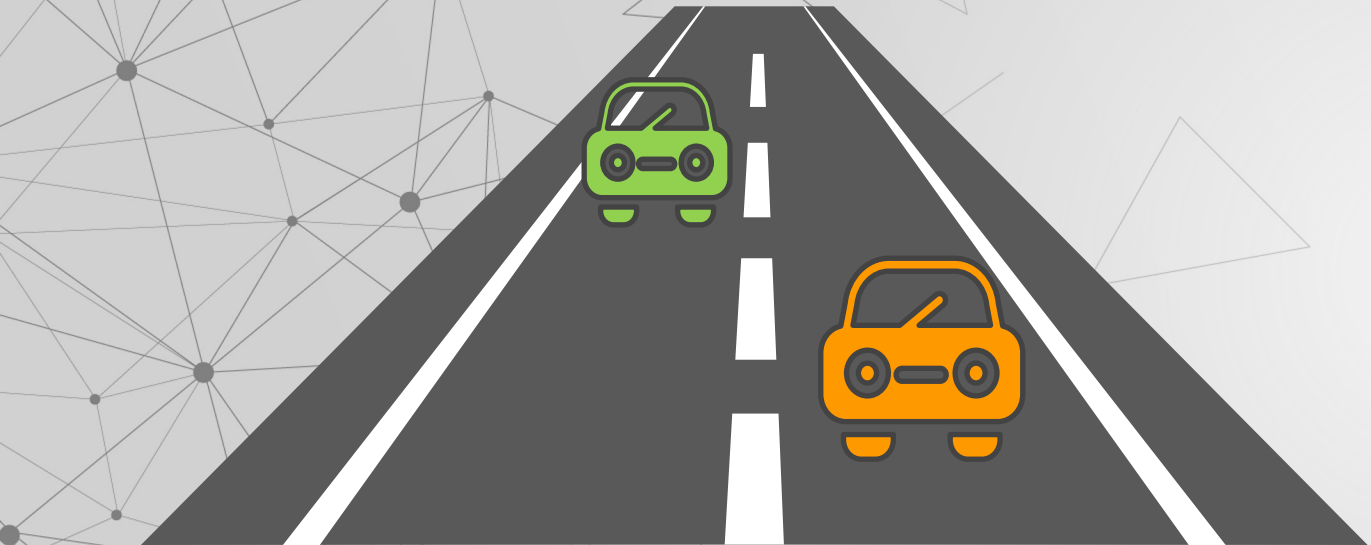
Autonomous and Mobile Robotics - Final project

A.Y. 2020/2021

Discrete-time Control Barrier Functions for safe vehicle control



Candidates

Rossella Bonuomo, 1923211

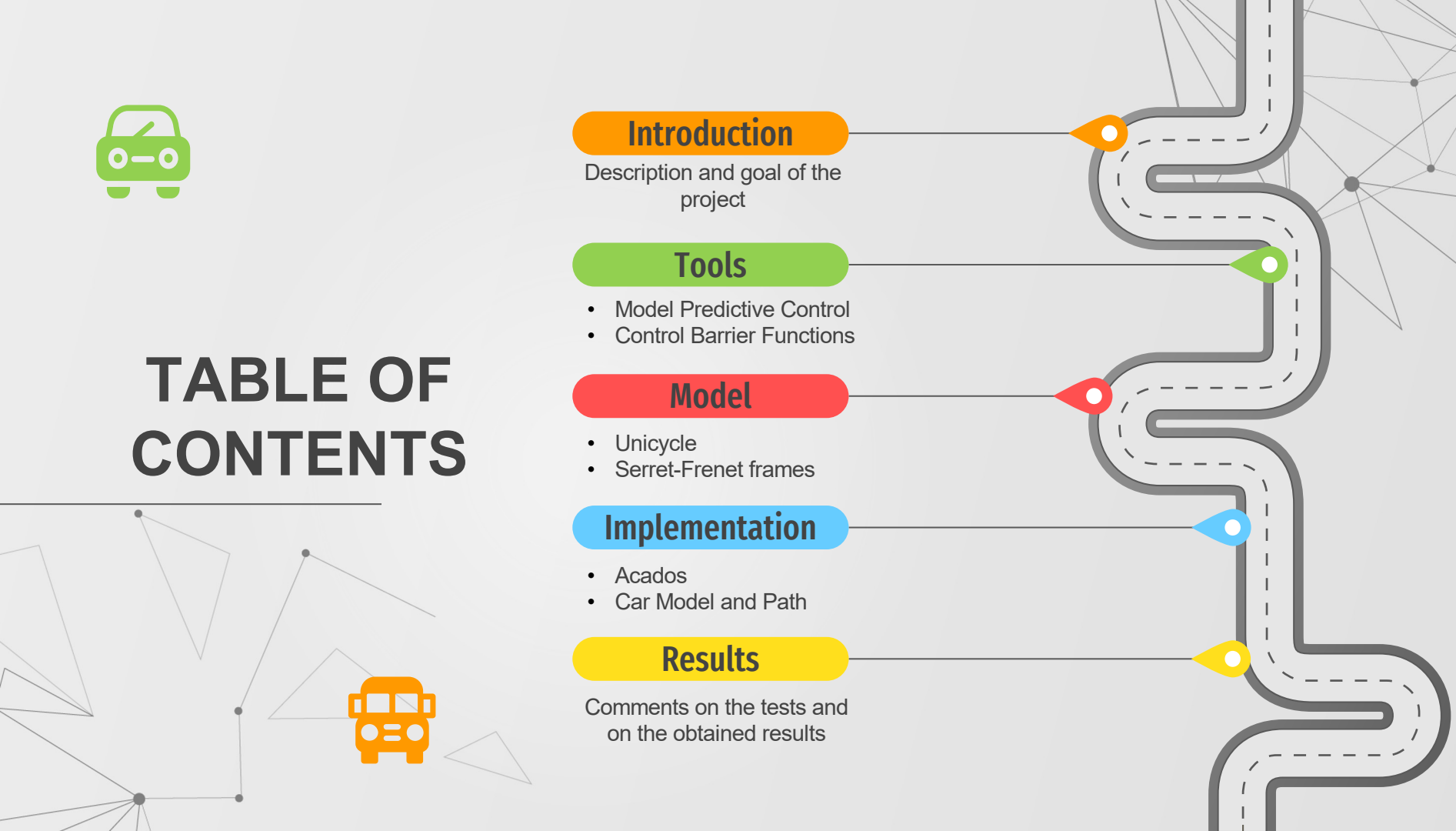
Marco Pennese, 1749223

Veronica Vulcano, 1760405

Professor Giuseppe Oriolo

Supervisor Tommaso Belvedere

TABLE OF CONTENTS



INTRODUCTION

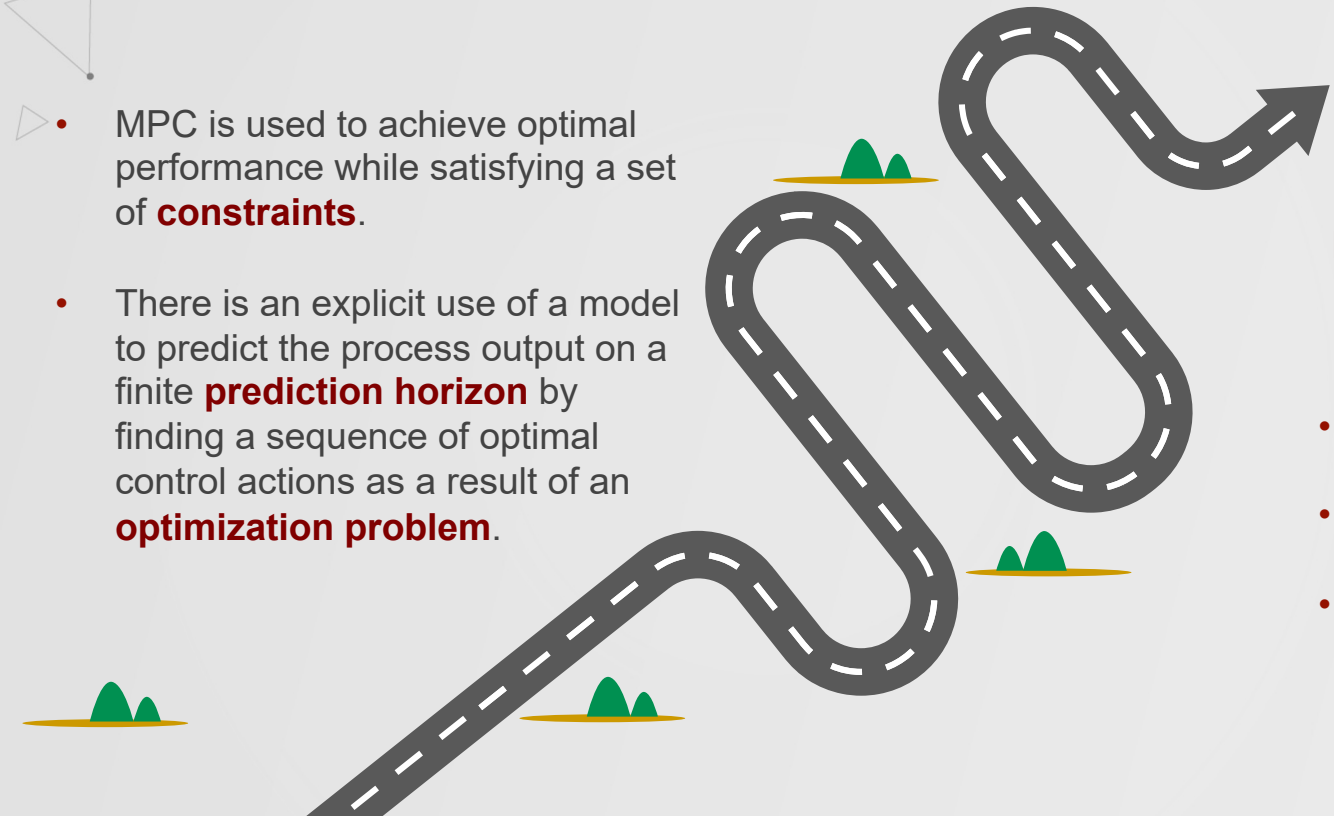
- Every engineered system should be designed to be **safe**, that means that "bad" things are avoided.
- Safety-critical optimal control is one of the most important problems in robotics.

The goal of the project is to apply **Nonlinear Model Predictive Control** with **Control Barrier Functions,** that generate safety constraints for obstacle avoidance, on the field of autonomous systems.

We are going to study the case of the **unicycle** by proposing **different scenarios**, making a comparison between **discrete time** and **continuous time**, and finally providing a **dynamic extension**.

TOOLS:
Model Predictive Control

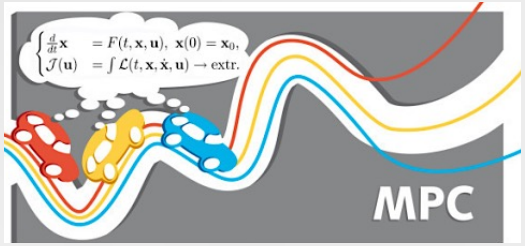
- MPC is used to achieve optimal performance while satisfying a set of **constraints**.
- There is an explicit use of a model to predict the process output on a finite **prediction horizon** by finding a sequence of optimal control actions as a result of an **optimization problem**.



We are going to consider **Nonlinear Model Predictive Control** since model and constraints are nonlinear.

- Finding the global optimum is more difficult.
- Convergence is **harder** to ensure.
- The algorithms are **time consuming** with respect to the linear case.

TOOLS:
Model Predictive Control



- At each time instant t, we consider a **prediction horizon** N and we predict the output of the process by using the process model.

Step 1

Step 2

- The control actions are computed solving an **optimization problem** which considers a cost function and some constraints.

- We apply just the first computed control action $u(t|t)$; the others are discarded (**receding horizon strategy**).
- We obtain the state and the output at the next time.

Step 3

Step 4

- The process is repeated.

TOOLS:
Control Barrier Functions

CBFs allow to express constraints which ***intrinsically*** enforce the system to be safe; these constraints can be used in the optimization problem.

Given a nonlinear control system $\dot{x} = f(x) + g(x)u$, then $h(x)$ becomes a **Control Barrier Function** if there exists an extended class $\mathcal{K}_\infty$ function γ such that for the control system $h(x)$ satisfies:

$$\sup_{u \in U} \left[L_f h(x) + L_g h(x) u \right] \geq \gamma\big(h(x)\big)$$

$$\text{with } L_f h(x) + L_g h(x) u = \dot{h}$$

In the CBF context, **safety** is defined in terms of a function $h(x)$, which should be greater or equal than 0.

CBF:
Continue vs Discrete Time

In order the car to be safe we need to satisfy $\boldsymbol{h(x) > 0}$ in each time instant. This results in a well-known **constraint** in the **MPC problem**.

Continuous Time

In continuous time, the CBF constraint can be expressed as:

$$\dot{h}(x_k, u_k) \geq -\gamma(h(x_k))$$

where $\boldsymbol{\gamma}$ is a **function** and $\boldsymbol{\gamma \in \mathcal{K}_\infty}$

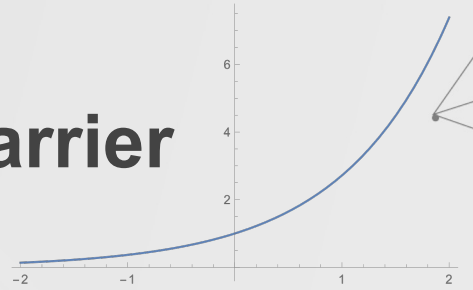
Discrete Time

In discrete time, the CBF constraint can be expressed as:

$$\Delta h(x_k, u_k) \geq -\gamma h(x_k)$$

where $\boldsymbol{\gamma}$ is a **scalar** and $\boldsymbol{0 < \gamma \leq 1}$

TOOLS: Exponential Control Barrier Functions



ECBFs are needed to deal with **high relative-degree** safety constraints

- We define $\eta_b = \begin{bmatrix} h(x) \\ \dot{h}(x) \\ \vdots \\ h^{r-1}(x) \end{bmatrix} = \begin{bmatrix} h(x) \\ L_f h(x) \\ \vdots \\ L_f^{r-1} h(x) \end{bmatrix}$
- We assume that we can write $L_f^r h(x) + L_g L_f^{r-1} h(x) u = \mu$

Then, the dynamics of $h(x)$ can be written as:

$$\dot{\eta}_b(x) = F\eta_b + G\mu$$
$$h(x) = C\eta_b(x)$$

By choosing a state feedback $\mu = -K_a \eta_b(x)$, then $h\big(x(t)\big) = Ce^{(F-GK_a)t}\eta_b(x_0)$

$h(x)$ becomes an **Exponential Control Barrier Function** if there exists a row vector $K_a \in \mathbb{R}^r$ such that:

$$\sup_{u \in U} \left[L_f^r h(x) + L_g L_f^{r-1} h(x) u \right] \geq -K_a \eta_b(x)$$

K_a can be defined by using pole placement strategies of feedback theory. In fact, the poles of the closed loop system $F - G\,K_a$ needs to be real and negative.

MPC–CBF

Let's see the formulation of the MPC problem for the discrete-time domain:

$$\min_{x,u} \sum_{k=0}^{N-1} x_k' Q x_k + u_k' R u_k + x_N' P x_N$$

$q(x_k, u_k)$ *stage cost*

$p(x_N)$ *terminal cost*

$$s.t.$$

$$x_0 = \bar{x}_0$$

initial condition

$$x_{min} \leq x_k \leq x_{max}$$
$$u_{min} \leq u_k \leq u_{max}$$

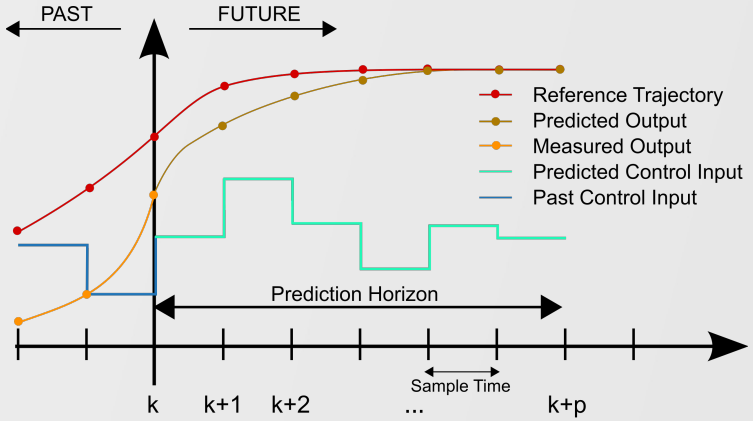
state and input constraints

$$x_{k+1} = F(x_k, u_k)$$

system's dynamics

$$\Delta h(x_k, u_k) \geq -\gamma h(x_k), \qquad 0 < \gamma \leq 1$$

CBF



MODEL: Unicycle

The model used is the kinematic model of the **unicycle:**

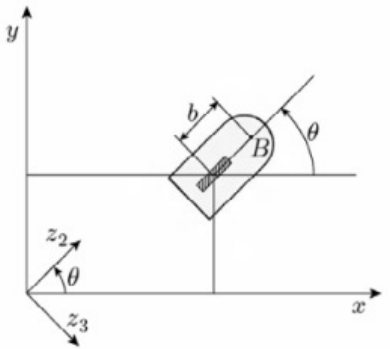
- it has a single orientable wheel
- the configuration is described by (x, y, θ)
- the inputs are v and ω

(x, y) are the Cartesian coordinates of the contact point of the wheel with the ground; θ is the orientation of the wheel with respect to the x axis.

v is the driving velocity,
ω is the steering velocity

Model expressed in Cartesian coordinates :

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega \end{cases}$$



Serret-Frenet Frame

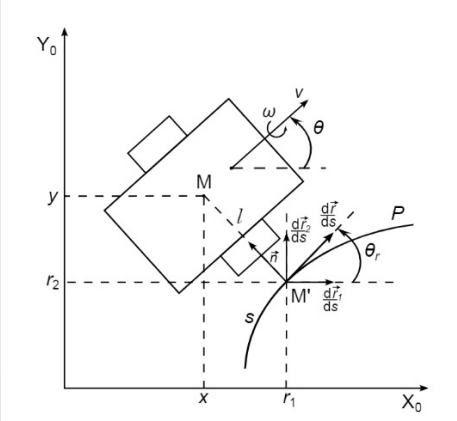
- We are going to consider the model of the unicycle expressed in the **Serret-Frenet frame**. It describes the kinematic of a point moving along a continuously, differentiable curve.

- To develop the model of the unicycle in the Serret-Frenet frame, we have to consider some additional quantities that can be represented as in the image.

$\boldsymbol{l}$ is the **lateral distance** from the path
$\boldsymbol{s}$ is the **curvilinear abscissa**
$\widetilde{\boldsymbol{\theta}} = \theta - \theta_r$ is the **orientation error**

Model expressed in **Serret-Frenet frame**:

$$\begin{cases} \dot{\boldsymbol{l}} = \boldsymbol{v}\sin\widetilde{\boldsymbol{\theta}} \\ \dot{\boldsymbol{s}} = \dfrac{\boldsymbol{v}\cos\widetilde{\boldsymbol{\theta}}}{\mathbf{1} \pm \boldsymbol{k}(\boldsymbol{s})\boldsymbol{l}} \\ \dot{\widetilde{\boldsymbol{\theta}}} = \boldsymbol{\omega} \pm \dfrac{\boldsymbol{k}(\boldsymbol{s})\boldsymbol{v}\cos\widetilde{\boldsymbol{\theta}}}{\mathbf{1} \pm \boldsymbol{k}(\boldsymbol{s})\boldsymbol{l}} \end{cases}$$



IMPLEMENTATION: acados

Acados is a software package for finding the solution of optimal control and estimation problems.

- It provides some libraries written in **C** which expose very simple interface to Python
- It is compatible with the language of **CasADi** (based on graphs)
- It uses **BLASFEO** for linear algebra operations to speed up the computation

It solves a **nonlinear Optimal Control Problem** whose aim is to minimize a cost function given the system dynamics, the initial conditions and the nonlinear path constraints.

$$\begin{aligned} w^{[i+1]} &= w^{[i]} + \Delta w^{[QP]} \\ \pi^{[i+1]} &= \pi^{[QP]} \\ \mu^{[i+1]} &= \mu^{[QP]} \end{aligned} \quad , \quad i = 0,1, \dots$$

There exist different solution strategies for QP interfaced from acados. We mainly focus on:

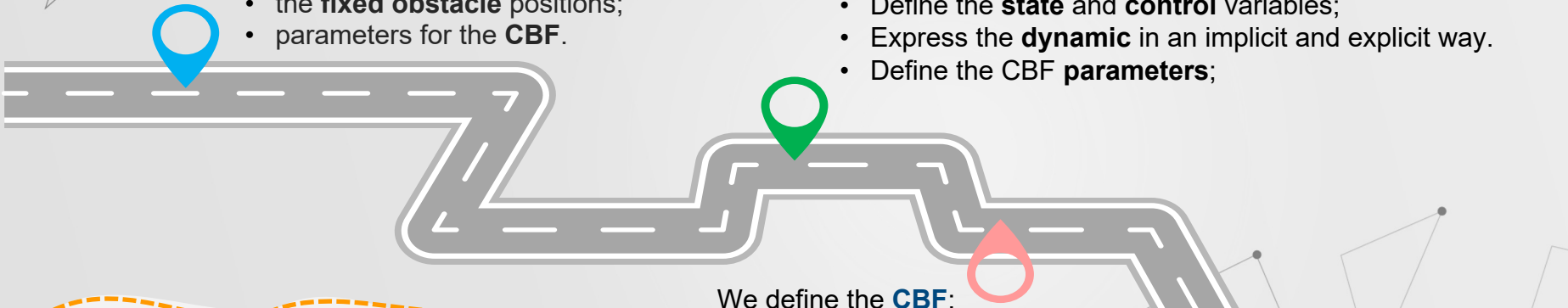
qpOASES	*HPIPM*
Active-set method	Interior-point method

Car Model

The car is represented by a **Python class**

It takes in input:
- the **path**;
- the **dimensions of the car**;
- the **fixed obstacle** positions;
- parameters for the **CBF**.

- Choose the **model** name;
- Define the **state** and **control** variables;
- Express the **dynamic** in an implicit and explicit way.
- Define the CBF **parameters**;



(s_c, l_c) represents the position of the car, (s_{obs}, l_{obs}) represents the position of the obstacle

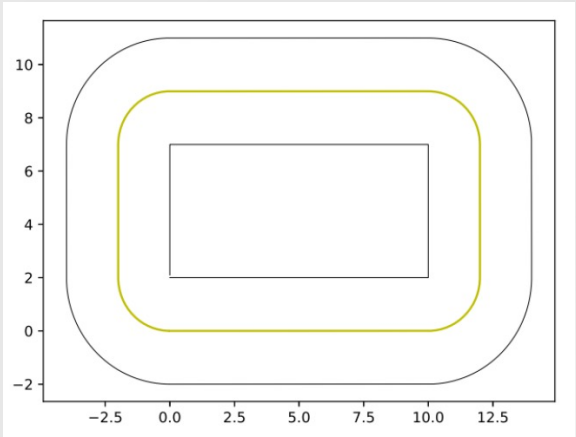
We define the **CBF**:

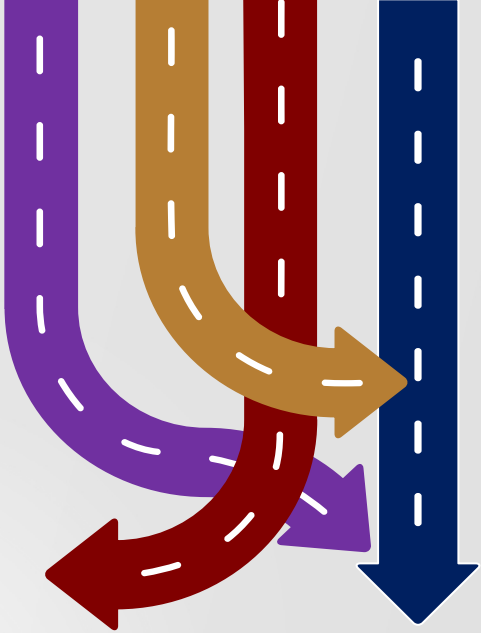
$$h_t^i = \frac{(s_c - s_{obs}^i)^4}{(2l_1)^4} + \frac{(l_c - l_{obs}^i)^4}{(2l_2)^4} - \alpha$$

We need to compute $\Delta h(x_k, u_k) \coloneqq h(x_{k+1}) - h(x_k)$, so we need to **integrate** the state. At first, we tried a simple **Euler integration** but due to inaccuracy errors, we moved to the **Runge-Kutta integrator**.

Path

The car has to move along the following path:

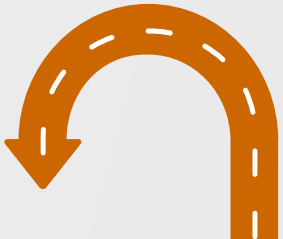




It can be created specifying:

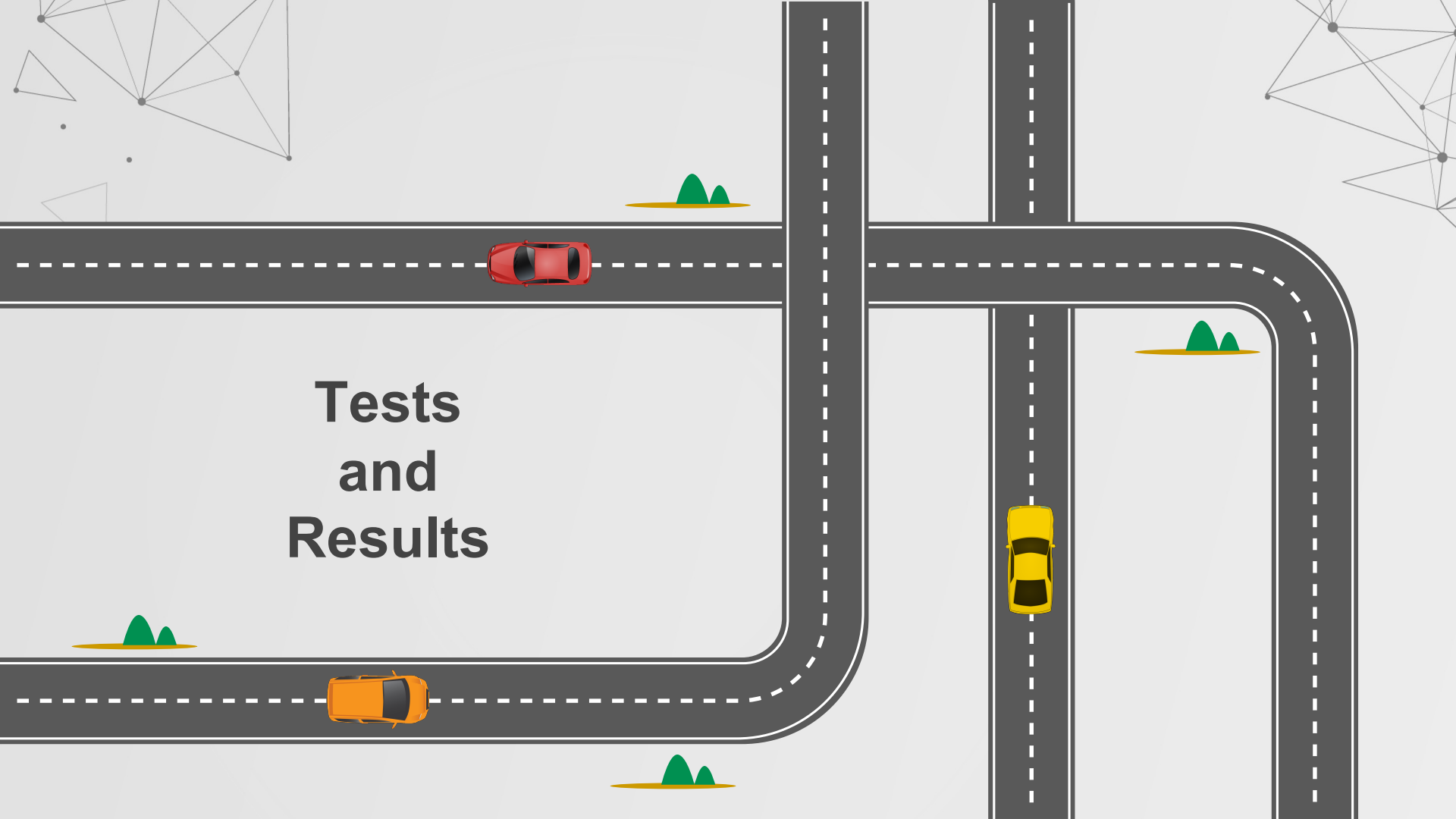
- the **vertical and horizontal length;**
- the **radius of the circumferences**.

We let the path provide us some information as a function of the **arc length** s, and these information are useful in the kinematic model:



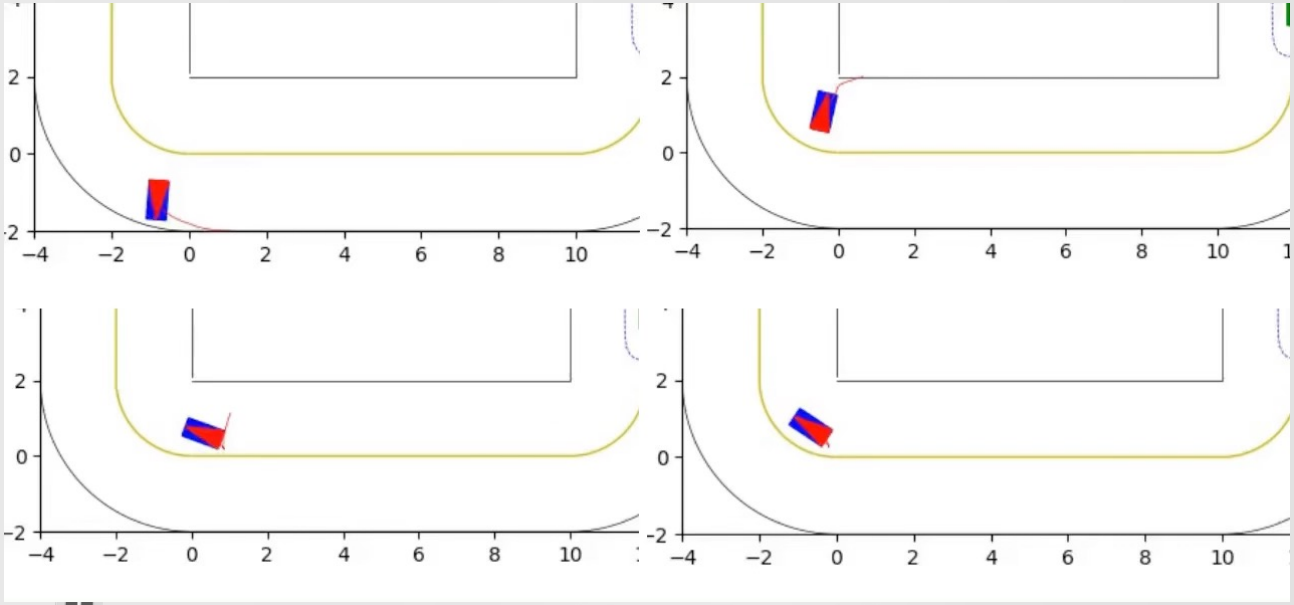
- k: is the **curvature of the path** in a specific point. It can be computed as the inversion of the radius of the circle tangent to the path at a point characterized by s
- θ_r: is the **path orientation** at a point characterized by s

We also equipped the class with a method that returns the total **length** of the path.


Tests
and
Results

Convergence to trajectory

The aim of these tests is to check that the system is **asymptotically stable**; we have to be sure that even when the car is far from the reference trajectory, it is able to approach it.



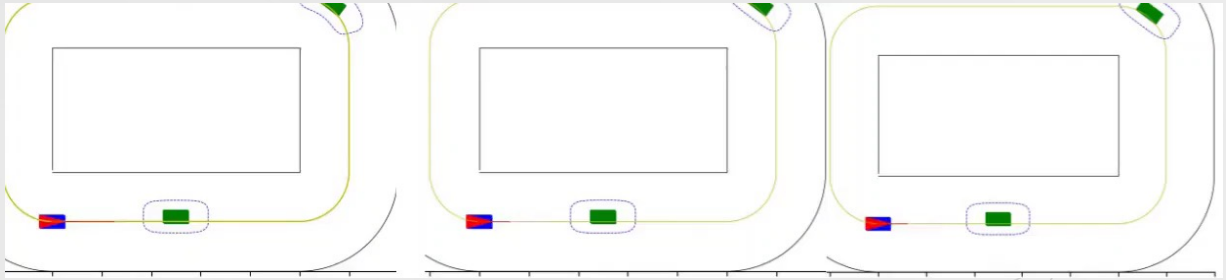
4 tests were performed with 4 different initial configurations.

In all the situations, after some maneuvers, the car is able to approach the path correctly.

Single car with fixed obstacles

Comparison on γ → it influences the way in which the car overcomes the obstacle. The larger is γ, the more the car begins to overtake the obstacle when it is very close to it.

In the last case, the car starts to *steer later*, so it has to rotate more to overcome the obstacle.



$\gamma = 0.1$ $\gamma = 0.5$ $\gamma = 1.0$

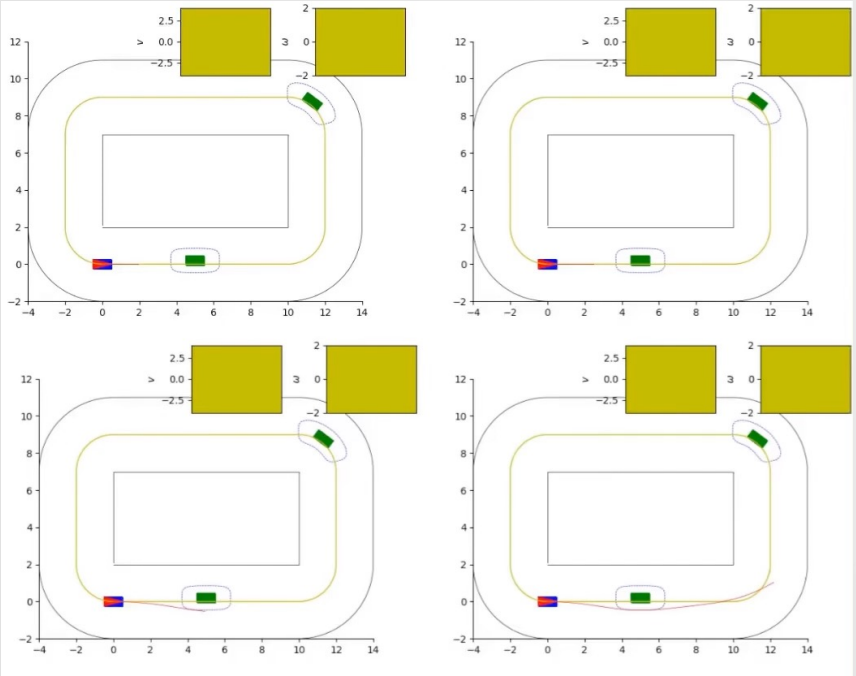
Horizon	γ	Mean time	Std dev	Max time	Cost integral
1 s	0.1	0.0296	0.0476	0.7414	134.93
1 s	0.5	0.0244	0.0315	0.4348	135.88
1 s	1	0.0261	0.0680	1.6073	136.00

There is a correlation between γ and the control effort but it does not influence the mean time.

Single car with fixed obstacles (cont.)

Comparison on different prediction horizons → It describes how much we are able to predict in the future.

$T_f = 0.8\ s$ $T_f = 1\ s$



$T_f = 2\ s$ $T_f = 5\ s$

If the prediction horizon is longer, the car is able to predict a better trajectory.

γ	Horizon	Mean time	Std dev	Max time	Cost integral
0.5	0.8 s	0.0128	0.0073	0.0781	×
0.5	1 s	0.0244	0.0315	0.4348	135.88
0.5	2 s	0.1265	0.1979	1.7619	134.03
0.5	5 s	2.2235	2.7018	15.2301	135.26

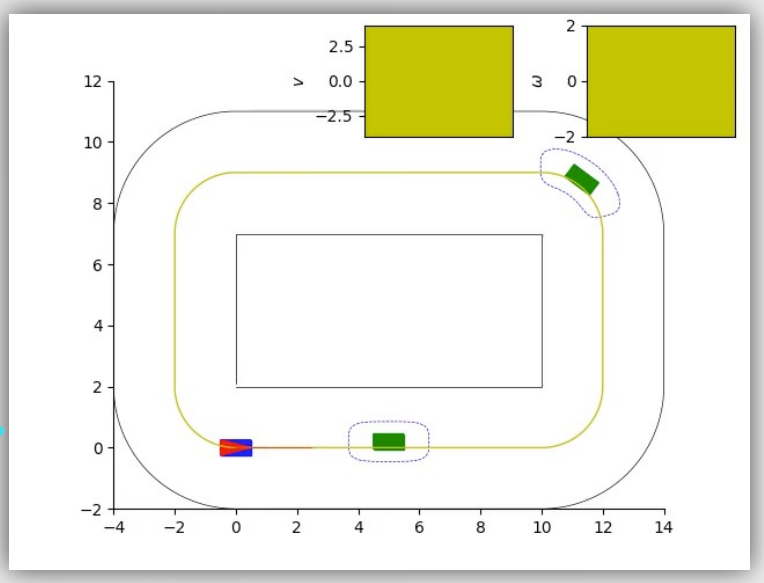
By increasing the prediction horizon, the average iteration time increases more than linearly.

Single car with fixed obstacles (cont.)

We would like to choose both γ and the prediction horizon so to obtain a ***desired behaviour***, while keeping the computational time small.

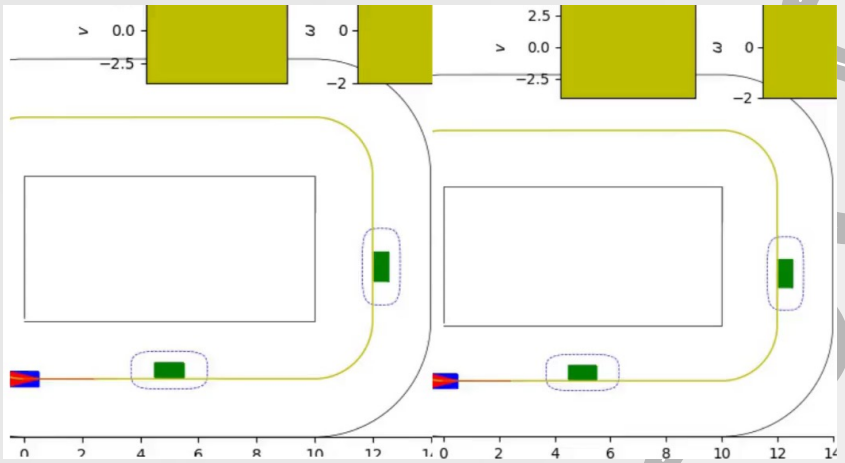
- **Reduce the prediction horizon** in order to have a shorter computational time
- **Compensate** for the loss of optimality brought by this **with the reduction of γ**.

γ	Horizon	Mean time	Std dev	Max time	Cost integral
0.05	0.8 s	0.0244	0.0455	0.6829	124.36
0.1	**1 s**	**0.0296**	**0.0476**	**0.7414**	**134.93**
0.5	1.5 s	0.0636	0.0998	0.9168	134.06
1	2 s	0.1184	0.1778	1.8536	134.05



Single car with moving obstacles

- The obstacles move with a *constant velocity of* $1\ m/s$;
- The car moves with a target velocity of $2.5\ m/s$; therefore, the car will overcome the obstacles.
- We consider both discrete time and continuous time by making a comparison on the **computational time**.



Continuous time **Discrete time**

	γ	Horizon	Mean time	Std dev	Max time	Cost integral
Discrete	0.03	1 s	0.0389	0.0554	0.7089	129.0932
Continuous	$1.5 \cdot h(x)$	1 s	0.0391	0.0605	0.7497	128.9603

Multiple cars

In order to make things more challenging, we tried to put together multiple controlled cars moving on the same circuit among obstacles.

We have considered two different approaches:

Centralized approach	Independent approach
All the cars are in the **same model**. We have a single optimization problem in which each car is aware of the others' behaviour.	Each car has its **own model**. We have as many optimization problems as the number of controlled cars.

The number of *state variables* is $n \times m$

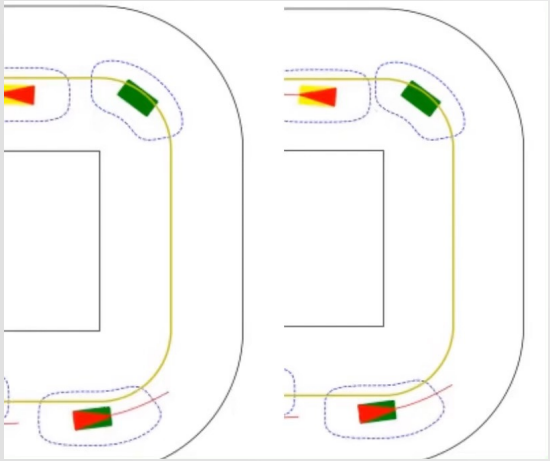
- n is the number of generalized coordinates
- m is the number of cars

The centralized approach is just an **ideal behavior**; the independent approach is the one that we can find in real applications.

Multiple cars

Differences appear when the cars approach each other.

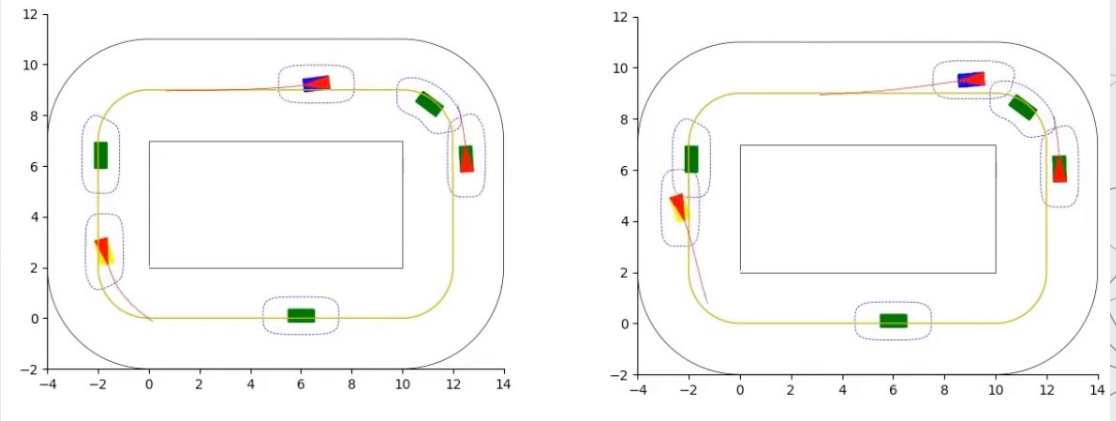
Let's see an example of **overtaking**:



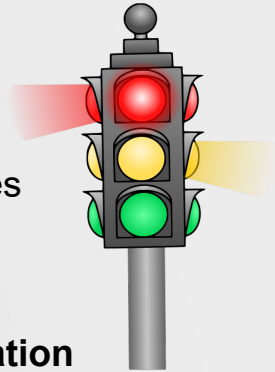
Centralized approach

Independent approach

If we look at the **final frame**, the cars in the centralized approach (on the left) are moved a bit forward with respect to the ones in the independent approach (on the right):



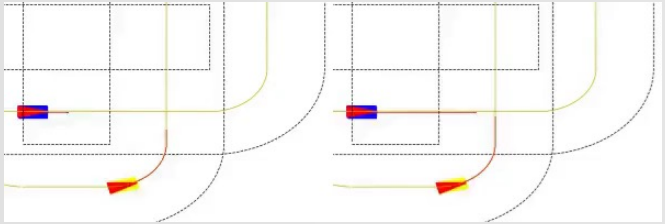
Multiple cars with intersection



Also in this case we consider the **independent** and **centralized** approaches

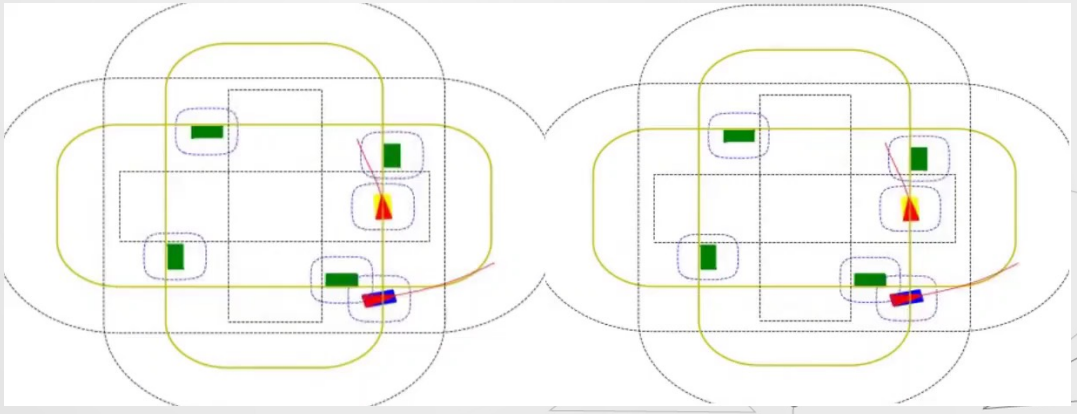
- Each car is just **aware** of other car's velocity and position but does not know which future actions it will do.
- We expect that they are more **prudent** at the intersection (i.e. they slow down or change the trajectory).

Independent Centralized



We expect an **optimal negotiation** when the cars intersect each other.

Independent Centralized



- When the cars are far from each other, they behave in the same way.
- When the cars meet, differences appear.

Dynamic extension

The control inputs we are going to consider are the **linear acceleration** and **angular acceleration**, so we have to add two integrators in the model:

Continuous time

We have a **relative degree** $r = 2$, therefore we need to design an ECBF. In our case, the ECBF is:

$$\ddot{h}(x) \geq -[K_1 \quad K_2] \begin{bmatrix} h(x) \\ \dot{h}(x) \end{bmatrix}$$

We are going to change the value of $K_a = [K_1 \quad K_2]$ in order to obtain the desired behaviour.

K_a influences the position of the **closed loop poles**, therefore it will influence the response of the systems at the given inputs.

$$\begin{cases} \dot{l} = v\sin\tilde{\theta} \\ \dot{s} = \dfrac{v\cos\tilde{\theta}}{1 \pm k(s)l} \\ \dot{\tilde{\theta}} = \omega \pm \dfrac{k(s)v\cos\tilde{\theta}}{1 \pm k(s)l} \\ \dot{v} = a \\ \dot{\omega} = a_\omega \end{cases}$$

Now the state variables are:

$$q = [l, s, \tilde{\theta}, v, \omega]$$

The output variables are:

$$y = [l, s, \tilde{\theta}]$$

The input variables are

$$u = [a, a_\omega]$$

Discrete time

- We don't need to design an ECBF because the relative degree of the **discretized** equivalent model is **always equal to 1**.
- In the experiments, we are still going to **tune the parameter γ** in the CBF in order to obtain the desired behaviour.

Computing derivatives for Continuous CBF

As we have seen, we need to compute derivatives in order to apply the CBF in the **continuous** case.

Since the relative degree is 2, we need up to the second derivative.

$$h(x) = \frac{\left(s_c - s_{obs}^i\right)^4}{(2l_1)^4} + \frac{\left(l_c - l_{obs}^i\right)^4}{(2l_2)^4} - \alpha$$

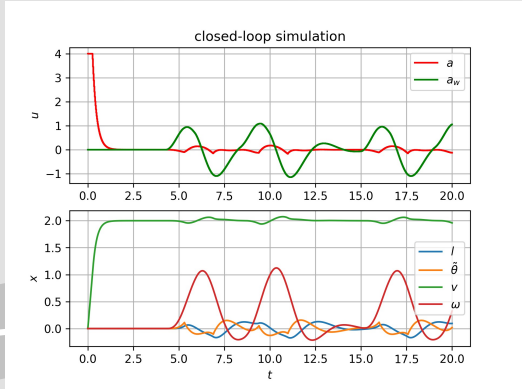
$$\dot{h}(x) = \frac{4\left(\dot{s}_c - \dot{s}_{obs}^i\right)\left(s_c - s_{obs}^i\right)^3}{(2l_1)^4} + \frac{4\left(\dot{l}_c - \dot{l}_{obs}^i\right)\left(l_c - l_{obs}^i\right)^3}{(2l_2)^4}$$

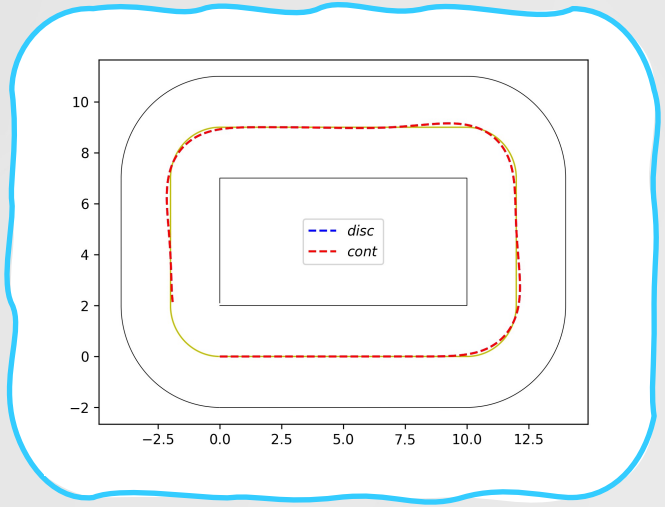
$$\ddot{h}(x) = \frac{12\left(\dot{s}_c - \dot{s}_{obs}^i\right)^2\left(s_c - s_{obs}^i\right)^2 + 4\ddot{s}_c\left(s_c - s_{obs}^i\right)^3}{(2l_1)^4} + \frac{12\left(\dot{l}_c - \dot{l}_{obs}^i\right)^2\left(l_c - l_{obs}^i\right)^2 + 4\ddot{l}_c\left(l_c - l_{obs}^i\right)^3}{(2l_2)^4}$$

Dynamic extension

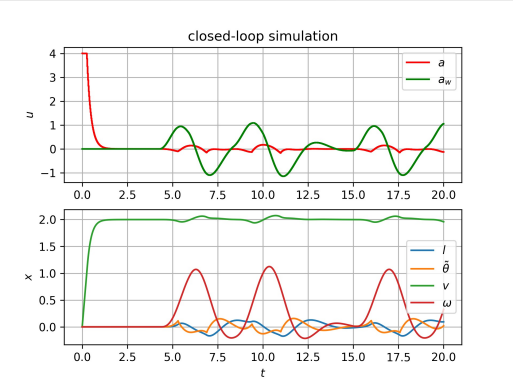
We run experiments without obstacle to compare the computational time.

Discrete time





Continuous time

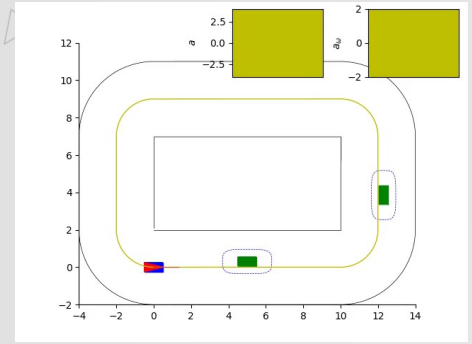


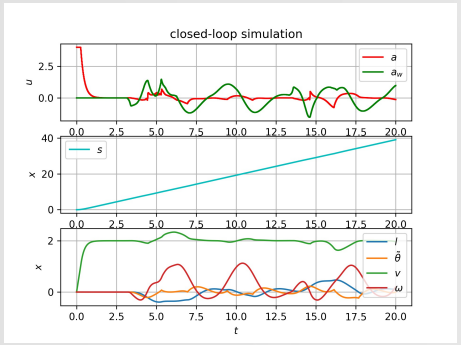
	Mean time	Std dev	Max time	Cost integral
Discrete	0.0012	0.0002	0.0034	12.7786
Continuous	0.0012	0.0002	0.0034	12.7786

Dynamic extension – simulations

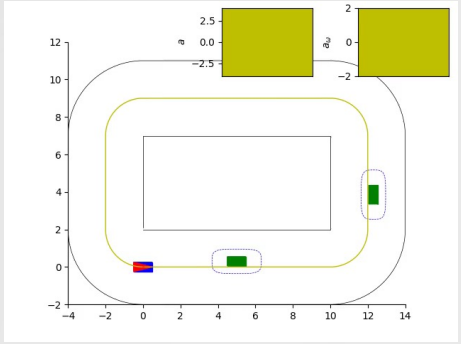
Continuous time: we are going to change K_a so to study the behaviour of the system
Discrete time: we are going to still change γ to study the behaviour of the systems

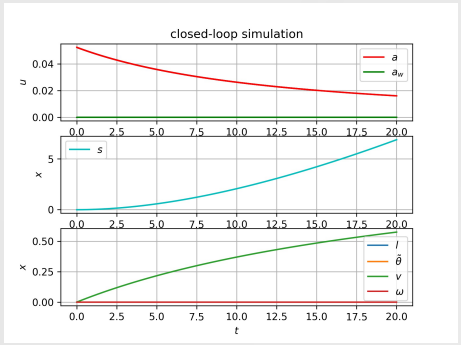
$K_a = [225 \quad 30]$ ➡ the closed loop poles are at -15 along the real axis.



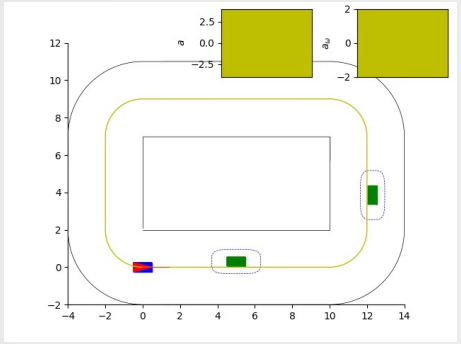


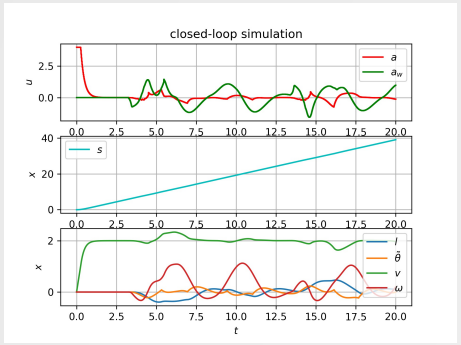
$K_a = [0.01 \quad 0.2]$ ➡ the closed loop poles are at -0.1 along the real axis.



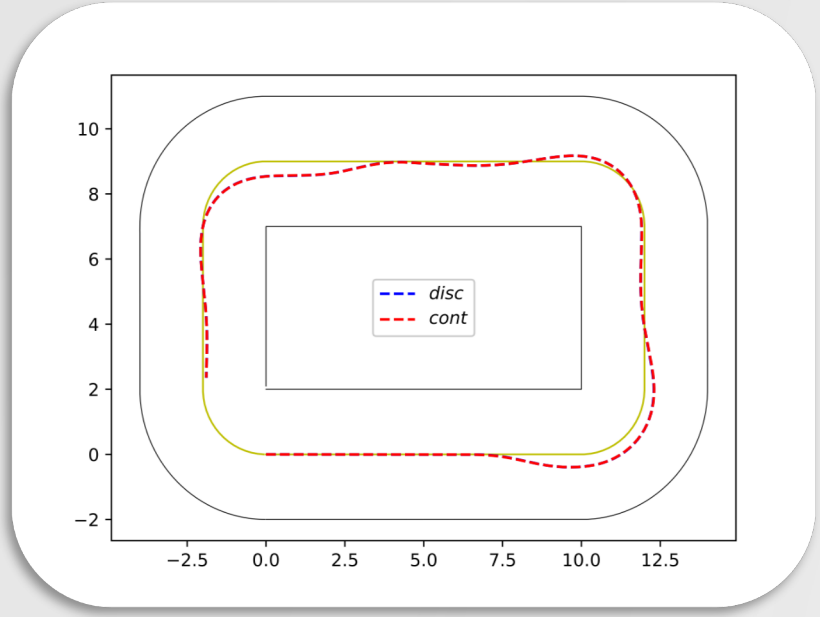


$\gamma = 0.3$



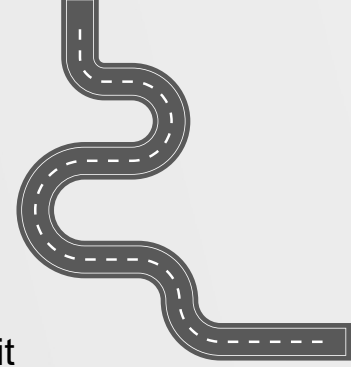


Dynamic extension – simulations (cont.)



	Mean time	Std dev	Max time	Cost integral
Discrete time	0.0029	0.0012	0.0101	15.3281
Continuous time	0.0025	0.0013	0.0101	15.0242

Conclusions



Control Barrier Function constraints can be used within the Model Predictive Control framework to ensure safety.

- CBF constraints allow the system to **avoid obstacles** even when it is **far** from them.
- It is possible to design a **well suited CBF** for the specific problem.
- As the problem complexity increases, so does the computational time, but we can mitigate this issue by **tuning** γ and T_f accordingly.

Discrete vs Continuous: There is not an evident advantage in performances in using one instead of the other. One could prefer to use the discrete one for its easier formulation.

- We have used Real-time iteration (RTI) so to speed up things.
- The obtained results are good; in most of the simulations, the car always safely races and overtakes the obstacles, keeping the desired path and the desired target velocity even in the more challenging scenarios.

THANK YOU
FOR LISTENING!