# Lab 6 – Using mixed integer programming for obstacle avoidance

**REPLAN team**

Tuesday 1st April, 2025

## Contents



compiled at: 01/04/2025, 07:34

We recapitulate the notion of polyhedral obstacle and add in the description binary variables to characterize an exclusion condition of the form $x \notin P$.
We integrate such a condition into a problem of generating the " on-the-go " trajectory, as a result of repeatedly solving a constrained optimization problem.

# 1 Theoretical background

To generate a " on-the-fly " trajectory in a complex environment (with fixed or mobile obstacles) we need:

- an effective way to describe a region of interest (be it an obstacle or a safety zone);
- a mechanism to return control to the system in such a way as to meet the requirements of the mission (obstacle avoidance, minimization of trajectory).

A common description is that of the polyhedral region:

$$P = \{x \in \mathbb{R}^n : \ F_i^\top x \leq \theta_i, \forall i = 1 : N_h\}, \tag{1}$$

where the space covered by the obstacle is given as an intersection of inequalities (remember the definitions in Lab 1). The definition in (1) defines all those points that respect *inclusion* $x \in P$ while we are interested in *exclusion* $x \notin P$. To model such a relationship, we enter binary variables $z_i \in \{0, 1\}$, which allows us to write a " big-M " form:

$$x \notin P \Longleftrightarrow \begin{cases} -F_i^\top x & \leq -\theta_i + M(1 - z_i), \ \forall i = 1 : N_h \\ \sum_{i=1}^{N_h} z_i & = 1. \end{cases} \tag{2}$$

Relation (2) may be interpreted as follows:

- one binary variable, the one with index $i$ is equal to '1 ', all the others ($\forall j = 1 : N_h$ for $j \neq i$) are zero;
- therefore for all indices $j \neq i$ we have that the inequalities involving $x$ are reduced to $-F_j^\top x \leq -\theta_j + M \cdot 1$, which means that for values sufficient big $M$, can be ignored;
- we are left with only the inequality $-F_i^\top x \leq -\theta_i + M \cdot 0$ which assures us that we are outside the $P$ obstacle.

Equations (2) combine real ($x$) and binary ($z_i$) variables, so we call the optimization problem in which they are integrated *mixed variable optimization problem.*

Such a problem, although relatively easy to formulate, is difficult (NP-hard, without guarantees of global optimum) because the number of cases increases exponentially (in the example in the relation (2) there are $2^{N_h}$ possibilities to write inequalities).

One way (described in Lab 2) to plan the movement is to pre-calculate a trajectory and then follow it during the simulation / experiment. The other usual approach is to calculate the "on the go" trajectory, solving an optimization problem at every step.

A very popular method is " Predictive Control Model - MPC " where a typical formulation is given by the optimization problem:

$$\min_{u_k \dots u_{k+N-1}} \sum_{i=1}^{N} \left(x_{k+i} - \bar{x}\right)^\top \left(x_{k+i} - \bar{x}\right) + \sum_{i=0}^{N-1} u_{k+i}^\top u_{k+i} \tag{3a}$$

$$\text{s.t.} \quad x_{k+i+1} = Ax_{k+i} + Bu_{k+i}, \tag{3b}$$

$$|u_{k+i}| \leq \bar{u}, \tag{3c}$$

$$|x_{k+i+1}| \leq \bar{x}, \tag{3d}$$

$$x_{k+i+1} \notin P, \qquad \forall i = 1 : N. \tag{3e}$$

Without going into details, such a method is preferred due to its versatility (we can change / add as we wish elements within (**??**)) and due to the fact that it explicitly takes into account constraints:

- costs to be minimized are defined (effort along the path): $u_{k+i}^\top u_{k+i}$, distance to target $\left(x_{k+i} - \bar{x}\right)^\top \left(x_{k+i} - \bar{x}\right)$, etc.);
- cosntraints have to be validated (on input: $|u_{k+i}| \leq \bar{u}$, on state: $|x_{k+i+1}| \leq \bar{x}$, for obstacle avoidance: $x_{k+i+1} \notin P$);
- the constraints and cost apply to a finite prediction horizon (of length $N$) and from the sequence of commands obtained $\{u_k, \dots, u_{n+N-1}\}$ only the first applies, $u_k$;
- repeat previous steps increasing the index $k \mapsto k+1$.

The $N$ parameter deserves a few additional words. The temptation is, of course, to choose a long prediction horizon in order to "see" the behavior of the system as far as possible in the future. The difficulty is that the problem (3) must be solved at every step $k$. At some point (depending on the nature and size of the problem) the computation time becomes significant (we can't lose seconds for a computation if a new command has to be sent every 50 or 100ms, as is the case with a drone).

In particular, if we add pseudo-linear constraints like the one in (2), the problem becomes significantly more difficult to solve ((3) is actually a MIQP-MPC). In general, the more complex a model is and the more difficult constraints it considers, the more difficult a optimization problem is.

Tools Yalmip (available only in Matlab / Octave) or CasADi (available in Matlab / Octave / Python / C ++) allow efficient implementations that ultimately call specialized solvers such as CPLEX, GUROBI, MOSEK or IPOPT.

## 2 Obstacle avoidance through mixed integer programming and MPC

As a first step, we define the equations that require avoiding a square:

$$\begin{cases} x_1 & \leq 5 + M(1 - z_1), \\ -x_1 & \leq 5 + M(1 - z_2), \\ x_2 & \leq 5 + M(1 - z_3), \\ -x_2 & \leq 5 + M(1 - z_4), \\ 1 & = z_1 + z_2 + z_3 + z_4. \end{cases} \tag{4}$$

We consider a simplified model, the "double integrator", discretized with step 0.1,

$$\begin{cases} x_1^+ & = x_1 + 0.1x_3, \\ x_2^+ & = x_2 + 0.1x_4, \\ x_3^+ & = x_3 + 0.1u_1, \\ x_4^+ & = x_4 + 0.1u_2, \end{cases} \tag{5}$$

where $(x_1, x_2)$ define the position, $(x_3, x_4)$ the velocity and $(u_1, u_2)$ the accelerations of the system. Not in the least, we impose constraints on the accelerations

$$-1 \leq u_1 \leq 1, \qquad -1 \leq u_2 \leq 1. \tag{6}$$

Integrating these elements as in (**??**) leads to a constrained optimization problem as in the following code snippet:

```
objective = 0
solver.subject_to(X[:, 0] == X_init)

for k in range(controller['Npred']):

    # @ does matrix multiplication in Casadi
    solver.subject_to(X[0, k + 1] == (X[0, k] + plant['Te']*X[2,k]))
    solver.subject_to(X[1, k + 1] == (X[1, k] + plant['Te']*X[3,k]))
    solver.subject_to(X[2, k + 1] == (X[2, k] + plant['Te']*U[0,k]))
    solver.subject_to(X[3, k + 1] == (X[3, k] + plant['Te']*U[1,k]))

    solver.subject_to(plant['umin'] <= U[:, k])
    solver.subject_to(U[:, k] <= plant['umax'])

    objective = objective + \
                casadi.transpose(X[:, k] - X_target) @ (X[:, k] -
    X_target) + \
                casadi.transpose(U[:, k]) @ (U[:, k])

solver.minimize(objective)
```

Once the 'solver' object is obtained, it is called repeatedly, changing only the initial values, in a simulation loop:

```
solver.set_value(X_target, [2,2,0,0])

for i in range(controller['Nsim']):
    solver.set_value(X_init, x0)

    sol = solver.solve()
    u0 = sol.value(U[:, 0])

    x0[0]=x0[0]+plant['Te']*x0[2]
    x0[1]=x0[1]+plant['Te']*x0[3]
    x0[2]=x0[2]+plant['Te']*u0[0]
    x0[3]=x0[3]+plant['Te']*u0[1]
```

## 3 Proposed exercises

*Exercise* 1. The code associated with this lab only implements the case with a single obstacle. How would you go about generalizing your case with multiple obstacles?

*Exercise* 2. We considered a very simple model (hence not a realistic one). Modify the code order to use the Dubins car dynamics:

i) Test the trajectories obtained in the case without obstacles;

ii) Check the behavior by varying the custom restrictions and the length of the prediction horizon. Illustrate the calculation time.