

The closest obstacle, that is *OBS1* at $t = 1$, could change and become *OBS2* at mid integration time interval depending on the control input we choose. If the integration time is too big, in the limit case, the unicycle could also crash into the *OBS2*, since we considered obstacle one as the closest.

So, in general, to avoid this situation, the distance between an obstacle and our robot, should be expressed, totally as function of x. In this way, since $\dot{x}$ depends on u, the closest obstacle will change by changing u, and the optimization problem could adjust the computed input consequently. For this reason, since we considered $[x_{obs}, y_{obs}]$ as a fixed point, we do the assumption :

Assumption 1 : the integration time interval is small enough to not let the closest obstacle change. (i.e. the closest obstacle does not change during the integration intervall)

The pseudo-algorithm of the CBF-based RRT planner is given below:

Algorithm 2 RRT with CBFs

procedure INITIALIZATION
    $r_G \leftarrow 0.2$ ▷ *circular goal region*
    $found \leftarrow false$ ▷ *whether solution is found or not*
    $U \leftarrow \{u_1 ... u_n\}$
    $i \leftarrow 0$ ▷ *current iteration*
    $i_{max} \leftarrow 30000$
    $\boldsymbol{q_{start}} \leftarrow$ *'starting robot configuration'*
    $d_{safe} \leftarrow r_{uni} + 0.1$ ▷ *safe distance = unicycle radius + 0.1*
    $T \leftarrow \boldsymbol{q_{start}}$ ▷ *tree structure*
    $\boldsymbol{q_{goal}} \leftarrow null$ ▷ *it gets a value only when a solution is found*

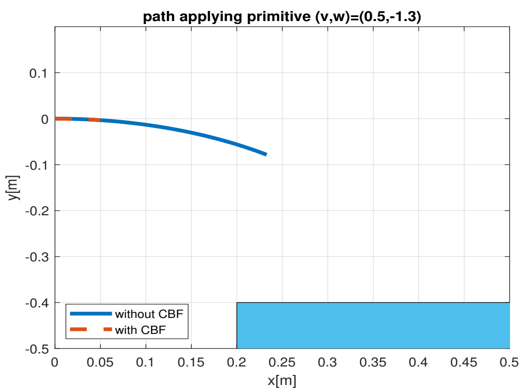
procedure MAINLOOP
    while $(i \leq i_{max})$ & $\boldsymbol{!}found$ **do**
        $\boldsymbol{q_{rand}} \leftarrow getRandomConfiguration()$
        $\boldsymbol{q_{near}} \leftarrow getNearestNode(T, \boldsymbol{q_{rand}})$
        $\boldsymbol{u_{ref}} \leftarrow choosePrimitiveRandomly(U)$
        $\boldsymbol{u_{cbf}} \leftarrow getUControlBarrierFunction(\boldsymbol{q_{near}}, \boldsymbol{u_{ref}})$ ▷ *additional function w.r.t.* ***Algorithm*** *2*
        $\boldsymbol{q_{new}} \leftarrow computeNewConfiguration(\boldsymbol{q_{near}}, \boldsymbol{u_{cbf}})$
        $tree.Add(\boldsymbol{q_{new}})$
        if $q_{new} \in G$ **then**
            $found \leftarrow true$
            $\boldsymbol{q_{goal}} = \boldsymbol{q_{new}}$

procedure BUILDPATHTOGOALREGION
    $\boldsymbol{q_{curr}} \leftarrow \boldsymbol{q_{goal}}$
    $path.Add(\boldsymbol{q_{curr}})$
    while $q_{curr}! = q_{start}$ **do**
        $\boldsymbol{q_{curr}} \leftarrow \boldsymbol{q_{curr}.parent}$
        $path.Add(\boldsymbol{q_{curr}})$
    $path.reverse()$

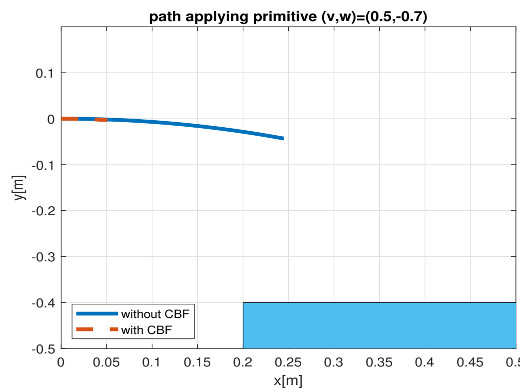
Algorithm 2 is very similar to **Algorithm** 1. The main differences are the absence of the function $isCollisFree(\boldsymbol{q_{near}}, \boldsymbol{q_{new}}, \boldsymbol{u_{ref}})$ and the additional computation on the function $getUControlBarrierFunction(\boldsymbol{q_{near}}, \boldsymbol{u_{ref}})$ to which we dedicated section 3.2.4.

3.2.5 Effect of CBF on motion primitives

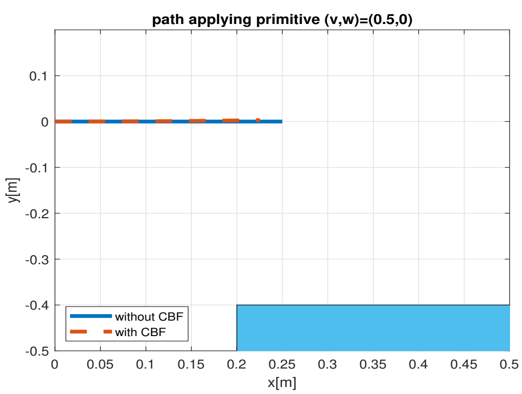
Each step of the RRT algorithm produces a control input modified by the optimization problem with control barrier function approach. In the case on which we are far from obstacles the optimization problem returns the same control inputs of the primitive. In the following is showed the influence of the control barrier functions approach on the primitives, when the robot is close to an obstacle. Because of the very short time interval, and the **Assumption 1** , the influence of a single obstacle is considered. To show the actual effect of the CBFs we compare the input reference given to the optimization problem and the produced input in the same plot.



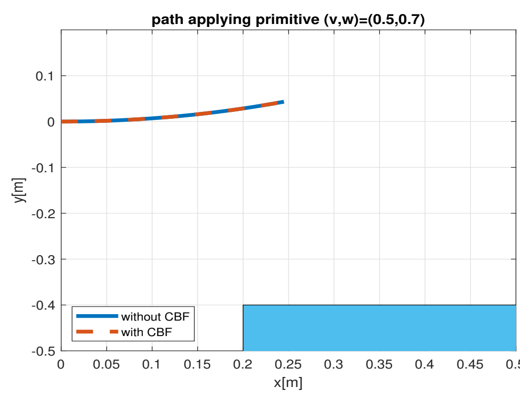
Primitive 1



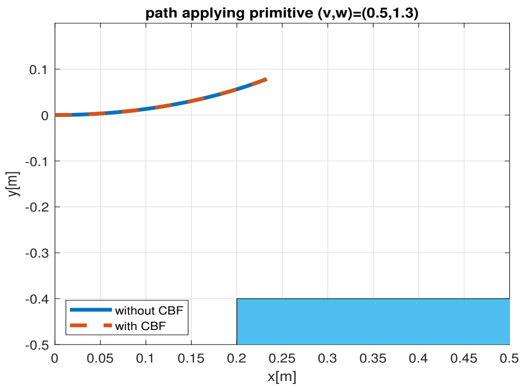
Primitive 2



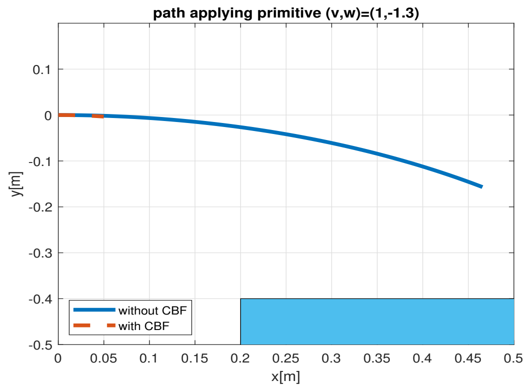
Primitive 3



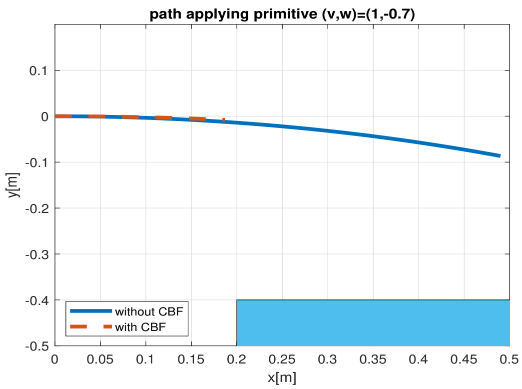
Primitive 4



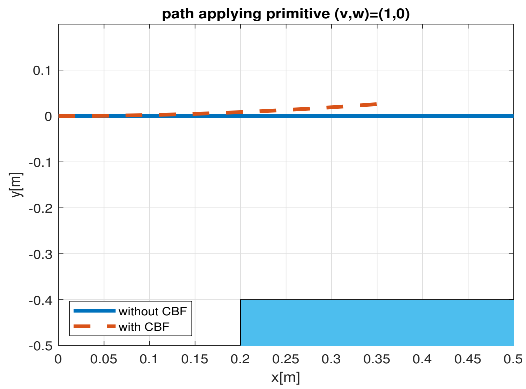
Primitive 5



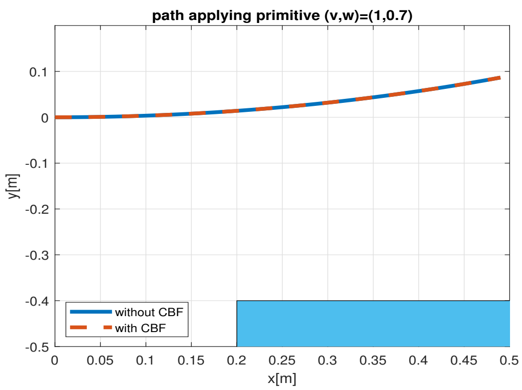
Primitive 6



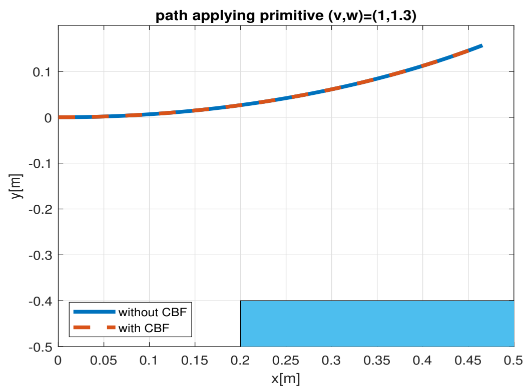
Primitive 7



Primitive 8



Primitive 9



Primitive 10

4 Simulation Results

In the following the two different **RRT** algorithms are used in order to solve a motion planning problem for a unicycle. Then the results of the two methods are compared. The simulations are performed inside an environment of increasing complexity. In order to compare fairly the **RRT-CBF** and the **RRT** we added a third simulation in which the classic algorithm has to guarantee a minimum clearance as in the **CBF** case. To achieve a minimum clearance we added a bounding box that enlarges the unicycle's footprint.

4.1 Simulation Settings

The considered Workspace has the following dimensions:

- $[x_{min}, x_{max}] = [-2.5, 2.5]$ [m]
- $[y_{min}, y_{max}] = [-2.5, 2.5]$ [m]

The primitives are selected according to the following values of the command inputs:
$v \in \{0.5, 1\}$ and $\omega \in \{-1.3, -0.7, 0, 0.7, 1.3\}$. The possible input combinations are in table (1)

v	0.5	0.5	0.5	0.5	0.5	1	1	1	1	1
ω	-1.3	-0.7	0	0.7	1.3	-1.3	-0.7	0	0.7	1.3

Table 1: Motion Primitives

The other simulation parameters are:

- Time interval (integration of primitives) : **0.5 [s]**
- Radius goal region : **0.1 [m]**
- RRT primitive choose method : **random**
- Collision checking : **active**
- Max number of iterations : **30000**

And in the version with RRT with CBF, the following parameters are added:

- Collision checking : **disabled**
- Constant α (used as class-$\mathscr{K}_\infty$ function for control barrier function) : **2**
- Minimum v module accepted : **0.1 [m/s]**
- safe distance τ: **unicycle radius + 0.1 [m]**

Some comments on these parameters are necessary to have a proper understanding.

The parameter *Minimum v module accepted* assures that the v component of the control input is greater than 0.1. This parameter is inserted directly on the optimization problem constraints, since we assume to not use backward motion, and we try to avoid to find solutions where the only motion performed is rotation around the vertical axis. The reason is that most of times, when the unicycle is close to the obstacles, the Optimization problems is feasible, just by letting the unicycle rotate on its axis. For this reason, in some cases, the final path from root to goal, could include some nodes in which the unicycle rotates multiple times around its axis without moving at all. We decided to exclude this behavior.

The parameter *Constant α* correspond to the class-$\mathscr{K}_\infty$ function through which we defined the Control Barrier Function. This parameter is the key around the correct functioning of this algorithm. This choice is also linked to the choice of the primitives, the integration time, and the safe distance. Once we fixed the time interval and the primitives to be used, we did a sort of tuning phase, on which we fixed the proper value of α which generate collision free paths. To do this, we implemented a regulator controller for the unicycle and we filtered the generated input commands with **CBF**. We observed collision avoidance properties with a certain range of α and depending on its value, the unicycle could get closer to the obstacles. Then, with our choice of primitives, we fixed the optimal value for α.

4.2 Simulation Environment - 5 Obstacles

The simulations were performed in the following scenario:

Figure 10: Simulations Scenario - 5 Obstacles

4.2.1 RRT with CBF

In this table we show the results of our simulations:

#	Iter.	Found	Ex. time[ms]	n° to goal	TOT n°	n° QP is unfeasible	Minimum clearance [m]
1	175	Y	150	23	128	34	0.149
2	117	Y	73	26	87	22	0.145
3	741	Y	770	31	524	87	0.145
4	268	Y	247	22	200	56	0.163
5	1802	Y	2055	35	1048	379	0.146
6	229	Y	257	25	206	7	0.162
7	192	Y	236	27	150	39	0.168
8	194	Y	259	24	160	30	0.151
9	198	Y	172	24	137	54	0.156
10	225	Y	200	23	156	50	0.154
AVG	414.1	100%	441.9	26.0	279.6	75.8	0.154

Table 2: RRT-CBF 5 obstacles

where:

- 'Iter.' are the number of iterations the algorithm takes to find a solution
- 'Found' tells if the solution has been found or not and can assume values 'Y=yes' or 'N=no'
- 'Ex. time' is the execution time of the entire path planning algorithm
- 'n° to goal' is the number of nodes necessary to go from root node to goal node
- 'TOT n°' is the total number of nodes that the algorithm has been able to found
- 'n° QP not feas.' is the number of nodes discarded because the QP problem was unfeasible

For each of this simulation we gather also other data regarding the heavier operation which is the Quadratic Problem (QP) solution:

#	Total QP time [μs]	Fastest Execution [μs]	Slowest Execution [μs]	Average Execution [μs]
1	132462	88	2976	756
2	64888	92	2424	554
3	626713	90	3335	845
4	216101	95	2941	806
5	1505758	85	3799	835
6	231086	84	2890	1009
7	218770	83	77156	1139
8	241331	83	78971	1243
9	153200	77	2573	773
10	178494	83	3206	793
AVG	356880.3	86.0	18027.1	875.3

Table 3: RRT-CBF 5 obstacles

4.2.2 RRT Classic

We run also the RRT classic algorithm in the same scenario and we got the following results:

#	Iter.	Found	Ex. time[ms]	n° to goal	TOT n°	n° collisions	Minimum clearance
1	1141	Y	1010	26	853	288	0.0805
2	446	Y	313	19	293	153	0.0614
3	357	Y	300	26	270	87	0.00168
4	464	Y	371	30	358	106	0.0114
5	554	Y	489	20	409	145	0.0124
6	259	Y	233	26	215	44	0.0915
7	199	Y	140	19	141	58	0.148
8	1248	Y	1025	19	788	460	0.0159
9	955	Y	945	26	723	232	0.0275
10	1174	Y	1075	31	836	338	0.103
AVG	679.7	Y	590.1	24.2	488.6	191.1	0.055

Table 4: RRT 5 obstacles

And for each simulation we collect also the following data about collision checking

(CC) algorithm:

#	Total CC time [μs]	Fastest Execution [μs]	Slowest Execution [μs]	Average Execution [μs]
1	704824	18	4797	617
2	254201	18	2933	569
3	256632	22	3897	718
4	303308	18	2596	653
5	404367	19	3225	729
6	205310	22	5631	792
7	122314	17	1735	614
8	691112	19	4521	553
9	715986	18	80238	749
10	745826	18	3432	635
AVG	440388.0	18.9	11300.5	662.9

Table 5: RRT 5 obstacles

4.2.3 RRT Classic with Bounding box

#	Iter.	Found	Ex. time[ms]	n° to goal	TOT n°	n° collisions	Minimum clearance
1	559	Y	991	27	270	289	0.163
2	1253	Y	3349	18	859	394	0.15
3	101	Y	281	19	83	18	0.152
4	598	Y	1218	25	334	264	0.152
5	244	Y	644	19	194	50	0.155
6	408	Y	865	26	241	167	0.152
7	713	Y	1250	19	331	382	0.169
8	131	Y	348	19	104	27	0.12
9	391	Y	1015	26	290	101	0.2
10	642	Y	812	19	217	425	0.162
AVG	504.0	Y	1077.3	21.7	292.3	211.7	0.1575

Table 6: RRT BB 5 obstacles

#	Total CC time [μs]	Fastest Execution [μs]	Slowest Execution [μs]	Average Execution [μs]
1	919852	19	5207	1645
2	2949502	21	6471	2353
3	273676	23	5002	2709
4	1127421	20	5521	1885
5	618184	24	4801	2533
6	814456	22	4699	1996
7	1136581	17	4651	1594
8	337664	23	4361	2577
9	961676	25	5136	2459
10	738494	17	5127	1150
AVG	987750.6	21.1	5097.6	2090.1

Table 7: Time RRT BB 5 obstacles

4.2.4 Comparison

A preliminary comparison is done between the three cases in which only 5 obstacles are present in the workspace. This case has the purpose to prove first that the method with CBF works well, and secondly to compare performances with RRT classic. To be able to make a proper comparison we report the average data in here:

Algorithm	Iter.	Found	Ex. T[ms]	n° to goal	TOT n°	n° collision	n° QP not feas.	Minimum clearance
RRT+CBF	414.1	100%	441.9	26.0	279.6	—	75.8	0.154
RRT	679.7	100%	590.1	24.2	488.6	191.1	—	0.0553
RRT SD	504.0	100%	1077.3	21.7	292.3	211.7	—	0.1575

Table 8: Averages comparison 5 obstacles

Algorithm	Total time [μs]	Fastest Execution [μs]	Slowest Execution [μs]	Average Execution [μs]
QP	356880.3	86.0	18027.1	875.3
CC	440388.0	18.9	11300.5	662.9
CC SD	987750.6	21.1	5097.6	2090.1

Table 9: Time averages comparison 5 obstacles

As we can see from average data, for this environment, the RRT with CBF performs much better. Both methods have found a solution in all the simulations. RRT with CBF reaches the solution with less iteration, execution time and n° of nodes to goal. It performs better also in all other parameters, for example also the number of total node created are lower, so a lower amount of memory in the computer is required.

An explanation for better performances could be that, instead of having a discretized set of primitives the control inputs can assume all the possible values bounded by the maximal magnitude of the velocities. v and ω are bounded according to their maximal and minimal values inside the set of primitives. So if the reference primitive brings to collision, a slightly modified version is adapted to make the problem feasible.

Moreover it is possible to see from the averages that the **RRT** with bounding box has the worst performances in terms of time due to the greater number of discarded nodes. In fact, by enlarging the robot's footprint a lot of primitives lead to collision so are no longer usable and there is no way of recovering such motions as in the **RRT-CBF** case.

We will see in the following that this trend is not maintained increasing the number of obstacles.

4.3 Simulation Environment - 7 Obstacles

The simulations was performed in the following scenario:

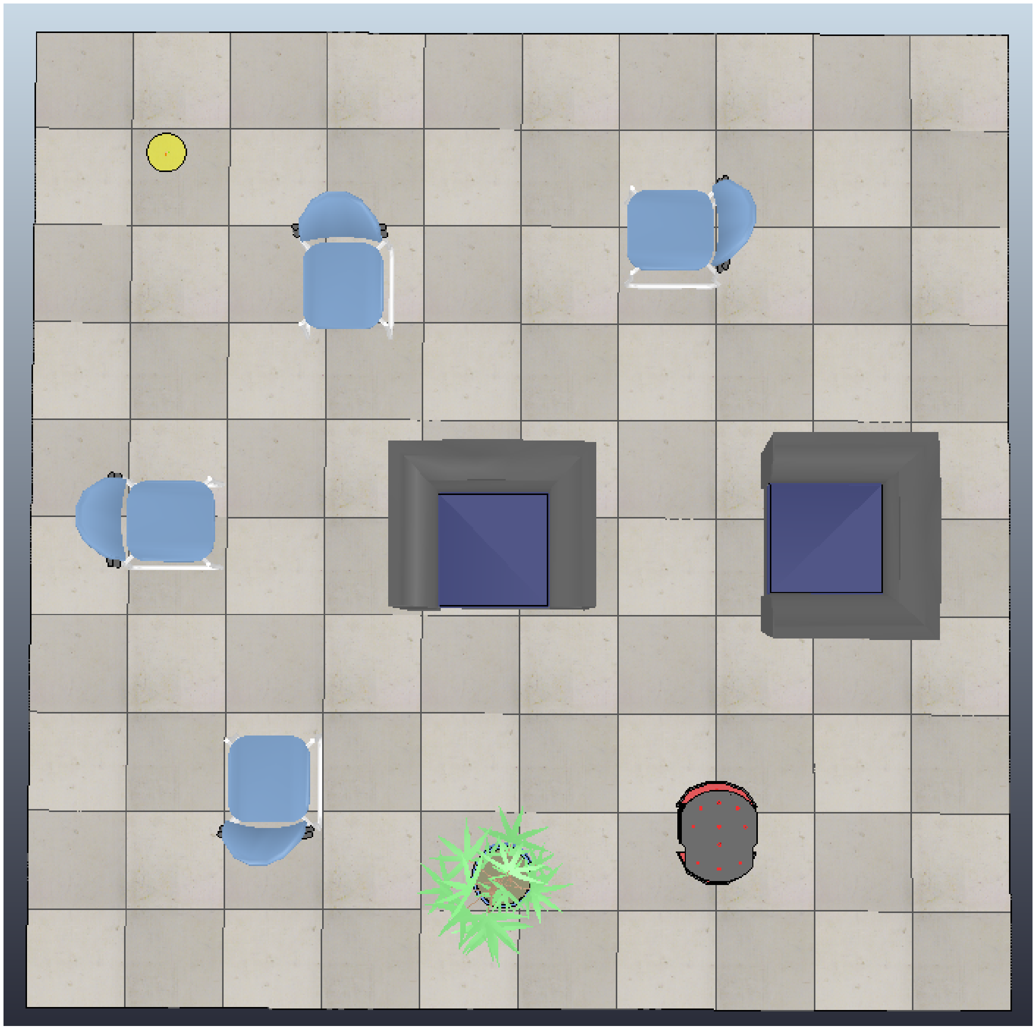


Figure 11: Simulations Scenario - 7 obstacles

The simulation parameters are not repeated since are exactly equal to the previous cases.

4.3.1 RRT with CBF

#	Iter.	Found	Ex. time[ms]	n° to goal	TOT n°	n° QP is unfeasible	Minimum clearance [m]
1	136	Y	360	18	116	18	0.155
2	1884	Y	5853	43	1173	444	0.151
3	146	Y	343	18	114	13	0.156
4	189	Y	437	30	105	78	0.123
5	237	Y	572	22	98	133	0.141
6	232	Y	692	16	157	70	0.154
7	651	Y	1758	19	346	175	0.151
8	372	Y	1183	20	224	118	0.138
9	205	Y	505	18	167	30	0.129
10	758	Y	2269	25	409	215	0.156
AVG	481.0	100%	1397.2	22.9	290.9	129.4	0.1454

Table 10: RRT-CBF 7 obstacles

#	Total QP time [μs]	Fastest Execution [μs]	Slowest Execution [μs]	Average Execution [μs]
1	348735	160	7612	2564
2	5156735	95	7513	2737
3	331236	151	5801	2268
4	422664	119	5538	2236
5	553949	151	5662	2337
6	668556	185	6848	2881
7	1659644	110	9112	2549
8	1144889	94	151890	3077
9	484743	165	6342	2364
10	2136216	166	6824	2818
AVG	1290736.7	139.6	21314.2	2583.1

Table 11: RRT-CBF 7 obstacles

4.3.2 RRT Classic

#	Iter.	Found	Ex. time[ms]	n° to goal	TOT n°	n° collisions	Minimum clearance
1	304	Y	367	21	176	128	0.00513
2	319	Y	198	21	185	134	0.0341
3	1550	Y	1354	42	794	756	0.0823
4	262	Y	199	21	170	92	0.0447
5	1381	Y	1009	21	662	719	0.00737
6	790	Y	680	21	476	314	0.00357
7	439	Y	295	21	238	201	0.0221
8	606	Y	489	21	317	289	0.000781
9	1273	Y	1025	21	687	586	0.0329
10	701	Y	500	21	399	302	0.0327
AVG	762.5	Y	611.6	23.1	410.4	352.1	0.026

Table 12: RRT 7 obstacles

#	Total CC time [μs]	Fastest Execution [μs]	Slowest Execution [μs]	Average Execution [μs]
1	339914	20	86630	1118
2	169169	20	1933	530
3	959105	21	5470	618
4	175121	22	3150	668
5	719079	18	3522	520
6	546323	21	4633	691
7	245861	18	2704	560
8	410194	20	5130	676
9	712983	22	3007	560
10	391232	22	2059	558
AVG	466898.1	20.4	11823.8	649.9

Table 13: RRT 7 obstacles

4.3.3 RRT Classic with Bounding box

#	Iter.	Found	Ex. time[ms]	n° to goal	TOT n°	n° collisions	Minimum clearance
1	777	Y	1949	23	342	435	0.166
2	13052	Y	40821	23	4399	8653	0.163
3	1090	Y	2568	23	443	647	0.147
4	4414	Y	8270	23	1180	3234	0.166
5	2757	Y	6211	23	968	1789	0.155
6	528	Y	614	24	120	408	0.148
7	1438	Y	3012	23	522	916	0.155
8	1499	Y	2879	23	499	1000	0.148
9	628	Y	1494	30	271	357	0.165
10	5184	Y	11671	22	1705	3479	0.148
AVG	3136.7	Y	7948.9	23.7	1044.9	2091.8	0.1561

Table 14: RRT BB 7 obstacles

#	Total CC time [μs]	Fastest Execution [μs]	Slowest Execution [μs]	Average Execution [μs]
1	1828055	20	9887	2352
2	23751061	21	10259	1819
3	2372487	19	7377	2176
4	6745639	22	10105	1528
5	5289188	22	10482	1918
6	571424	23	5895	1082
7	2737348	19	8934	1903
8	2589667	21	8761	1727
9	1408767	19	8809	2243
10	8921450	20	9103	1720
AVG	5621508.6	20.6	8961.2	1846.8

Table 15: Times RRT BB 7 obstacles

4.3.4 Comparison

Algorithm	Iter.	Found	Ex. T[ms]	n° to goal	TOT n°	n° collision	n° QP not feas.	Minimum clearance
RRT+CBF	481.0	100%	1397.2	22.9	290.9	—	129.4	0.1454
RRT	762.5	100%	611.6	23.1	410.4	352.1	—	0.0265
RRT SD	3136.7	100%	7948.9	23.7	1044.9	2091.8	—	0.1561

Table 16: Averages comparison 7 obstacles

Algorithm	Total time [μs]	Fastest Execution [μs]	Slowest Execution [μs]	Average Execution [μs]
QP	1290736.7	139.6	21314.2	2583.1
CC	466898.1	20.4	11823.8	649.9
CC SD	5621508.6	20.6	8961.2	1846.8

Table 17: Time averages comparison 7 obstacles

Again the **RRT-CBF** algorithm performs less iterations with respect to the **RRT** and has less nodes in the tree. However the number of discarded nodes due to unfeasibility of the **QP** problem increases, requiring an higher time interval to find a solution. The execution time of **RRT-CBF** is twice the execution time of **RRT**. Even in this case the **RRT** with bounding box has the worst performances and the number of collisions keep growing due to the crowded environment.

4.4 Simulation Environment - 11 Obstacles

The simulations was performed in the following scenario:

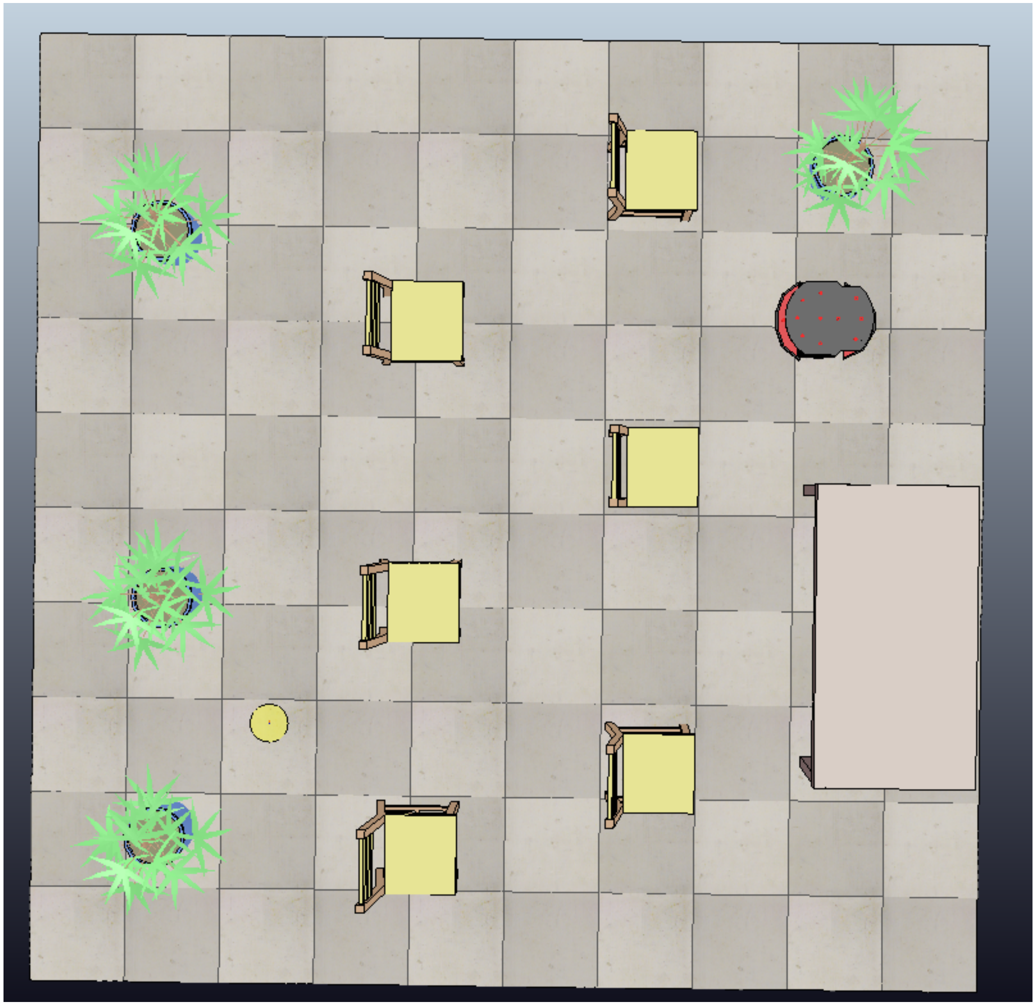


Figure 12: Simulations Scenario - 11 obstacles

The simulation parameters are not repeat since are the same of the previous cases.

4.4.1 RRT with CBF

#	Iter.	Found	Ex. time[ms]	n° to goal	TOT n°	n° QP is unfeasible	Minimum clearance [m]
1	27396	Y	110140	25	9425	14773	0.137
2	5618	Y	11673	27	2224	2701	0.149
3	1242	Y	2489	26	606	435	0.147
4	3631	Y	8034	21	1544	1486	0.151
5	854	Y	1589	22	434	330	0.136
6	6030	Y	12874	28	2535	2664	0.14
7	4077	Y	8865	27	1819	1692	0.125
8	19685	Y	62878	33	6857	10256	0.148
9	2053	Y	3786	24	857	967	0.136
10	6730	Y	16129	23	2792	2963	0.0504
AVG	7731.6	100%	23845.7	25.6	2909.3	3826.7	0.1320

Table 18: RRT-CBF 11 obstacles

#	Total QP time [μs]	Fastest Execution [μs]	Slowest Execution [μs]	Average Execution [μs]
1	43077005	65	6352	1572
2	8005121	109	6309	1424
3	2225614	150	11058	1791
4	6456236	95	7304	1778
5	1442560	132	55874	1689
6	8408858	116	6482	1394
7	6659410	113	8998	1633
8	25585796	66	6113	1299
9	3263411	68	4242	1589
10	10899582	67	8486	1619
AVG	11602359.3	98.1	12121.8	1578.8

Table 19: RRT-CBF 11 obstacles

4.4.2 RRT Classic

#	Iter.	Found	Ex. time[ms]	n° to goal	TOT n°	n° collisions	Minimum clearance
1	1537	Y	1790	17	845	692	0.0132
2	1290	Y	1201	31	616	674	0.0339
3	1933	Y	2038	19	905	1028	0.0132
4	1486	Y	1535	16	707	779	0.000162
5	248	Y	306	17	182	66	0.00896
6	391	Y	317	17	158	233	0.016
7	2478	Y	3069	18	1307	1171	0.0132
8	1398	Y	1504	16	715	683	0.0224
9	669	Y	670	16	356	313	0.0192
10	692	Y	761	16	382	310	0.00896
AVG	1212.2	100%	1319.1	18.3	617.3	594.9	0.0149

Table 20: RRT 11 obstacles

#	Total CC time [μs]	Fastest Execution [μs]	Slowest Execution [μs]	Average Execution [μs]
1	1358165	23	5947	883
2	932671	21	22938	723
3	1490992	21	4865	771
4	1162113	22	6125	782
5	282510	24	4851	1139
6	283589	22	4394	725
7	2106443	21	3738	850
8	1162245	22	7008	831
9	568263	25	2285	849
10	655205	24	5638	946
AVG	1000219.6	22.5	6778.9	849.9

Table 21: RRT 11 obstacles

4.4.3 RRT Classic with Bounding box

#	Iter.	Found	Ex. time[ms]	n° to goal	TOT n°	n° collisions	Minimum clearance
1	1680	Y	7998	23	589	1091	0.184
2	10726	Y	61894	19	3957	6769	0.153
3	13623	Y	71628	23	4353	9270	0.153
4	5256	Y	32120	17	2288	2968	0.167
5	1494	Y	7689	22	583	911	0.153
6	2406	Y	11231	23	849	1557	0.15
7	6592	Y	37632	19	2657	3935	0.153
8	18267	Y	116148	18	6668	11599	0.157
9	3249	Y	13389	22	985	2264	0.154
10	6561	Y	34018	16	2298	4263	0.158
AVG	6985.4	100%	39374.7	20.2	2522.7	4462.7	0.1582

Table 22: RRT with BB 11 obstacles

#	Total CC time [μs]	Fastest Execution [μs]	Slowest Execution [μs]	Average Execution [μs]
1	7598435	21	18670	4522
2	48622211	22	23802	4533
3	51992793	22	20966	3816
4	28044468	21	24034	5335
5	7348486	20	31066	4918
6	10421686	21	22876	4331
7	32063138	21	23071	4863
8	80454698	21	22727	4404
9	12232352	21	21183	3764
10	28978669	21	24130	4416
AVG	30775693.6	21.1	23252.5	4490.2

Table 23: Times RRT with BB 11 obstacles

4.4.4 Comparison

Algorithm	Iter.	Found	Ex. T[ms]	n° to goal	TOT n°	n° collision	n° QP not feas.	Minimum clearance
RRT+CBF	7731.6	100%	23845.7	25.6	2909.3	—	3826.7	0.1320
RRT	1212.2	100%	1319.1	18.3	617.3	594.9	—	0.0150
RRT SD	6985.4	100%	39374.7	20.2	2522.7	4462.7	—	0.1582

Table 24: Averages comparison 11 obstacles

Algorithm	Total time [μs]	Fastest Execution [μs]	Slowest Execution [μs]	Average Execution [μs]
QP	11602359.3	98.1	12121.8	1578.8
CC	1000219.6	22.5	6778.9	849.9
CC SD	30775693.6	21.1	23252.5	4490.2

Table 25: Time averages comparison 11 obstacles

In this environment the **RRT-CBF** performs worse than **RRT** in all the comparison terms. It must be noted that the execution time is highly increased due to the minimum clearance and the solver struggles in finding a solution for the **QP** problem. However when trying to maintain a minimum clearance with the classic **RRT** the performances worsen making the **RRT-CBF** the most suitable choice for a good trade off between safe distance and execution time.

4.5 Simulation Environment - 17 Obstacles

The simulations was performed in the following scenario:

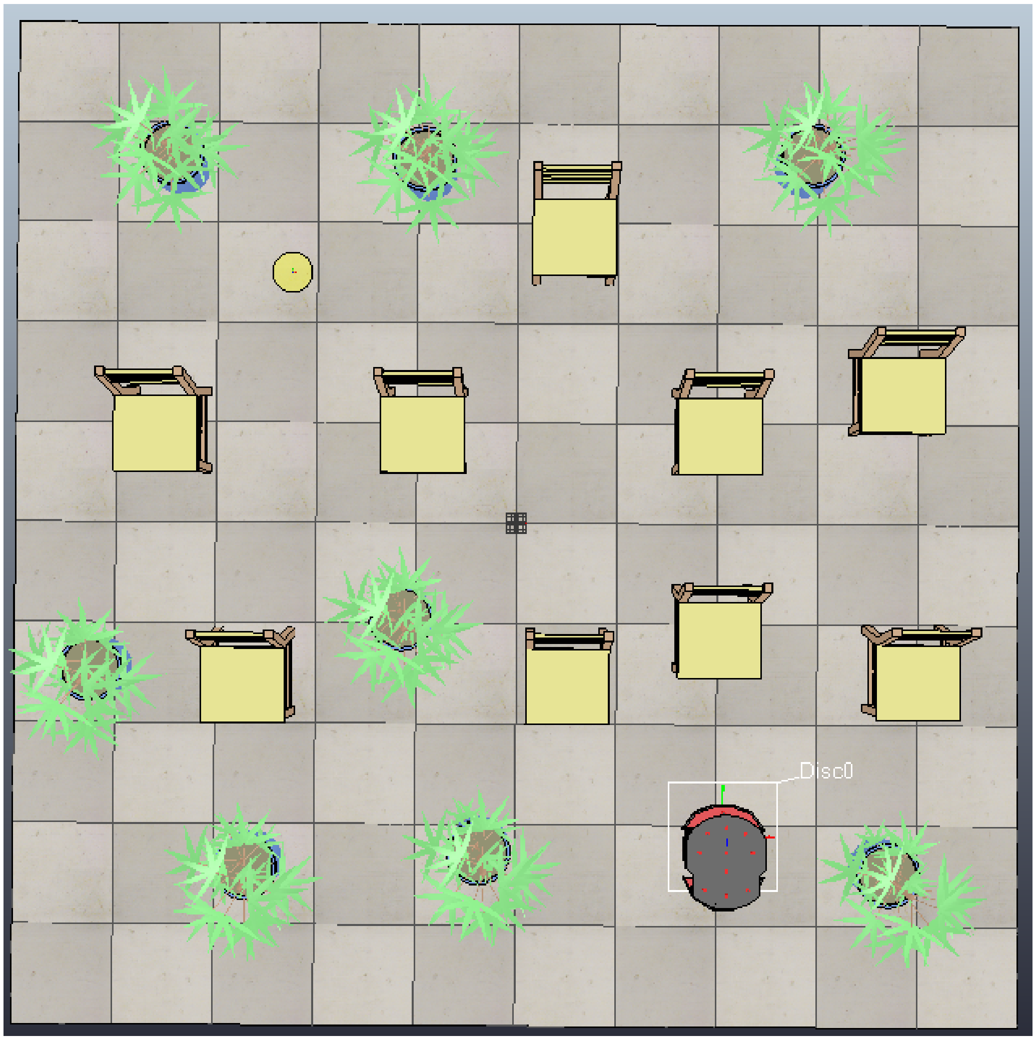


Figure 13: Simulations Scenario - 17 obstacles

4.5.1 RRT-CBF

#	Iter.	Found	Ex. time[ms]	n° to goal	TOT n°	n° QP is unfeasible	Minimum clearance
1	30000	N	96082	0	3360	26641	0
2	30000	N	97634	0	3231	26770	0
3	30000	N	95077	0	3360	26641	0
4	30000	N	97974	0	3390	26611	0
5	30000	N	93353	0	3152	26849	0
6	30000	N	94104	0	3269	26732	0
7	30000	N	100037	0	3342	26659	0
8	30000	N	99277	0	3388	26613	0
9	30000	N	97563	0	3250	26751	0
10	30000	N	95971	0	3451	26550	0
AVG	30000	0%	96707.2	0	3319.3	26681.7	0

Table 26: RRT-CBF 17 obstacles

#	Total QP time [μs]	Fastest Execution [μs]	Slowest Execution [μs]	Average Execution [μs]
1	59892821	73	8384	1996
2	61269575	75	8567	2042
3	59071174	74	8505	1969
4	60384474	71	9247	2012
5	58155961	70	8480	1938
6	58962847	71	82678	1965
7	63463027	72	8303	2115
8	61331832	71	8751	2044
9	61923043	80	9026	2064
10	58362003	71	7408	1945
AVG	60281675.7	72.8	15934.9	2009.0

Table 27: RRT-CBF 17 obstacles

4.5.2 RRT Classic

#	Iter.	Found	Ex. time[ms]	n° to goal	TOT n°	n° collisions	Minimum clearance
1	17845	Y	30244	31	3866	13979	0.00249
2	1525	Y	1208	25	366	1159	0.00675
3	14679	Y	24776	36	3127	11552	0.00344
4	6558	Y	6180	31	1152	5406	0.0042
5	11190	Y	19776	32	3238	7952	0.008
6	9183	Y	14106	31	2531	6652	0.00375
7	18317	Y	25595	36	3094	15223	0.00176
8	30000	N	85317	0	6255	23745	0.0
9	4222	Y	4923	31	1188	3034	0.0059
10	3443	Y	4715	31	1148	2295	0.00235
11	8616	Y	10072	32	1892	6724	0.00514
AVG	12557.8	Y	22691.2	31.6	2785.7	9772.1	0.004378

Table 28: RRT 17 obstacles

#	Total CC time [μs]	Fastest Execution [μs]	Slowest Execution [μs]	Average Execution [μs]
1	10219766	25	11331	572
2	937762	25	9030	614
3	9282245	25	13896	632
4	3495821	22	10307	533
5	8546698	22	16054	763
6	6297044	26	10785	685
7	9344953	25	10734	510
8	22235375	22	11524	741
9	3203913	26	12104	758
10	3153075	27	7718	915
11	5207461	22	56765	604
AVG	8192411.3	26.7	17024.8	732.7

Table 29: RRT 17 obstacles

4.5.3 RRT Classic with Bounding box

#	Iter.	Found	Ex. time[ms]	n° to goal	TOT n°	n° collisions	Minimum clearance
1	30000	N	99326	0	4279	25721	0
2	30000	N	90934	0	4283	25717	0
3	30000	N	208598	0	4189	25811	0
4	30000	N	90301	0	4255	25745	0
5	30000	N	86899	0	4121	25879	0
6	30000	N	84978	0	4169	25831	0
7	30000	N	216165	0	4280	25720	0
8	30000	N	96551	0	4219	25781	0
9	30000	N	84378	0	4286	25714	0
10	30000	N	243429	0	4231	25769	0
AVG	30000.0	0%	130155.9	0	4231.2	25768.8	0

Table 30: RRT with BB 17 obstacles

#	Total CC time [μs]	Faster Execution [μs]	Slower Execution [μs]	Average Execution [μs]
1	56714430	20	30832	1890
2	49408018	17	34528	1646
3	119621775	30	61145	3987
4	49635118	17	25932	1654
5	47721090	17	21361	1590
6	46946632	17	16596	1564
7	111912200	33	150873	3730
8	52005575	17	31636	1733
9	46367119	17	19928	1545
10	132815535	32	60439	4427
AVG	71314749.2	21.7	45327.0	2376.6

Table 31: Times RRT with BB 17 obstacles

4.5.4 Comparison

Algorithm	Iter.	Found	Ex. T[ms]	n° to goal	TOT n°	n° collision	n° QP not feas.	Minimum clearance
RRT+CBF	30000	0%	96707.2	0	3319.3	—	26681.7	0
RRT	12557.8	90%	22691.2	31.6	2785.7	9772.1	—	0.004378
RRT SD	30000	0%	130155.9	0	4231.2	25768.8	—	0

Table 32: Average values for 17 obstacles scenario

Algorithm	Total time [μs]	Fastest Execution [μs]	Slowest Execution [μs]	Average Execution [μs]
QP	60281675.7	72.8	15934.9	2009.0
CC	8192411.3	26.7	17024.8	732.7
CC SD	71314749.2	21.7	45327.0	2376.6

Table 33: Time Average values for 17 obstacles scenario

As it can be seen from the results in table (32) and (33) when the environment is more crowded, the **RRT-CBF** is no longer able to find a solution to the motion planning problem as well as the **RRT** with bounding box. While guaranteeing a minimum clearance from the obstacles it introduces the drawback of not finding the solution passing very close to $\mathscr{C}_{obs}$, as the classic **RRT** does.

5 Conclusion

RRT-CBF is roughly speaking "weighted by the distance from the obstacles", it has a feature which can be seen as good or bad depending on the interpretation. The main feature of **RRT-CBF** consists in the fact that the robot tends to follow safe paths which are more distant from the obstacles while, **RRT** performs only a collision checking. Therefore in **RRT** case no minimum clearance is present in the generated paths, that can be dangerously close to the obstacles. The main issue in **RRT-CBF** is related to narrow passages and crowded environments. In particular the presence of a minimum clearance makes the **QP** problem hard to solve as can be seen from the experimental results. However, it has been shown that the **RRT-CBF** is the best way to achieve a desired minimum clearance instead of using the classic **RRT** with bounding boxes. In fact the worsening in performances of **RRT** with minimum safe distance lighten the fact that **RRT-CBF** is more suitable for cases in which safety is prioritized.

For what concerns the time needed to find a solution in crowded environment, the **RRT** performs better than the **RRT-CBF** if no minimum clearance is required. Instead for applications that requires a mandatory minimum clearance the most suitable choice results to be the **RRT-CBF** algorithm.

The last important difference relies on the bigger freedom on the choice of motion primitives from the **RRT-CBF**. Thanks to the optimization problem, this algorithm is not indeed constrained to a finite number of motion primitive, which need to be set *a priori*, as the classic **RRT** version does. This fact theoretically should allow the robot to reach points in the workspace that are not feasible by moving only along the motion primitives. Anyway, the same problem could be solved with classic **RRT**, by decreasing the time interval.

The approximation of the unicycle to a circle could play a negative role on this and generate and excess of safety, where instead it is not strictly necessary.

As last, in addition to the tunable parameters of the **RRT**, the **RRT-CBF** requires also to adjust the values related to the optimization problem.

References

[1] Guang Yang, Bee Vang, Zachary Serlin, Calin Belta, Roberto Tron
Sampling-based Motion Planning via Control Barrier Functions
2019 Association for Computing Machinery.

[2] Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada
Control Barrier Functions: Theory and Applications
2019 18th European Control Conference (ECC)

[3] Quan Nguyen and Koushil Sreenath
Exponential Control Barrier Functions for Enforcing High Relative-Degree Safety-Critical Constraints
2016 American Control Conference (ACC)

[4] Hans Joachim Ferreau et al.
qpOASES Documentation, User's Manual, 2014

[5] B. Siciliano, L. Sciavicco, L.Villani, G.Oriolo
Robotics: Modelling, Planning and Control
2009 Springer