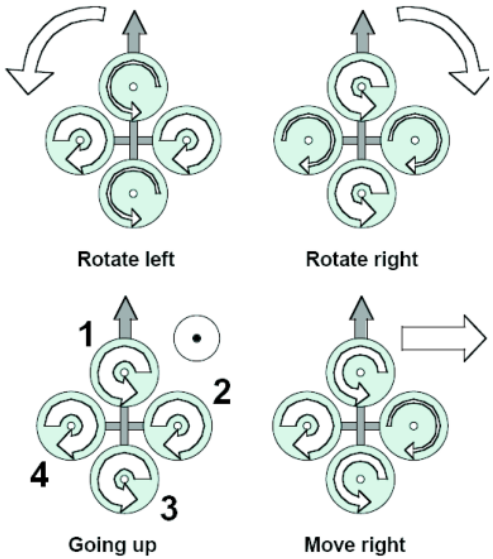
Making Flips with Quadrotors in Constrained Environments

Elie Hatem

Advisors: Dr. Sebastien Briot, Dr. Isabelle Fantoni

August 26, 2021

(a) DJI Phantom quadcopter (UAV)[1]



(b) Quadrotor Concept. Width of the arrows is proportional to the angular speed of the propellers[2]

Figure: Commercial quadrtotor platform (left) and quadrotor concept (right).

[1] https://en.wikipedia.org/wiki/Quadcopter#/media/File:Quadcopter_camera_drone_in_flight.jpg

[2] Design and control of quadrotors with application to autonomous flying, 2007, S. Bouabdallah

Over the last few years, quadrotors have gained large popularity in academia and industry. Because, they are:

- Simple to design and assemble using relatively cheap components.
- Different use cases: aerial photography, agriculture, surveillance tasks, etc.
- Quite agile and maneuverable during flight, especially when compared to other types of UAVs.

One of the main challenges

Designing control and planning methods to allow tracking aggressive trajectories.

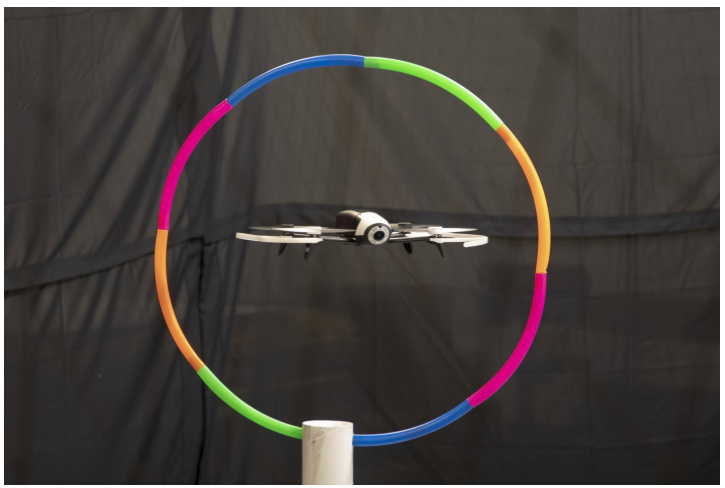
This difficulty is due to:

- The fast dynamics associated with the small dimensions of such agile quadrotors.
- Several dynamic effects will become important to consider during aggressive maneuvers.
- The motors will be commanded high speeds and accelerations, which will cause them to saturate and introduce delays.

The goal of the master thesis:

- Design a multi-flip trajectory.
- Implement Model Predictive Control to solve the presented issues.
- Perform the maneuvers in a constrained environment.

(a) Quadrotor performing a triple flip[3]



(b) Quadrotor going though a loop[4]

Figure: Representation of the issues to be tackled in this master thesis.

[3] Adaptive fast open-loop maneuvers for quadrocopters, 2012, S. Lupashin and R. D'Andrea

[4] `https://newatlas.com/drones/muscle-signals-drone-control/`

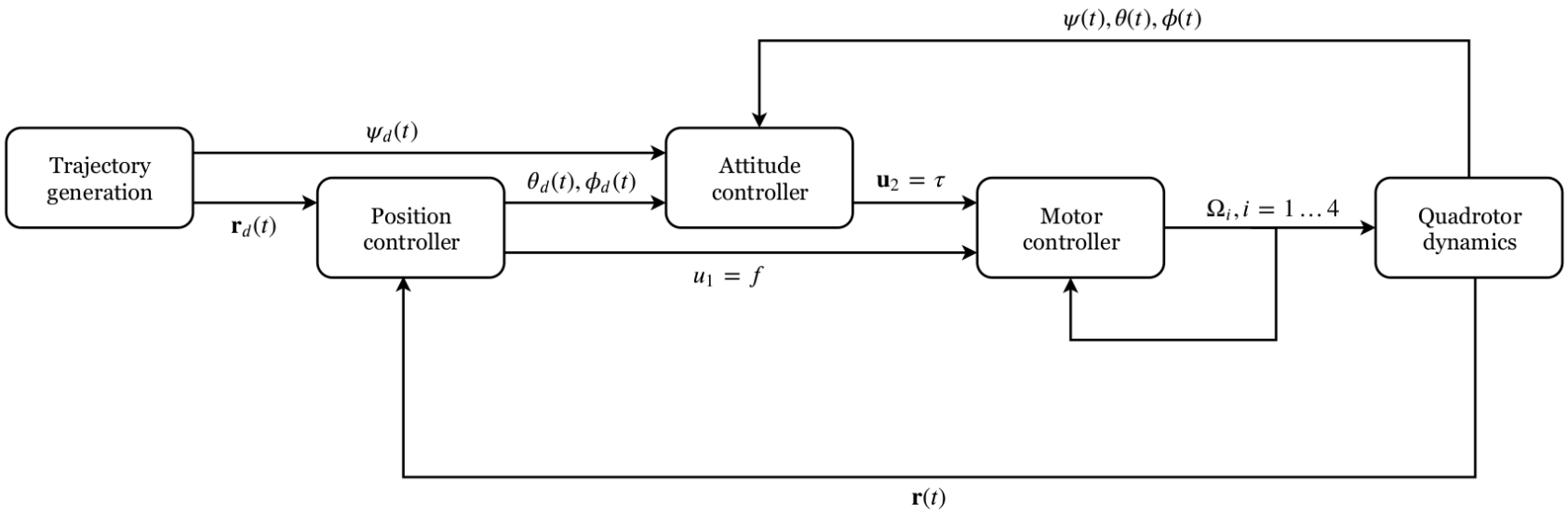


Figure: Diagram of the general control architecture of a quadrotor.

- Position controller: slow rise time - drives translational dynamics errors to 0.
- Attitude controller: faster rise time - drives rotational dynamics errors to 0.
- Motor controller: fastest rise time - maps the control inputs to motor speeds.

Remark

- The designed controller cannot be faster than the one at a lower level.
  - The orientation cannot be controlled any faster than the motors can be controlled.

A well-established finding is that the dynamic model of a quadrotor is differentially flat.

- Very useful property in under-actuated systems.
- Allows to generate trajectories in the lower dimensional space.
- The trajectories can then be mapped into the full dimensional space.

The standard choice of flat outputs for quadrotors are:

$$\mathbf{y} = \begin{bmatrix} x & y & z & \psi \end{bmatrix}^{\mathsf{T}} \tag{1}$$

As a result:

- Trajectories can be designed in the 4-dimensional space.
- They can then be mapped to the 6-dimenstional space.
  - This is due to the fact that the rotational and translational dynamics are tightly coupled.

General idea of MPC:

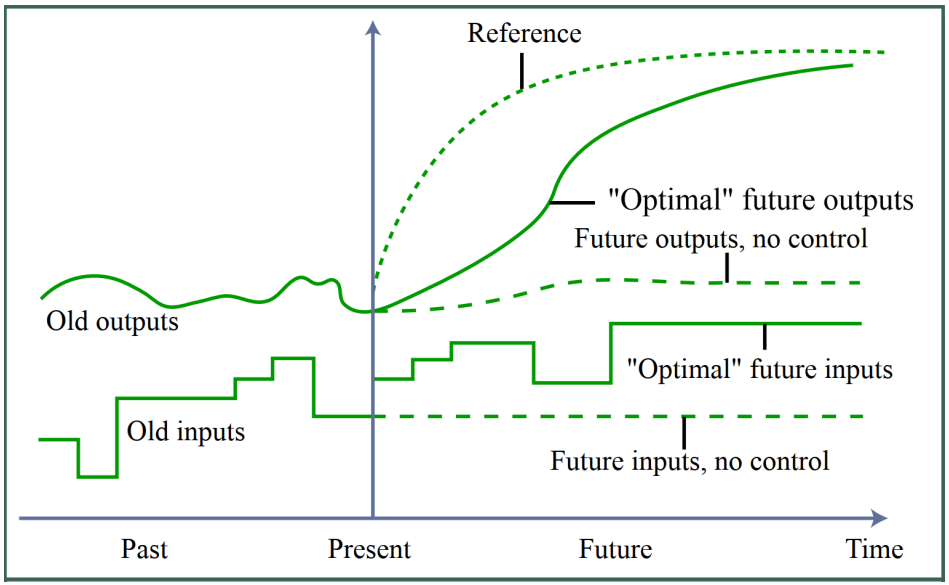


Figure: Basic idea of MPC[5]

- It is a **feedback control** algorithm.
- It uses a model to **predict** future outputs.
- It **solves an online optimization problem** to select the optimal control.

[5] Principles of Optimal Control, 2008, J. How

MPC Design parameters:

- Sample time.
- Prediction horizon.
- Control horizon.
- Constraints.
- Weights.

Choosing proper values for these parameters is important as they affect:

- The controller performance.
- The computational complexity of the MPC algorithm.

MPC Design parameters:

- Sample time.
- Prediction horizon.
- Control horizon.
- Constraints.
- Weights.

Choosing proper values for these parameters is important as they affect:

- The controller performance.
- The computational complexity of the MPC algorithm.

The most popular uses for MPC in quadrotors are:

- Centralized MPC: Single control loop for the system.
- Non-centralized MPC: Cascaded control consisting of more than 1 control loop. Examples:
  - MPC_{master}-MPC_{slave}
  - MPC-PD-P
  - Other options can be used for the inner loop.

Remark

- Centralized MPC: More accurate, high computation cost.
- Non-centralized MPC: Less accurate, lower computation cost.

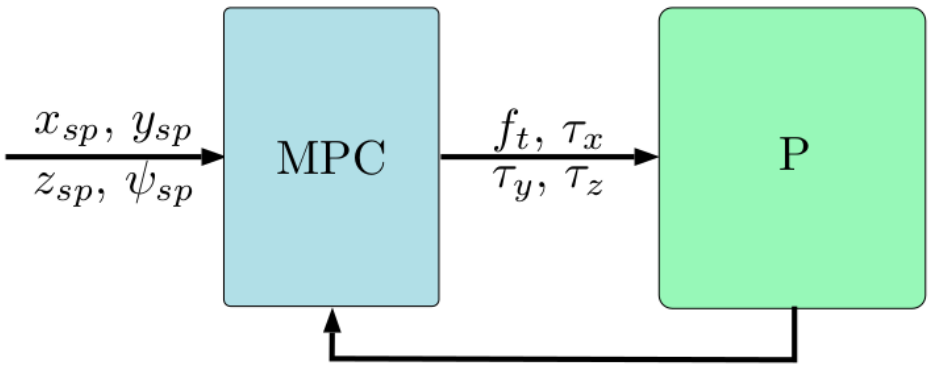


Figure: Centralized MPC[6]

Inputs of centralized MPC:

- Desired x, y and z positions and the yaw angle ψ.

Outputs of centralized MPC:

- Total thrust f_t.
- Torques: τ_x, τ_y and τ_z

Another version of the centralized MPC exists with:

- Added ϕ and θ angles as outputs.

[6] Design of predictive control structures track trajectories for multi-totor unmanned aerial vehicle, 2019, R.Alvarez-Valle et al.

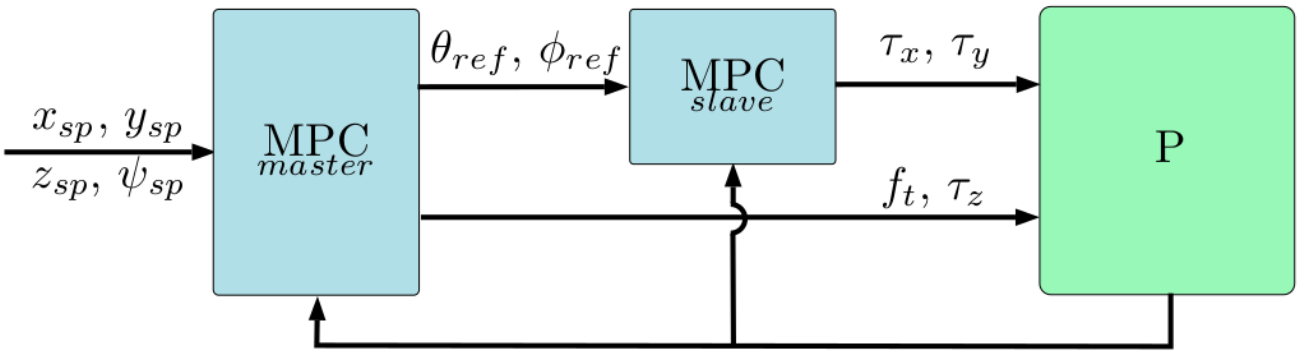


Figure: Non-centralized MPC (MPC$_{master}$-MPC$_{slave}$)[6]

Outer-loop: master MPC

- Inputs: Desired x, y, z, ψ.
- Outputs: f_t, τ_z, θ_{ref}, ϕ_{ref}.

Inner-loop: slave MPC

- Inputs: θ_{ref}, ϕ_{ref}.
- Outputs: τ_x, τ_y.

[6] Design of predictive control structures track trajectories for multi-totor unmanned aerial vehicle, 2019, R.Alvarez-Valle et al.

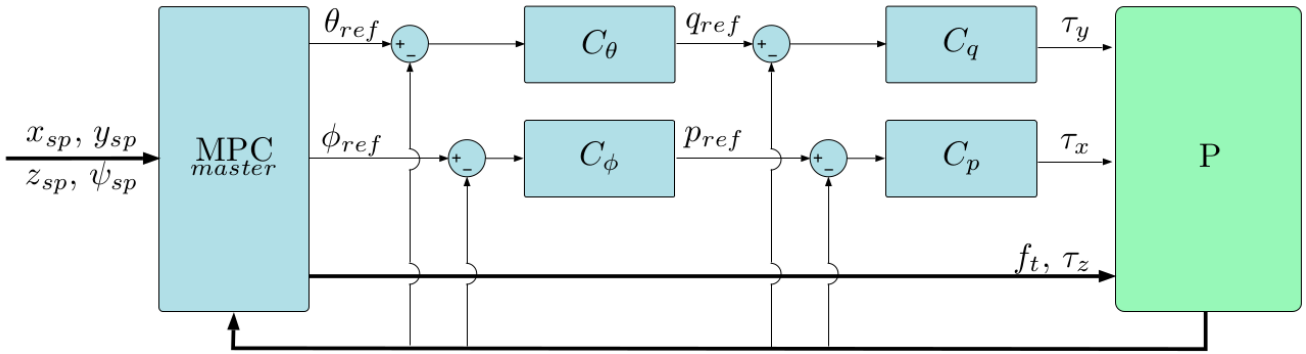


Figure: Non-centralized MPC (MPC-PD-P)[6]

Outer-loop: master MPC

- Inputs: Desired x, y, z, ψ.
- Outputs: f_t, τ_z, θ_{ref}, ϕ_{ref}

Inner-loop: PD-P controller

- Inputs: θ_{ref}, ϕ_{ref}.
- Outputs: τ_x, τ_y.

[6] Design of predictive control structures track trajectories for multi-totor unmanned aerial vehicle, 2019, R.Alvarez-Valle et al.

Another example of a non-centralized MPC:

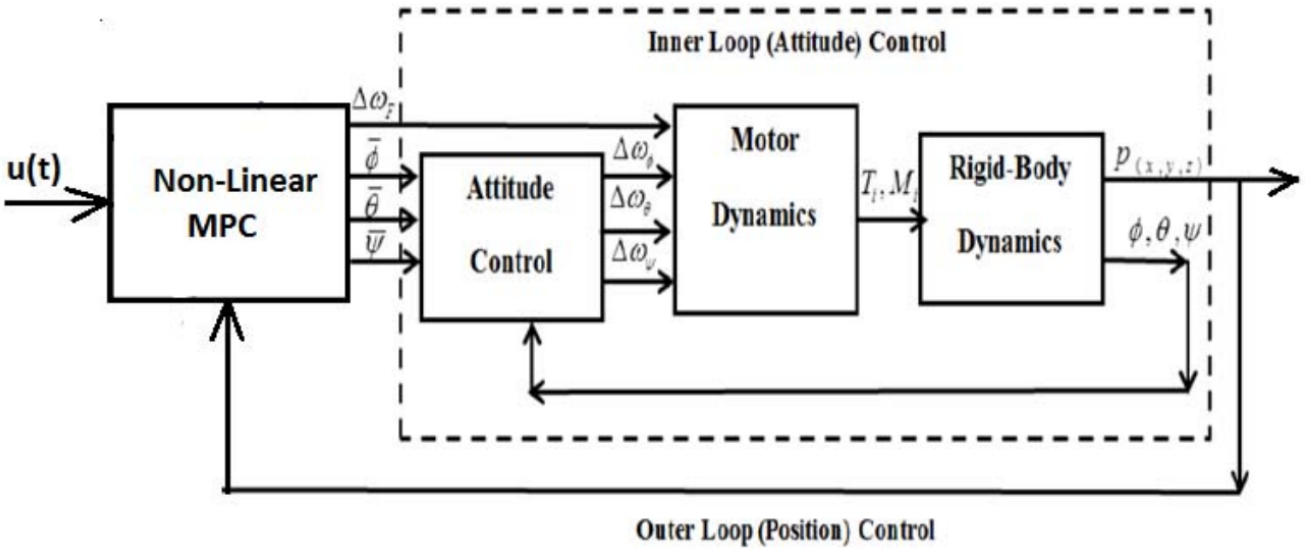


Figure: Non-centralized MPC[7]

Remark

The inner-loop can remain fixed, while the outer loop can be reprogrammed to meet the required task.

[7] Robust Hybrid Nonlinear Control Systems for the Dynamics of a Quadcopter Drone, 2020, F. Santoso et al.

The software used for implementing the MPC controllers is `acados`:

- Contains efficient optimal control algorithms implemented in C.
- Has a modular architecture.
- Interfaces to `C++`, `Python` and `MATLAB`.
- Uses the high-performance linear algebra package `BLASFEO`.
- Compatible with `CasADi` expressions.
- Deployable on a variety of embedded devices.
- Free and open-source software.

Main drawback:

- Prediction horizon and control horizon must be of same length.
  - This issue can be solved using the real-time iteration (RTI) method.

The general form of the nonlinear program that can be handled by `acados` is:

$$\begin{aligned}
\min_{\substack{x_0,\dots,x_N \\ u_0,\dots,u_{N-1} \\ z_0,\dots,z_{N-1} \\ s_0,\dots,s_N}} \quad & \sum_{k=0}^{N-1} l_k(x_k, u_k, z_k) + M(x_N) + \sum_{k=0}^{N} \rho_k(s_k) \\
\text{s.t.} \quad & \begin{bmatrix} x_{k+1} \\ z_k \end{bmatrix} = \phi_k(x_k, u_k) && k = 0, 1, \dots, N-1, \\
& 0 \geq g_k(x_k, z_k, u_k) - J_{s,k} s_k && k = 0, 1, \dots, N-1, \\
& 0 \geq g_N(x_N) - J_{s,N} s_N, \\
& 0 \leq s_k && k = 0, 1, \dots, N-1
\end{aligned} \tag{2}$$

And,

$$\rho_k(s_k) = \sum_{i=1}^{n_{s_k}} \alpha_k^i s_k^i + \beta_k^i s_k^{i^2} \tag{3}$$

with $\alpha_k^i \in \mathbb{R}, \beta_k^i > 0$.

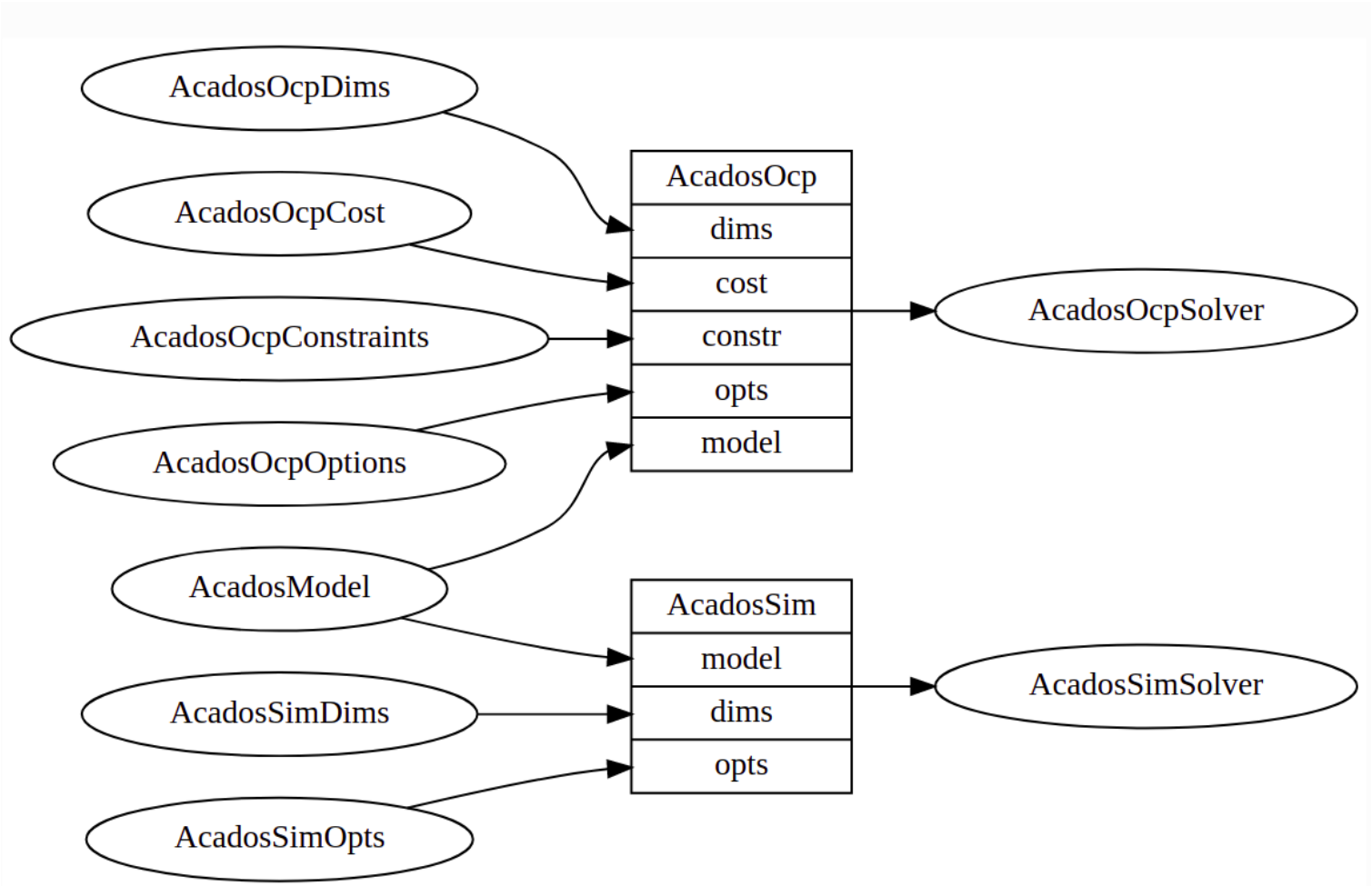


Figure: Overview of the Python API classes in acados[8]

[8] https://docs.acados.org/python_api/

The equations of motion of a planar quadrotor are:

$$\begin{cases} \ddot{y} = -\frac{u_1}{m} \sin\phi \\ \ddot{z} = -g + \frac{u_1}{m} \cos\phi \\ \ddot{\phi} = \frac{u_2}{I_{xx}} \end{cases} \tag{4}$$

With:

- u_1: Total thrust of the two rotors.
- u_2: Torque applied along the x-axis.

The equations of motion of a 3D quadrotor are:

$$\text{Translational model: } \begin{cases} \dot{\boldsymbol{p}}_i &= \boldsymbol{v}_i \\ \dot{\boldsymbol{v}}_i &= \frac{T}{m} \begin{bmatrix} 2(q_w q_y + q_x q_z) \\ 2(q_y q_z - q_w q_x) \\ 1 - 2(q_x^2 + q_y^2) \end{bmatrix} + \boldsymbol{g} \end{cases}$$

$$\text{Rotational model: } \begin{cases} \dot{\boldsymbol{q}}_i &= \frac{1}{2} \begin{bmatrix} 0 \\ \boldsymbol{\omega}_i \end{bmatrix} \otimes \boldsymbol{q}_i \\ \dot{\boldsymbol{\omega}}_i &= \boldsymbol{I}_i^{-1} \boldsymbol{\tau}_i - \boldsymbol{I}_i^{-1} (\boldsymbol{\omega}_i \times \boldsymbol{I}_i \boldsymbol{\omega}_i) \end{cases}$$

However, the MPC controller will not account for all of the equations (Non-centralized MPC):

$$\text{Treated by the MPC: } \begin{cases} \dot{\boldsymbol{p}}_i &= \boldsymbol{v}_i \\ \dot{\boldsymbol{v}}_i &= \frac{T}{m} \begin{bmatrix} 2(q_w q_y + q_x q_z) \\ 2(q_y q_z - q_w q_x) \\ 1 - 2(q_x^2 + q_y^2) \end{bmatrix} + \boldsymbol{g} \\ \dot{\boldsymbol{q}}_i &= \frac{1}{2} \begin{bmatrix} 0 \\ \boldsymbol{\omega}_i \end{bmatrix} \otimes \boldsymbol{q}_i \end{cases}$$

$$\text{Treated by the L.L. controller: } \left\{ \; \dot{\boldsymbol{\omega}}_i = \boldsymbol{I}_i^{-1} \boldsymbol{\tau}_i - \boldsymbol{I}_i^{-1} (\boldsymbol{\omega}_i \times \boldsymbol{I}_i \boldsymbol{\omega}_i) \right.$$

Remark

- A low level controller is assumed to exist to map the angular rates inputs $\boldsymbol{\omega}$ to the required torques $\boldsymbol{\tau}$.
- The equations above do not represent the complete model of the system.
  - However, they are sufficient to control the quadrotor.

Figure: Crazyflie 2.1 with added mass

- $m_{no\_load} = 29.5g$
- $m_{with\_load} = 47g$
- $T_{max}^{actual} = (47 \times 10^{-3} kg)9.81 \frac{m}{s^2} = 0.46N$
- $I_{xx} = 1.657171 \times 10^{-5} kg.m^2$
- $I_{yy} = 1.657171 \times 10^{-5} kg.m^2$
- $I_{zz} = 2.9261652 \times 10^{-5} kg.m^2$
- $L = 0.046m$

For the planar quadrotor case:

$$\begin{aligned} \min_{x,u} \quad & \frac{1}{2}\| \boldsymbol{V}_x \boldsymbol{x} + \boldsymbol{V}_u \boldsymbol{u} + \boldsymbol{V}_z \boldsymbol{z} - \boldsymbol{y}_{ref} \|^2_{\boldsymbol{W}} + \frac{1}{2}\| \boldsymbol{V}^e_x \boldsymbol{x} - \boldsymbol{y}^e_{ref} \|^2_{\boldsymbol{W}_e} \\ \text{s.t.} \quad & f(\boldsymbol{x}, \boldsymbol{u}) : \text{dynamics} \\ & u_{1_{min}} \leq u_1 \leq u_{1_{max}} \\ & -u_{2_{max}} \leq u_2 \leq u_{2_{max}} \end{aligned} \tag{5}$$

with:

- $\boldsymbol{V}_x \in \mathbb{R}^{n_y \times n_x}$
- $\boldsymbol{V}_u \in \mathbb{R}^{n_y \times n_u}$
- $\boldsymbol{V}_z \in \mathbb{R}^{n_y \times n_z}$
- $\boldsymbol{V}^e_x \in \mathbb{R}^{n_{ye} \times n_x}$
- $\boldsymbol{y}_{ref} \in \mathbb{R}^{n_y}$
- $\boldsymbol{y}_{ref_e} \in \mathbb{R}^{n_{ye}}$
- $\boldsymbol{W} \in \mathbb{R}^{n_y \times n_y}$
- $\boldsymbol{W}_e \in \mathbb{R}^{n_{ye} \times n_{ye}}$
- $\boldsymbol{Q}, \boldsymbol{Q}_e \in \mathbb{R}^{n_x \times n_x}$
- $\boldsymbol{R} \in \mathbb{R}^{n_u \times n_u}$
- $\boldsymbol{W} = diag(\boldsymbol{Q}, \boldsymbol{R})$
- $\boldsymbol{W}_e = \boldsymbol{Q}_e$

The state space variables:

$$\boldsymbol{x} = [y, z, \phi, v_y, v_z, \dot{\phi}]^\intercal \tag{6}$$

The state evolves according to:

$$\dot{y} = v_y \tag{7}$$

$$\dot{z} = v_z \tag{8}$$

$$\dot{\phi} = \dot{\phi} \tag{9}$$

$$\dot{v}_y = -\frac{u_1}{m}\sin(\phi) \tag{10}$$

$$\dot{v}_z = -g + \frac{u_1}{m}\cos(\phi) \tag{11}$$

$$\ddot{\phi} = \frac{u_2}{I_{xx}} \tag{12}$$

Control inputs: $\boldsymbol{u} = [u_1, u_2]^\intercal$

Initial condition:

$\boldsymbol{x}_0 = [y_0, z_0, \phi_0, v_{y_0}, v_{z_0}, \ddot{\phi}_0]^\intercal$

Desired states:

$\boldsymbol{x}_d = [y_d, z_d, \phi_d, v_{y_d}, v_{z_d}, \ddot{\phi}_d]^\intercal$

MPC parameters:

- $N = 100$
- $T_f = 1s$

Max. and min. thrust and torque:

- $u_{1_{max}} = 0.9\left(\frac{0.46N}{2}\right)$
- $u_{1_{min}} = 0.1(u_{1_{max}})$
- $u_{2_{max}} = 0.1\left(\frac{1}{2}u_{1_{max}}L\right)$
- $u_{2_{min}} = -u_{2_{max}}$

For the 3D quadrotor case:

$$\begin{aligned} \min_{x,u} \quad & \frac{1}{2}\|\boldsymbol{V}_x\boldsymbol{x} + \boldsymbol{V}_u\boldsymbol{u} + \boldsymbol{V}_z\boldsymbol{z} - \boldsymbol{y}_{ref}\|^2_{\boldsymbol{W}} + \frac{1}{2}\|\boldsymbol{V}^e_x\boldsymbol{x} - \boldsymbol{y}^e_{ref}\|^2_{\boldsymbol{W}_e} \\ \text{s.t.} \quad & f(\boldsymbol{x},\boldsymbol{u}) : \text{dynamics} \\ & T_{min} \leq T \leq T_{max} \\ & -4\pi \leq \omega_x, \omega_y, \omega_z \leq 4\pi \end{aligned} \tag{13}$$

with:

- $\boldsymbol{V}_x \in \mathbb{R}^{n_y \times n_x}$
- $\boldsymbol{V}_u \in \mathbb{R}^{n_y \times n_u}$
- $\boldsymbol{V}_z \in \mathbb{R}^{n_y \times n_z}$
- $\boldsymbol{V}^e_x \in \mathbb{R}^{n_{ye} \times n_x}$
- $\boldsymbol{y}_{ref} \in \mathbb{R}^{n_y}$
- $\boldsymbol{y}_{ref_e} \in \mathbb{R}^{n_{ye}}$
- $\boldsymbol{W} \in \mathbb{R}^{n_y \times n_y}$
- $\boldsymbol{W}_e \in \mathbb{R}^{n_{ye} \times n_{ye}}$
- $\boldsymbol{Q}, \boldsymbol{Q}_e \in \mathbb{R}^{n_x \times n_x}$
- $\boldsymbol{R} \in \mathbb{R}^{n_u \times n_u}$
- $\boldsymbol{W} = diag(\boldsymbol{Q}, \boldsymbol{R})$
- $\boldsymbol{W}_e = \boldsymbol{Q}_e$

The state space variables:

$$\boldsymbol{x} = [x, y, z, q_w, q_x, q_y, q_z, v_x, v_y, v_z,]^\mathsf{T} \tag{14}$$

Control inputs: $\boldsymbol{u} = [T, \omega_x, \omega_y, \omega_z]^\mathsf{T}$
Initial condition:
$\boldsymbol{x}_0 = [x_0, y_0, z_0, q_{w_0}, q_{x_0}, q_{y_0}, q_{z_0}, v_{x_0}, v_{y_0}, v_{z_0}]^\mathsf{T}$
Desired states:
$\boldsymbol{x}_d = [x_d, y_d, z_d, q_{w_d}, q_{x_d}, q_{y_d}, q_{z_d}, v_{x_d}, v_{y_d}, v_{z_d}]^\mathsf{T}$
MPC parameters:

- $N = 100$
- $T_f = 1s$

Maximum and minimum thrust:

- $T_{max} = 0.9(0.46N)$
- $T_{min} = 0.1(T_{max})$

The states evolve according to:

$$\dot{x} = v_x$$

$$\dot{y} = v_y$$

$$\dot{z} = v_z$$

$$\dot{q}_w = \frac{1}{2}(-w_x q_x - w_y q_y - w_z q_z)$$

$$\dot{q}_x = \frac{1}{2}(w_x q_w + w_z q_y - w_y q_z)$$

$$\dot{q}_y = \frac{1}{2}(w_y q_w - w_z q_x + w_x q_z)$$

$$\dot{q}_z = \frac{1}{2}(w_z q_w + w_y q_x - w_x q_y)$$

$$\dot{v}_x = 2(q_w q_y + q_x q_z)\frac{T}{m}$$

$$\dot{v}_y = 2(q_y q_z - q_w q_x)\frac{T}{m}$$

$$\dot{v}_z = (1 - 2q_x^2 - 2q_y^2)\frac{T}{m} - g$$

The dynamic criterion of a planar quadrotor is[9]:

$$\ddot{y}\cos\phi + (\ddot{z} + g)\sin\phi = 0 \tag{15}$$

- It represents the dynamics constraints that must be satisfied to track a trajectory.

The flip trajectory is divided into 3 phases:

- Reaching phase: Transition from hovering position to the required speed to initialize the flip phase (3 parts).
- Flip phase: Flip is performed and the singularity positions are crossed (1 part).
- Recovery phase: Recovery and transition back to a hovering position (3 parts).

[9] Making aggressive maneuvers with drones thanks to parallel singularity crossing approaches, 2019, M. Orsingher

To design dynamically feasible flip trajectories:

- The reaching and recovery phases of z and ϕ will be generated using Polynomials of order 9.
- The trajectory along y will then be computed by integrating the dynamic criterion twice:

$$y(t) = -\iint (\ddot{z} + g) \tan \phi dt dt \tag{16}$$

Main problem with this strategy

- y(t) is smooth and bounded only if $\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$

In order to satisfy the dynamic criterion during the flip phase, a simple solution (ideal case) is to turn off the rotors ($u_1 = u_2 = 0$):

$$\begin{cases} \ddot{y} = 0 \\ \ddot{z} = -g \\ \ddot{\phi} = 0 \end{cases} \Rightarrow \begin{cases} y(t) = y(0) + \dot{y}(0)t \\ z(t) = z(0) + \dot{z}(0)t - \frac{g}{2}t^2 \\ \phi(t) = \phi(0) + \dot{\phi}(0)t \end{cases} \tag{17}$$

The other difficulty in the proposed trajectory planning method

- Finding the proper set of parameters to completely define the flip trajectory.

There are 37 parameters in total that must be chosen in order to generate the flip trajectory:

- $z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8$
- $\dot{z}_2, \dot{z}_3, \dot{z}_6, \dot{z}_7$
- $\ddot{z}_2, \ddot{z}_3, \ddot{z}_6, \ddot{z}_7$
- $\phi_2, \phi_3, \phi_4, \phi_5, \phi_6, \phi_7$
- $\dot{\phi}_2, \dot{\phi}_3, \dot{\phi}_6, \dot{\phi}_7$
- $\ddot{\phi}_2, \ddot{\phi}_3, \ddot{\phi}_6, \ddot{\phi}_7$
- $t_1, t_2, t_3, t_4, t_5, t_6, t_7$

So, a parameter vector $\boldsymbol{p}$ is created which contains all the parameters to be optimized:

$$\boldsymbol{p} = \begin{bmatrix} z_1 & \dots & t_7 \end{bmatrix} \tag{18}$$

Some of the parameters will be assigned simple bounds:

$$\boldsymbol{p}_{min} \leq \boldsymbol{p} \leq \boldsymbol{p}_{max} \tag{19}$$

Nonlinear inequality constraints were also used:

- At each iteration, u_1 and u_2 are extracted from the generated trajectory:

$$\boldsymbol{u}_1 = \frac{m(\ddot{\boldsymbol{z}} + \boldsymbol{g})}{\cos\phi} \tag{20}$$

$$\boldsymbol{u}_2 = I_{xx}\ddot{\phi} \tag{21}$$

- Then, the solver will make sure that u_1 and u_2 are within their bounds:

$$u_{1_{min}} < u_1 < u_{1_{max}} \tag{22}$$

$$u_{2_{min}} < u_2 < u_{2_{max}} \tag{23}$$

- Other constraints on z were added to make sure the generated trajectory is concave.

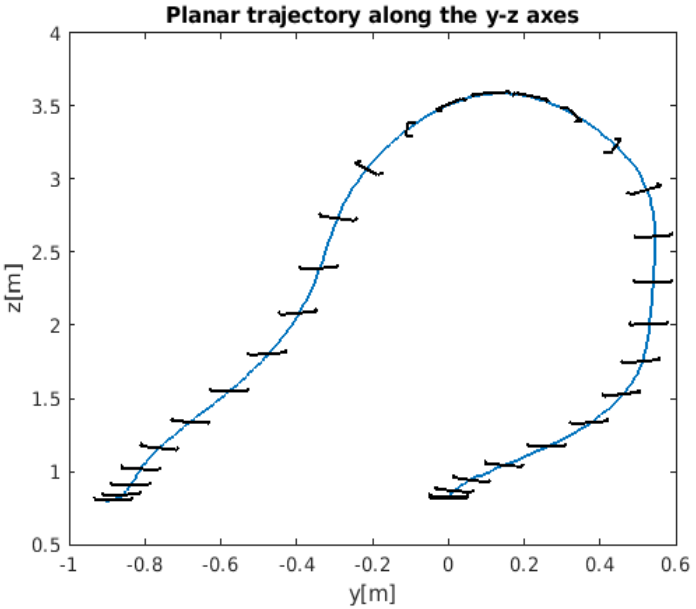
The parameter optimization problem can finally be expressed as follows:

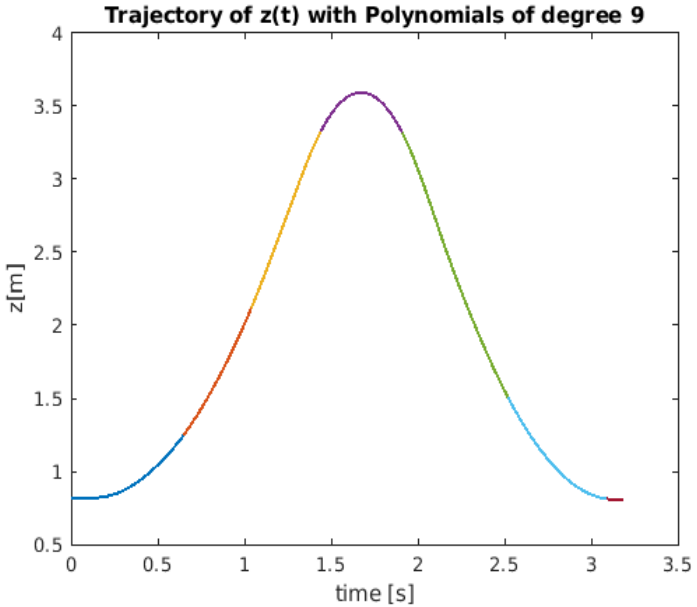
$$\begin{aligned} \min_{\boldsymbol{p}} \quad & J(\boldsymbol{p}) = \int_0^t u_1^2 \\ \text{s.t.} \quad & \boldsymbol{p}_{min} \leq \boldsymbol{p} \leq \boldsymbol{p}_{max} \\ & \boldsymbol{c}(\boldsymbol{p}) \leq \mathbf{0} \end{aligned} \tag{24}$$

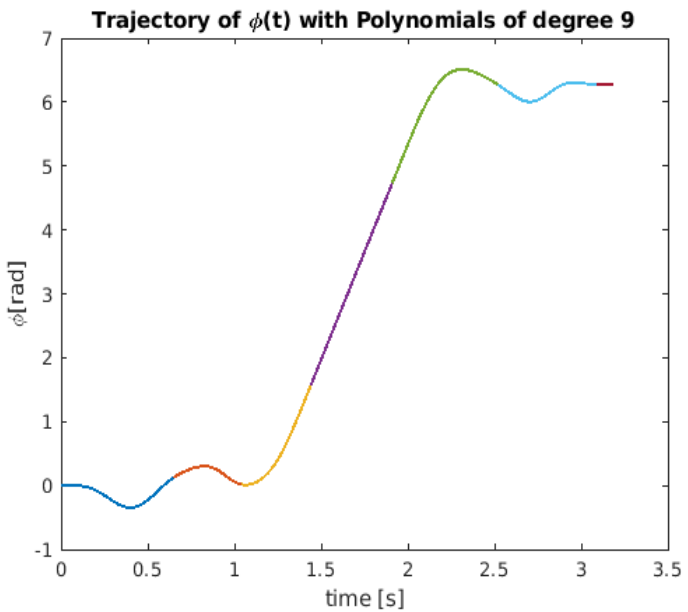
A solution for the optimization problem was found using the `GlobalSearch` solver in MATLAB:

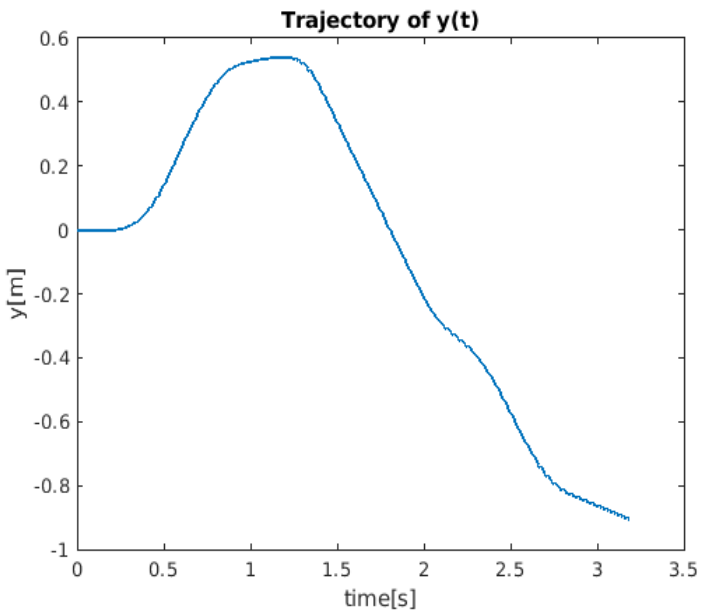
```
GlobalSearch stopped because it analyzed all the trial points.

1 out of 52 local solver runs converged with a positive local solver exit flag.

x =

  Columns 1 through 15

    0.8164    1.2486    2.1186    3.3263    3.3187    1.5076    0.8117    0.8031    1.6187    2.8544   -2.2642   -0.1652    3.4843    3.5045    3.7850

  Columns 16 through 30

    3.6209    0.1238    0.0180    1.5632    4.7301    6.2728    6.2722    1.8485   -0.6043   -1.7680   -0.0694  -10.6716   23.2159   -4.0151    6.6086

  Columns 31 through 37

    0.6509    0.3933    0.3972    0.4720    0.6143    0.5686    0.1050
```

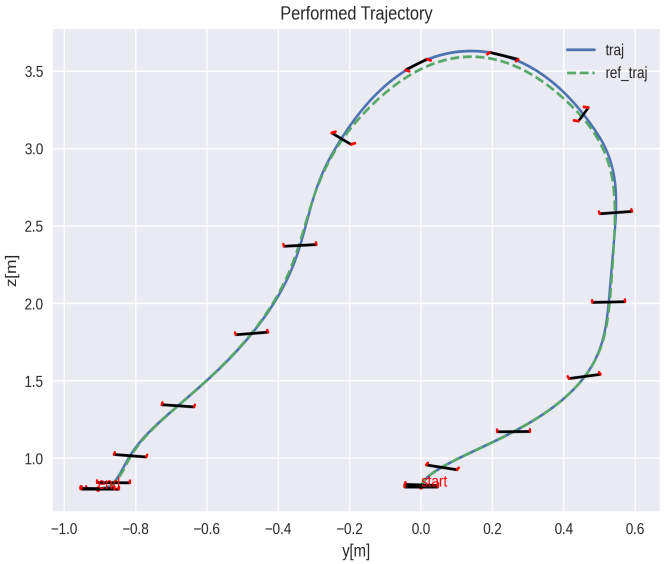
Figure: Output of `GlobalSearch` at the end of the optimization process

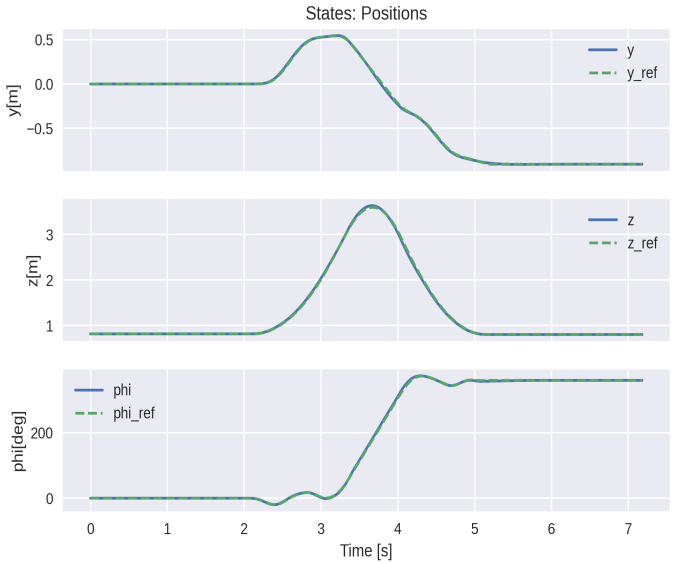

Planar trajectory along the y-z axes
z[m]
y[m]

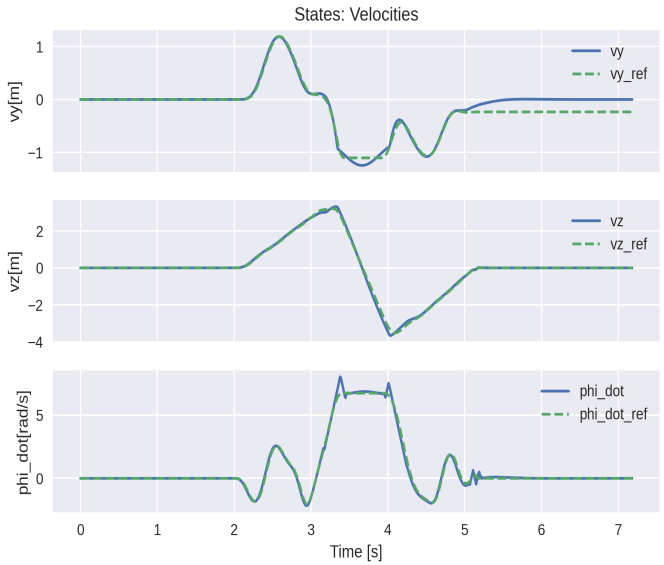

Trajectory of z(t) with Polynomials of degree 9
z[m]
time [s]

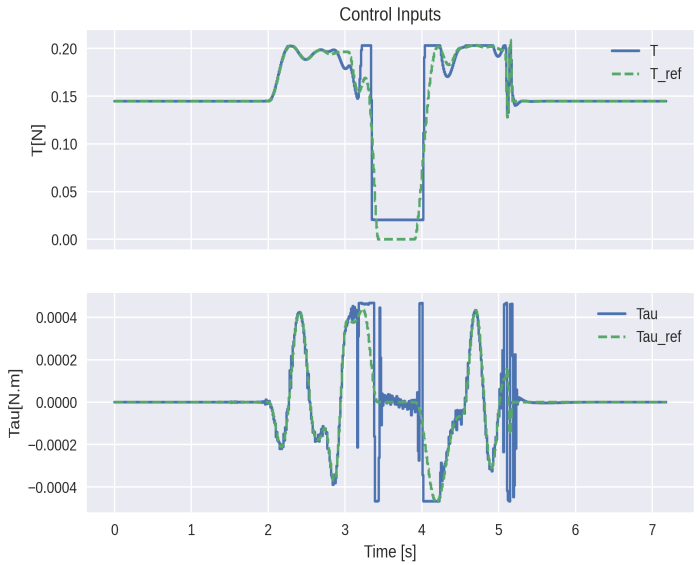

Trajectory of ϕ(t) with Polynomials of degree 9
ϕ[rad]
time [s]


Trajectory of y(t)
y[m]
time[s]

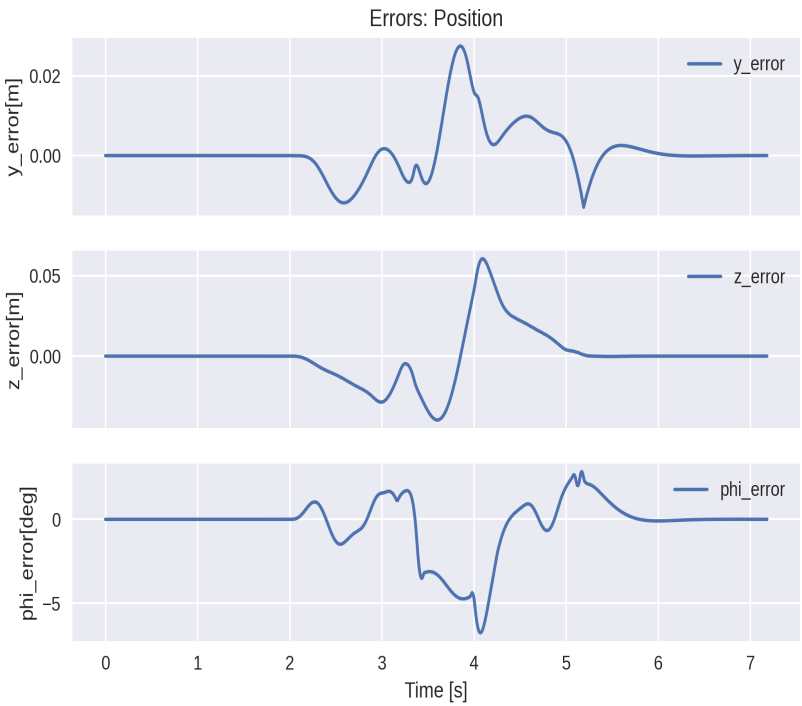

Performed Trajectory
traj
ref_traj
start
z[m]
y[m]

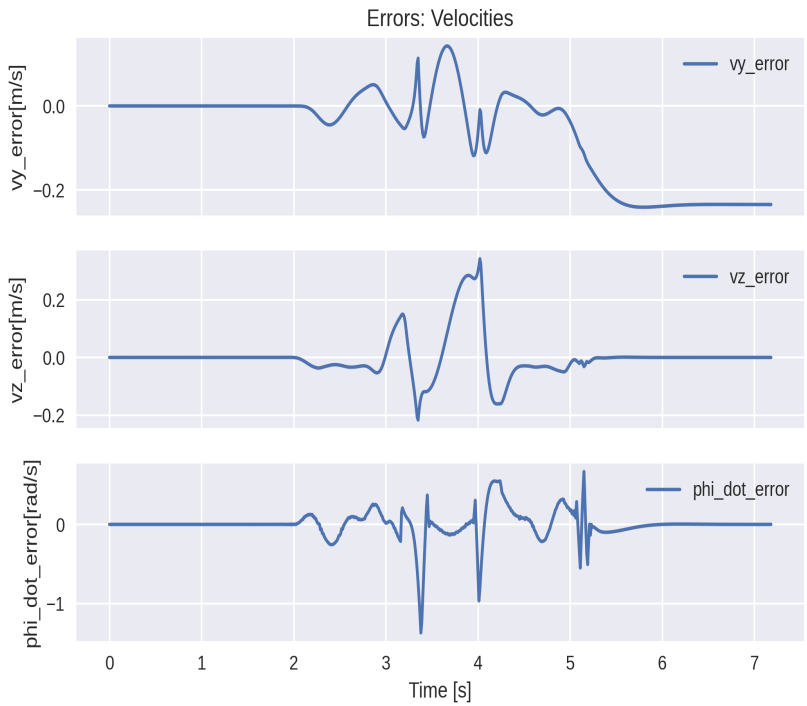

States: Positions
y
y_ref
y[m]
z
z_ref
z[m]
phi
phi_ref
phi[deg]
Time [s]

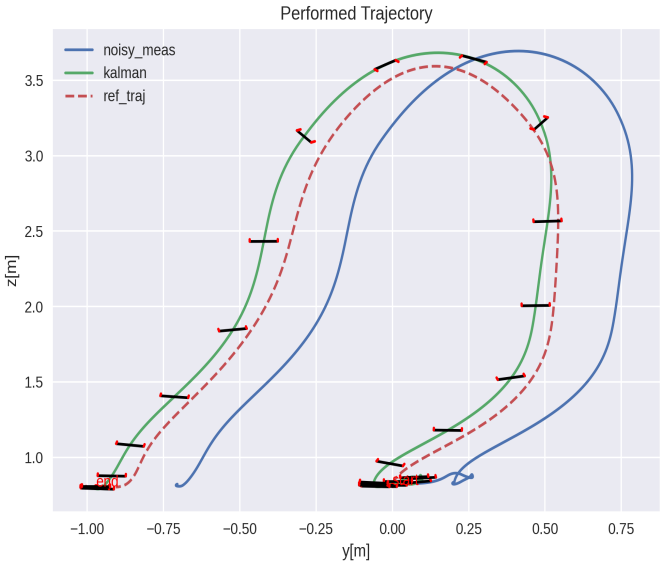

States: Velocities
vy
vy_ref
vy[m]
vz
vz_ref
vz[m]
phi_dot
phi_dot_ref
phi_dot[rad/s]
Time [s]

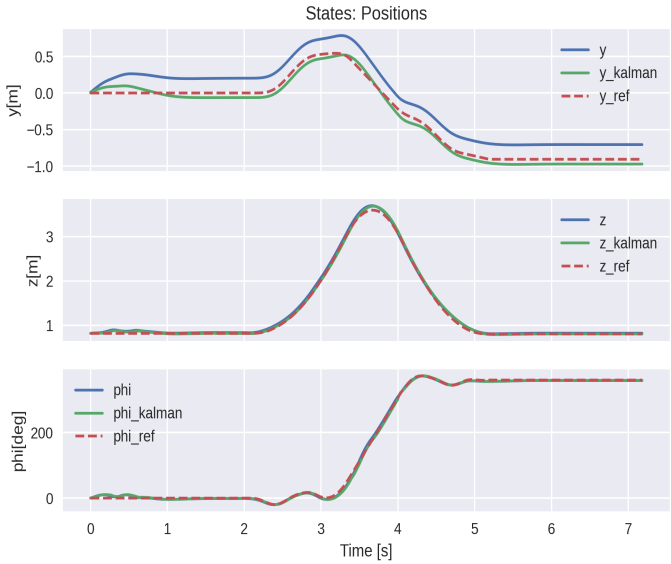

Control Inputs
T
T_ref
T[N]
Tau
Tau_ref
Tau[N.m]
Time [s]

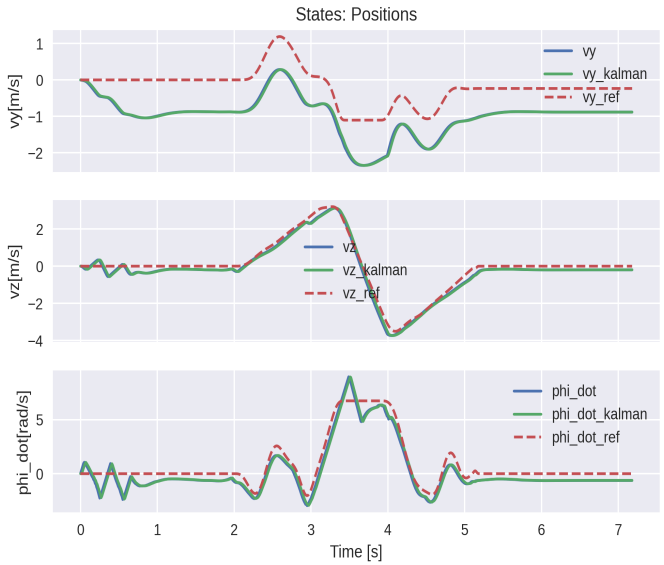
Simulations using acados for the Planar Quadrotor case - No added noise

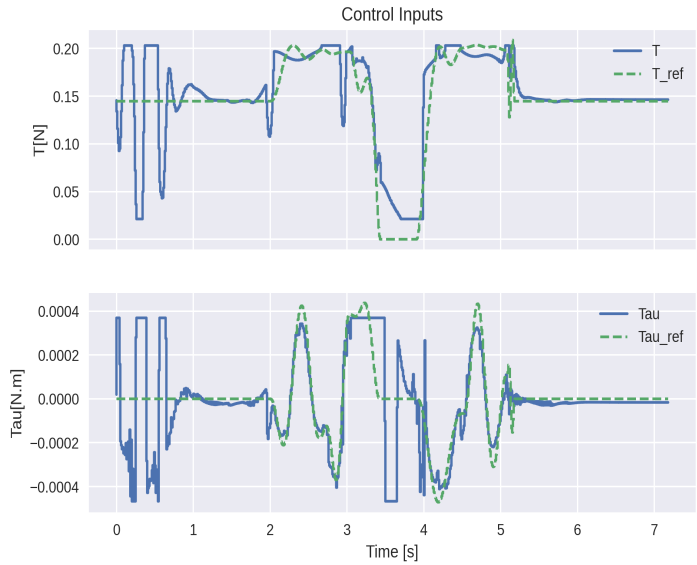


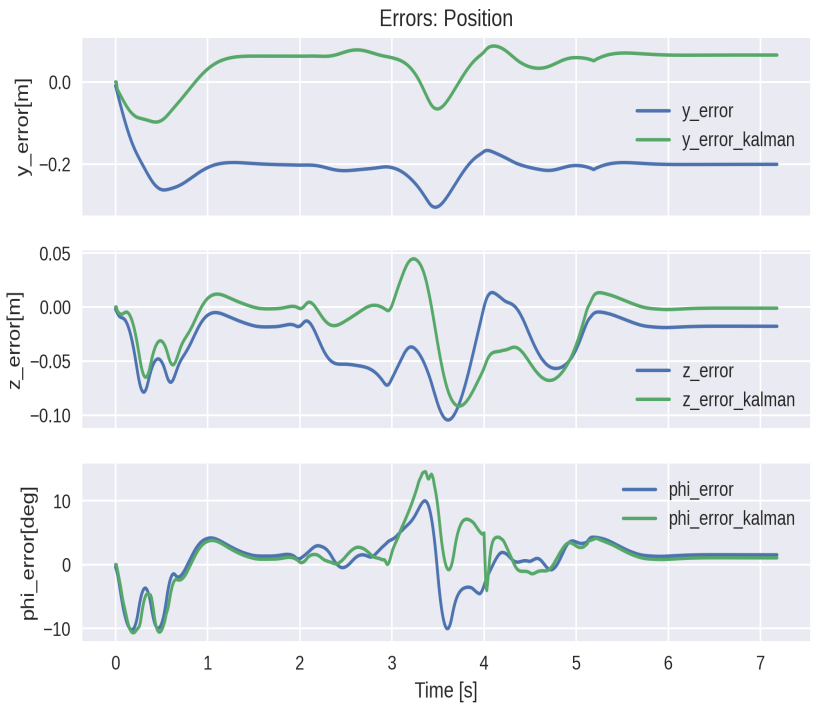


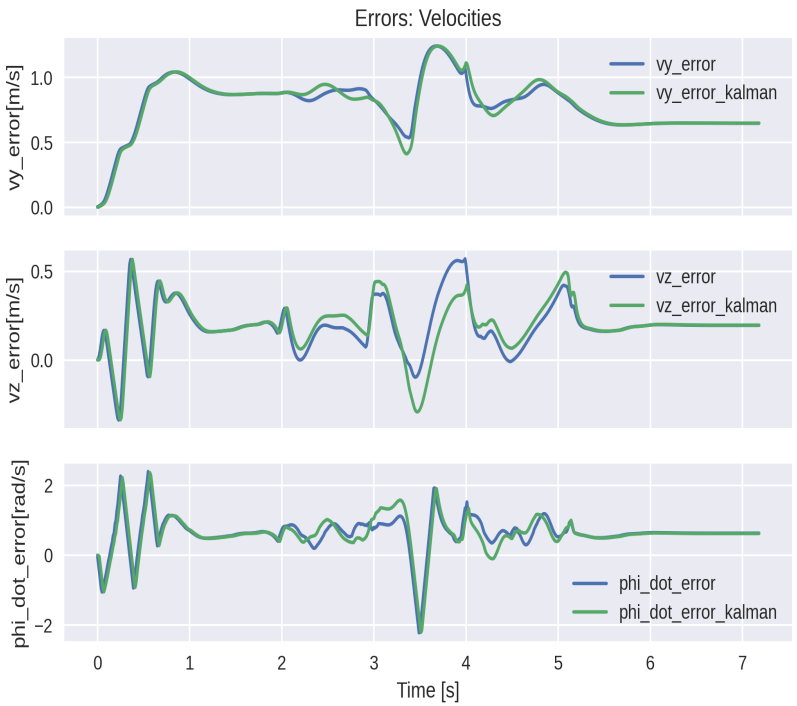

Performed Trajectory
noisy_meas
kalman
ref_traj
z[m]
y[m]


States: Positions
y
y_kalman
y_ref
y[m]
z
z_kalman
z_ref
z[m]
phi
phi_kalman
phi_ref
phi[deg]
Time [s]


States: Positions
vy
vy_kalman
vy_ref
vy[m/s]
vz
vz_kalman
vz_ref
vz[m/s]
phi_dot
phi_dot_kalman
phi_dot_ref
phi_dot[rad/s]
Time [s]


Control Inputs
T
T_ref
T[N]
Tau
Tau_ref
Tau[N.m]
Time [s]


Errors: Position
y_error[m]
0.0
−0.2
y_error
y_error_kalman
z_error[m]
0.05
0.00
−0.05
−0.10
z_error
z_error_kalman
phi_error[deg]
10
0
−10
phi_error
phi_error_kalman
0
1
2
3
4
5
6
7
Time [s]


Errors: Velocities
vy_error[m/s]
1.0
0.5
0.0
vy_error
vy_error_kalman
vz_error[m/s]
0.5
0.0
vz_error
vz_error_kalman
phi_dot_error[rad/s]
2
0
−2
phi_dot_error
phi_dot_error_kalman
0
1
2
3
4
5
6
7
Time [s]

The block diagram of the closed-loop control contains:

- The quadrotor plant $f(x)$
- The MPC controller

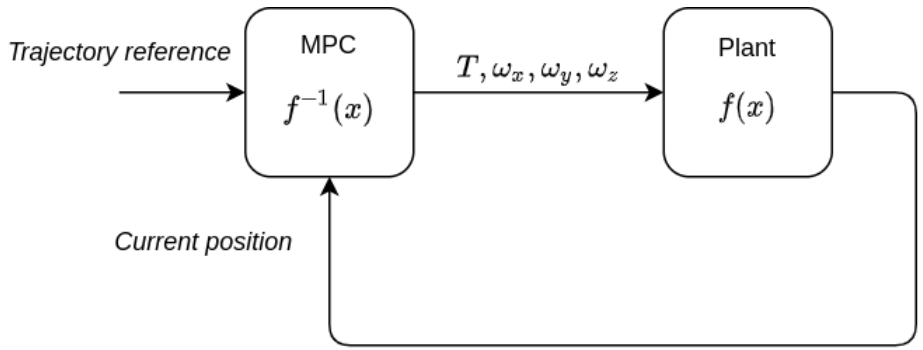


Figure: Diagram of the closed-loop control in the acados simulations

However, $f(x)$ is just an approximation of the real dynamics of the quadrotor $g(x)$

$$f(x) \approx g(x)$$

Remark

- The approximation is true if the angular dynamics are not very high.
  - The larger the angular dynamics, the worse the approximation becomes.
- A Software-in-the-Loop simulation must be used to have a more accurate representation of how the system will behave.

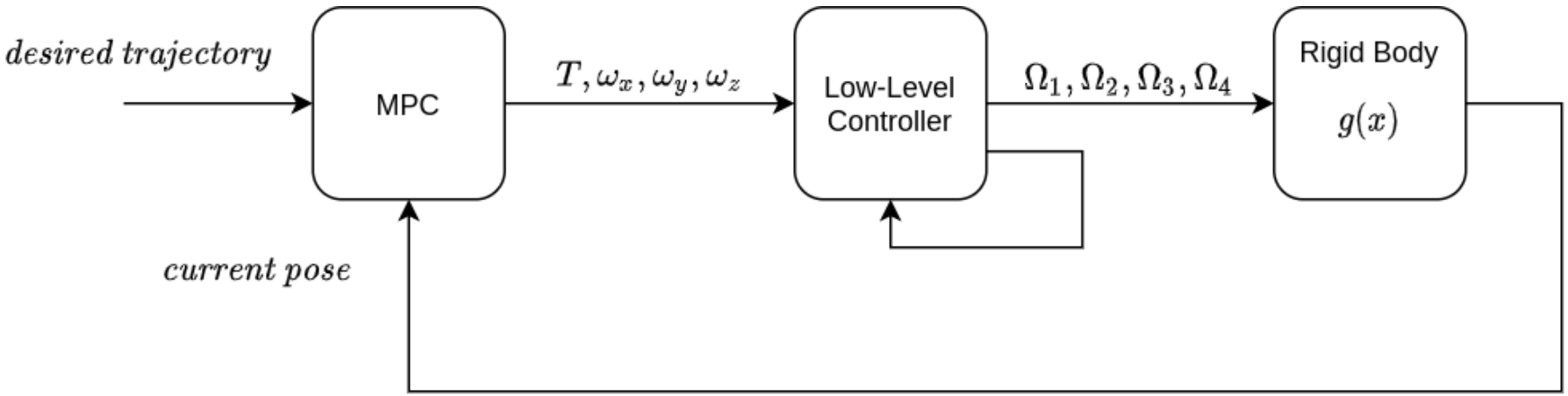


Figure: Diagram of the closed-loop control in the SIL simulation

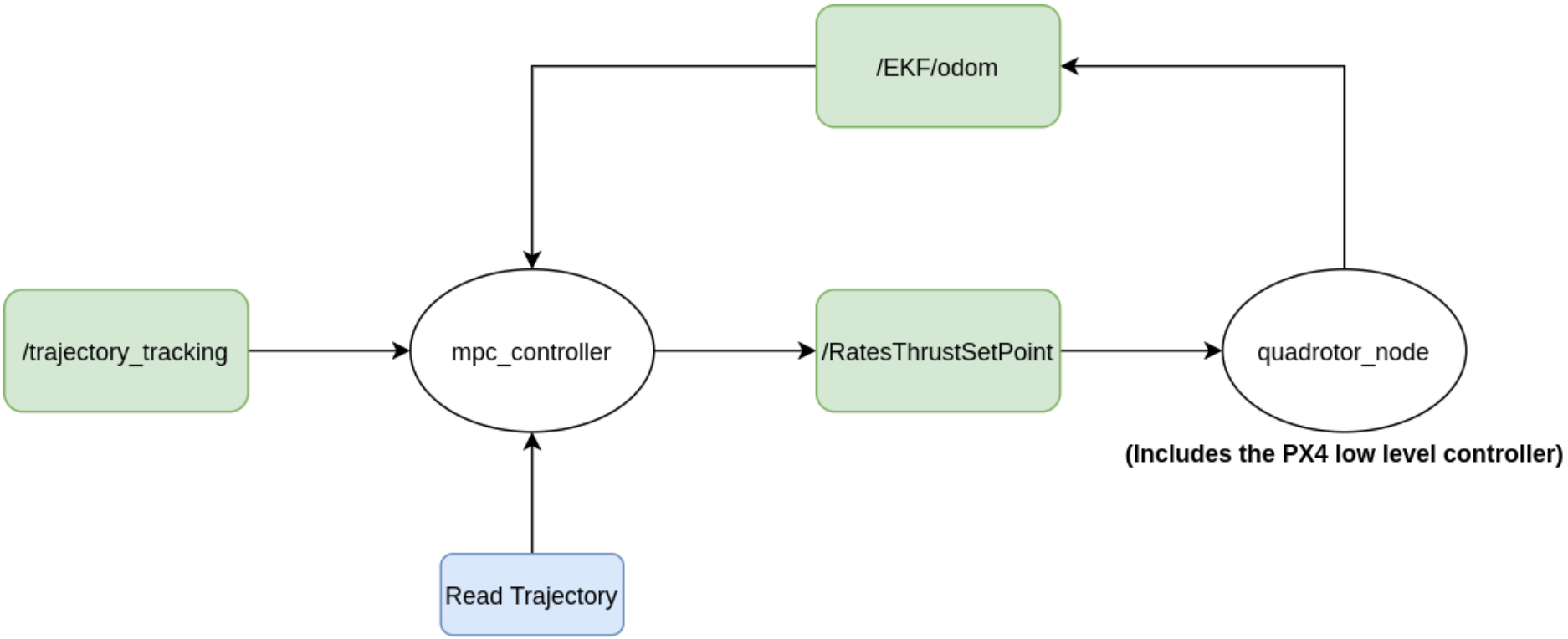
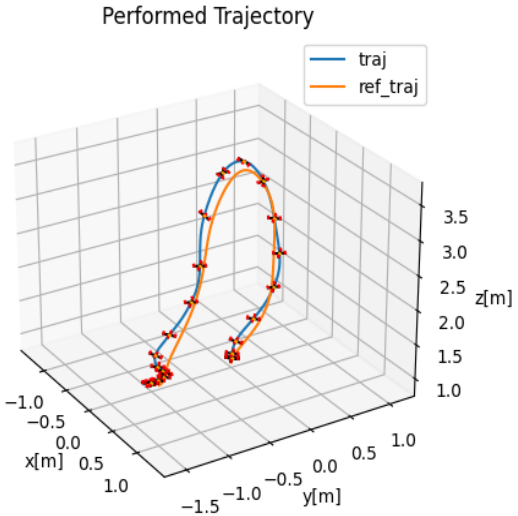
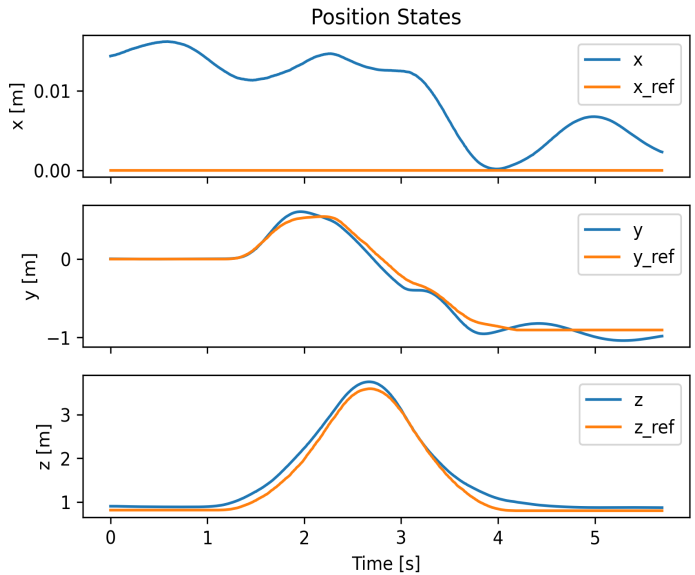


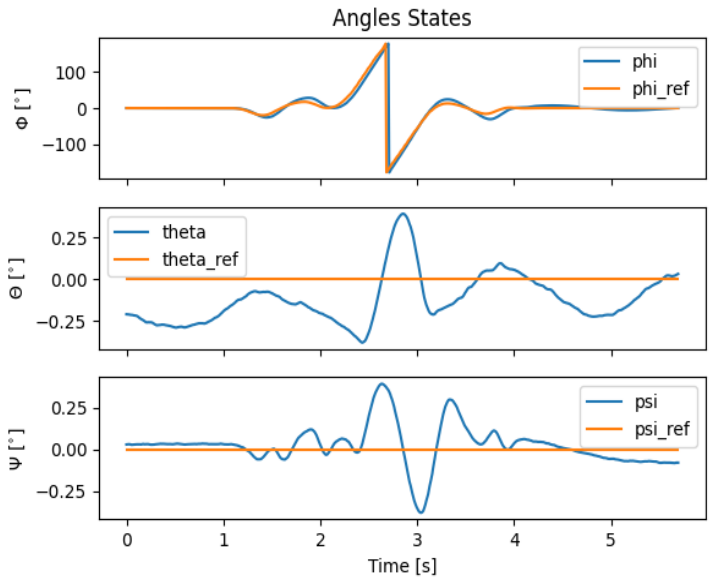
Figure: Diagram showing how the MPC controller node and the quadrotor node communicate with each other

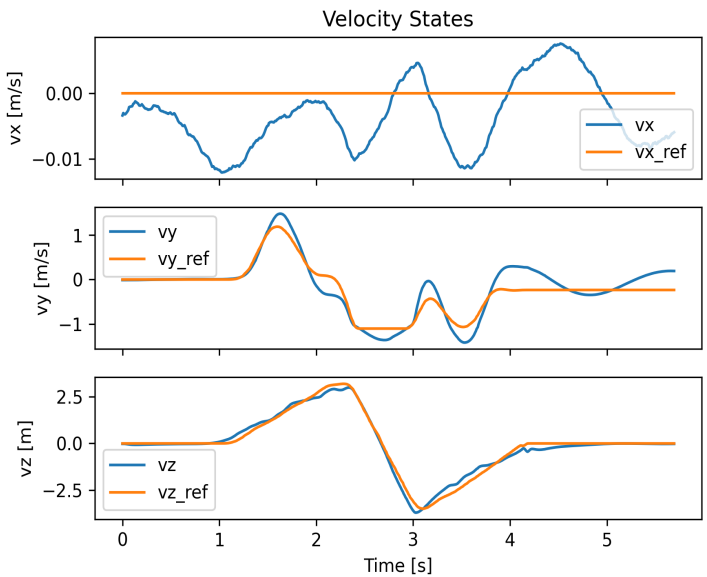
Modified MPC parameters:

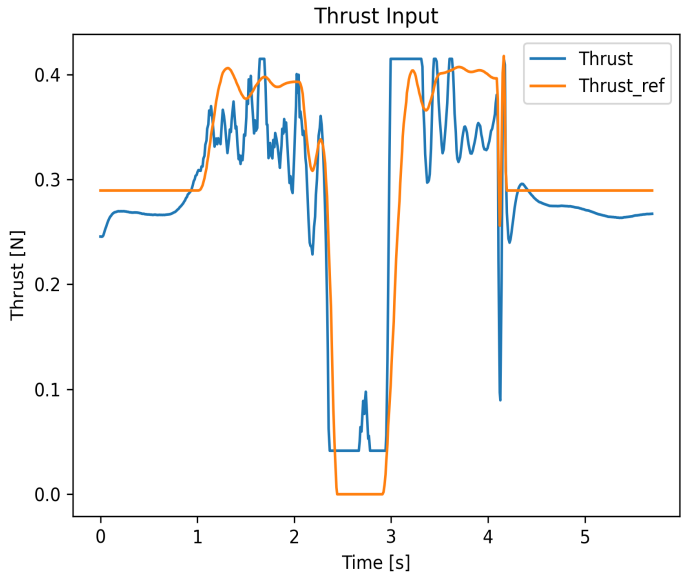
- $N = 50$
- $T_f = 0.5s$

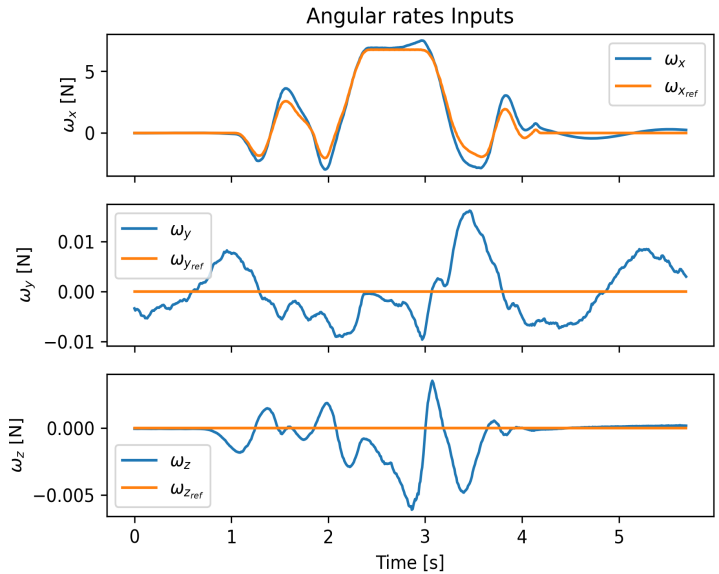

Performed Trajectory
traj
ref_traj
x[m]
y[m]
z[m]

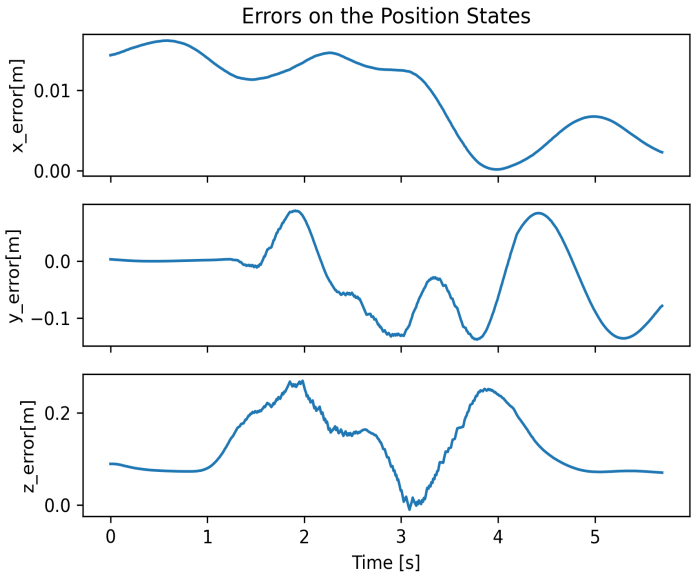

Position States
x
x_ref
y
y_ref
z
z_ref
x [m]
y [m]
z [m]
Time [s]

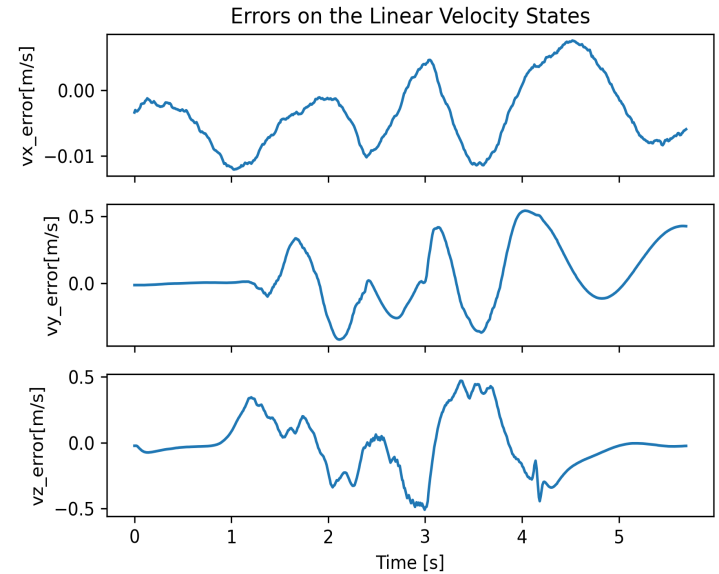

Angles States
phi
phi_ref
theta
theta_ref
psi
psi_ref
Φ [°]
Θ [°]
Ψ [°]
Time [s]


Velocity States
vx
vx_ref
vy
vy_ref
vz
vz_ref
vx [m/s]
vy [m/s]
vz [m]
Time [s]


Thrust Input
Thrust
Thrust_ref
Thrust [N]
Time [s]


Angular rates Inputs
ω_x
$\omega_{x_{ref}}$
ω_x [N]
ω_y
$\omega_{y_{ref}}$
ω_y [N]
ω_z
$\omega_{z_{ref}}$
ω_z [N]
Time [s]


Errors on the Position States
x_error[m]
y_error[m]
z_error[m]
Time [s]


Errors on the Linear Velocity States
vx_error[m/s]
vy_error[m/s]
vz_error[m/s]
Time [s]

Proper Results could not be be obtained:

- The motion capture system was not able to detect the quadrotor properly.
- Neither small nor large markers were able to detect the quadrotor continuously.
- The problem is mainly due to the markers being very close to each other.
- As soon as the detection is lost, the quadrotor would start drifting and crashes.

In conclusion:

- A solution was found for an aggressive flip trajectory (in the ideal case).
- It was demonstrated that MPC is a good control method to consider for tracking aggressive maneuvers.
  - It was able to handle disturbances on the system, and it adjusted the control inputs when is was necessary.
- The goal of performing the flips in constrained environments was not achieved.
- Smaller and more compact flip trajectories could have been generated with the suggested method if the used quadrotor had a larger thrust to weight ratio.

Thank you

Elie Hatem[1], Dr. Sébastien Briot[2], Dr. Isabelle Fantoni[2]

[1]École Centrale de Nantes, Laboratoire des Sciences du Numérique de Nantes (LS2N), Nantes, France
[2]CNRS, Laboratoire des Sciences du Numérique de Nantes (LS2N), Nantes, France