Comparison of Direct collocation and Direct multiple shooting based Non-linear MPC of Crazyflie 2.1

Rean Fernandes
M.Sc. Embedded Systems Engineering
MatrNr. 5250057

***Abstract*—Direct approach methods are a way of solving optimal control problems that discretize the problem in the time domain and solve the subsequent NLP. This report investigates the performance of a non-linear model predictive controller using direct multiple shooting and direct collocation for the Crazyflie 2.1 nano quadcopter. The results are based on the open loop and closed loop performance of the controller, for two tasks; hover and set-point navigation.**

***Index Terms*—Non-linear Model Predictive Control, Direct multiple shooting, Direct Collocation, Quadcopter, Crazyflie 2.1**

I. Introduction

The low level control of a quadcopter is a complex and time sensitive problem, owing to the complex non linear dynamics. Given the system dynamics, the problem of controlling the quadcopter can be formulated as a continuous time problem of the form

$$\begin{aligned} \underset{x(.),u(.)}{\text{minimize}} \quad & \int_0^T L(x(t),u(t))dt + E(x(T)) \\ \text{subject to} \quad & x(0) - x_0 = 0, \quad \text{(initial value)} \\ & \dot{x}(t) - f(x(t),u(t)) = 0, \quad \text{(system dynamics)} \\ & h(x(t),u(t)) \leq 0, \quad \text{(path constraints)} \\ & r(x(T)) \leq 0, \quad \text{(terminal constraints)} \end{aligned} \tag{1}$$

Where x denotes the state of the system, u denotes the control, T is the total time of solution, f denotes the progression of the system with time. L denotes the path cost and E denotes the terminal cost. The initial value constraint defines the initial value problem that we try to solve, the system dynamics constraint ensures that our solution follows the dynamics, the path constraints confine our state space and control space and the terminal constraint enforces the convergence of the system state to a reference.
The solution of this optimal control problem can be relegated to discretising the time domain and parametrising the infinite dimensional decision variables, in this case the control u as will be described in Section IV.

II. Crazyflie 2.1 quadcopter

The Crazyflie 2.1 is a versatile open source flying development platform [1]. It can be used to test and prototype various control related problems for different tasks in the real world. It is a good platform for development, as its small size, while easily perturbed by vibrations compared to larger platforms, ensures robust controller designs.

Property	Value
Mass (m) in Kg	33×10^{-3}
Distance between center and rotor (l) in m	3.275×10^{-4}
Inertia moment around x-axis (I_{xx}) in $Kg \cdot m^2$	1.395×10^{-5}
Inertia moment around y-axis (I_{yy}) in $Kg \cdot m^2$	1.395×10^{-5}
Inertia moment around z-axis (I_{zz}) in $Kg \cdot m^2$	2.173×10^{-5}
Drag coefficient (C_d) in $N \cdot (KRPM)^{-2}$	9.938×10^{-6}
Thrust coefficient (C_t) in $N \cdot (KRPM)^{-2}$	3.25×10^{-4}

TABLE I: Specifications of the Crazyflie 2.1

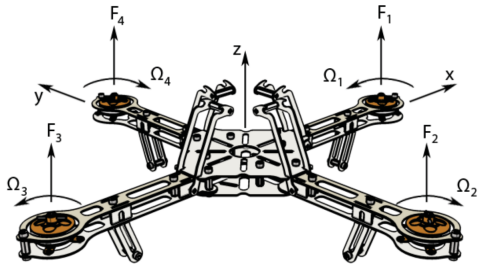


Fig. 1: Example of the quadrotor without propellers attached. Ω_{1-4} denotes the rotational speed of each motor, F_{1-4} is the force generated by each motor and x, y and z is the body fixed coordinate system. [2]

A. Dynamics model

The dynamics of the drone have been modeled with a quaternion implementation of the attitude, for efficient computation. The body frame is considered to be at the center of mass for the drone. The state vector comprises the pose in the world frame, the quaternions, the body frame velocities and body frame rates. It is defined as

$$\mathbf{X} = \begin{bmatrix} p_x & p_y & p_z & q_1 & q_2 & q_3 & q_4 & v_x & v_y & v_z & \Omega_x & \Omega_y & \Omega_z \end{bmatrix} \tag{2}$$

The controls that we wish to find for the drone are the rotational speeds of each of the 4 propellers (kilo-rotations/minute). The control vector is defined as

$$\mathbf{U} = \begin{bmatrix} \Omega_1 & \Omega_2 & \Omega_3 & \Omega_4 \end{bmatrix} \tag{3}$$

With these defined, the dynamics of the drone is now defined as

$$\dot{\mathbf{X}} = \begin{bmatrix} v_x(2q_1^2+2q_2^2-1) - v_y(2q_1q_4-2q_2q_3) + v_z(2q_1q_3+2q_2q_4) \\ v_y(2q_1^2+2q_3^2-1) + v_x(2q_1q_4+2q_2q_3) - v_z(2q_1q_2-2q_3q_4) \\ v_z(2q_1^2+2q_4^2-1) - v_x(2q_1q_3-2q_2q_4) + v_y(2q_1q_2+2q_3q_4) \\ -\frac{1}{2}(q_2\omega_x + q_3\omega_y + q_4\omega_z) \\ \frac{1}{2}(q_1\omega_x - q_4\omega_y + q_3\omega_z) \\ \frac{1}{2}(q_4\omega_x + q_1\omega_y - q_2\omega_z) \\ \frac{1}{2}(q_2\omega_y - q_3\omega_x + q_1\omega_z) \\ v_y\omega_z - v_z\omega_y + g_0(2q_1q_3 - 2q_2q_4) \\ v_z\omega_x - v_x\omega_z - g_0(2q_1q_2 + 2q_3q_4) \\ v_x\omega_y - v_y\omega_x - g_0(2q_1^2+2q_4^2-1) + \frac{C_t}{m}(\Omega_1^2+\Omega_2^2+\Omega_3^2+\Omega_4^2) \\ -\frac{C_t l}{I_{xx}}(\Omega_1^2+\Omega_2^2-\Omega_3^2-\Omega_4^2) + \frac{I_{yy}}{I_{xx}}\omega_y\omega_z - \frac{I_{zz}}{I_{xx}}\omega_y\omega_z \\ -\frac{C_t l}{I_{yy}}(\Omega_1^2-\Omega_2^2-\Omega_3^2+\Omega_4^2) + \frac{I_{xx}}{I_{yy}}\omega_x\omega_z - \frac{I_{zz}}{I_{yy}}\omega_x\omega_z \\ -\frac{C_d}{I_{zz}}(\Omega_1^2-\Omega_2^2+\Omega_3^2-\Omega_4^2) + \frac{I_{xx}}{I_{zz}}\omega_x\omega_y - \frac{I_{yy}}{I_{zz}}\omega_x\omega_y \end{bmatrix} \tag{4}$$

B. Proprietary control schematics of the Crazyflie 2.1

The Crazyflie already comes with internal PID controllers that provide an interface to control the drone at varying levels of abstraction. Using the Python API [3] provided by Bitcraze, it is not possible to control the rotational speed of the propellers, hence the controls obtained by the NMPC as defined in Eq.3 have been converted to send them to the drone. This will be discussed further in Section V.

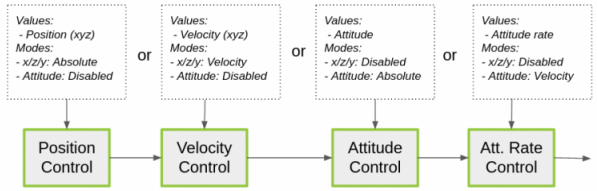


Fig. 2: The onboard PID controllers accept the setpoints as shown above, and use these to control the drone [4]

III. NLP FORMULATION

A. Direct Multiple Shooting

The time interval T as described in Eq. 1 is discretised into N shooting intervals of each of length h such that [5]

$$\begin{aligned} 0 < t_1 < t_2 < ... < t_n = T \\ t_{k+1} = t_k + h, \quad h = \frac{T}{N} \end{aligned} \tag{5}$$

With this done, now we describe the control $u(t)$ and state $x(t)$ as

$$u(t) = q_k \quad \text{for} \quad t \in [t_k, t_{k+1}] \tag{6a}$$

$$x_k(t_k, s_k, q_k) = s_k \quad \text{for} \quad t \in [t_k, t_{k+1}] \tag{6b}$$

$$\dot{x_k}(t, s_k, q_k) = \mathrm{F}(x_k(t, s_k, q_k), q_k) \tag{6c}$$

$q_k \in \mathbb{R}^4$ is a constant value in the interval, $s_k \in \mathbb{R}^{13}$ is the state at the start of each interval, and the function $\mathrm{F} : \mathbb{R}^{12} \times \mathbb{R}^4 \mapsto \mathbb{R}^{13}$ describes the nonlinear dynamics. We now define the discretised cost function as

$$l_k(s_k, q_k) := \int_{t_k}^{t_{k+1}} L(x_k(t_k, q_k, s_k), q_k) \mathrm{d}t \tag{7}$$

The cost function for each interval is now defined as

$$l_k(s_k, q_k) = \|s_k - s_{\text{ref}}\|_Q^2 + \|q_k - q_{\text{hover}}\|_R^2 \tag{8}$$

Where s_{ref} is the required state we want the drone to attain, and q_{hover} is the control which hovers the drone.
The path constraints model the physical aspects of our system and are described as

$$h(s_k, q_k) = \begin{bmatrix} |p_x| - 1 \\ |p_y| - 1 \\ |p_z - 0.5| - 0.5 \\ |v_x| - 3 \\ |v_y| - 3 \\ |v_z| - 3 \\ |\omega_x| - 0.5 \\ |\omega_y| - 0.5 \\ |\omega_z| - 0.5 \\ |\Omega_1 - 11000| - 11000 \\ |\Omega_2 - 11000| - 11000 \\ |\Omega_3 - 11000| - 11000 \\ |\Omega_4 - 11000| - 11000 \end{bmatrix} \tag{9}$$

The constraints on pose confine our drone to a box with a positive height(as it doesn't make sense for the drone to go below the ground) whose bottom face center is the world frame origin. The velocity constraint confines the drone to have a speed less than 3 ms^{-1}. The constraint on attitude rate makes it so that the attitude of the drone doesn't change too fast for the drone to reach an unrecoverable orientation. The constraint on the value of Ω confines the rpm of the drone to its maximum value of 22000 $\mathrm{rotations/minute}$. Finally, our NLP for direct multiple shooting is now defined as

$$\begin{aligned} \underset{\substack{s \in \mathbb{R}^{13(N+1)} \\ q \in \mathbb{R}^{4N}}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} l_k(s_k, q_k) + \|s_N - s_{\text{ref}}\|_{Q_\mathrm{N}}^2 \\ \text{subject to} \quad & x(0) - s_0 = 0 \\ & s_{k+1} - \mathrm{F}(s_k, q_k) = 0 \\ & h(s_k, q_k) \leq 0 \end{aligned} \tag{10}$$

Where Q_N Is the terminal cost weight, to give more importance to the end state of the drone. The dynamics are integrated using an explicit Runge- Kutta (RK4) scheme.

B. Direct Collocation

The direct collocation method is based on the numerical simulation method of Orthogonal collocation, which is a variant of implicit RK methods. We use here the same discretisation for the time grid as described in Eq. 5. The solution of the state at each interval is approximated by a polynomial $p(t, v_k) \in \mathbb{R}^{13}$ of order d, defined as

$$p_k(t, v_k) = \sum_{i=0}^{d} v_{k,i} l_{k,i}(t), \quad l_{k,i}(t) = \prod_{j=0, j\neq i}^{d} \frac{t - t_{k,j}}{t_{k,i} - t_{k,j}} \in \mathbb{R} \tag{11}$$

where $v_{k,i} \in \mathbb{R}^{13}$ is a subset of the polynomial coefficients $v_k \in \mathbb{R}^{13 \cdot (d+1)}$ and the collocation times are chosen using the

Gauss-Legendre collocation scheme. The discretisation of the controls and states is done as in Eq. 6a and Eq.6b respectively. The collocation-based integration of the state dynamics on the interval $[t_k, t_{k+1}]$ is done solving equation of the vector of equations $c_k(v_k, s_k, q_k)$ defined as in [5]

$$c_k(v_k, s_k, q_k) = \begin{bmatrix} v_{k,0} - s_k \\ \dot{p}_k(t_{k,1}, v_k) - \mathrm{F}(v_{k,1}, t_{k,1}, q_k) \\ . \\ . \\ \dot{p}_k(t_{k,d}, v_k) - \mathrm{F}(v_{k,d}, t_{k,d}, q_k) \end{bmatrix} = 0 \quad (12)$$

where F defines the system dynamics as in Eq. 6c, and Eq. 12 is solved over the variables $v_{k,i} \in \mathbb{R}^{13}$ with $i = 0, ..., d$. In addition to solving these equations, we also have to enforce the continuity condition to ensure that the state at the end of one collocation interval matches the starting state at the next collocation interval, which is done as

$$p_k(t_{k+1}, v_k) = s_{k+1}, \quad k = 0, ..., N \quad (13)$$

Now we approximate the continuous cost function by Gaussian quadrature on the collocation intervals as

$$l_k(s_k, q_k, v_k) := \int_{t_k}^{t_{k+1}} L(s_k, q_k, v_k)\mathrm{d}t \quad (14)$$

giving us the cost function for each interval as the sum of the approximation evaluated at each collocation interval

$$l_k(s_k, q_k, v_k) = \sum_{i=0}^{d} h \cdot B_i \cdot \mathrm{cost}(v_{k,i}, q_k) + \mathrm{cost}(s_k, q_k) \quad (15)$$

where B_i is the quadrature function evaluated at the i^{th} collocation interval and,

$$\mathrm{cost}(s_k, q_k) = \|s_k - s_{\mathrm{ref}}\|_Q^2 + \|q_k - q_{\mathrm{hover}}\|_R^2 \quad (16)$$

The path constraints are defined just as in Eq. 9, with the only difference being that they are also evaluated at the collocation nodes too. Our formulated NLP for direct collocation is now

$$\begin{aligned} \underset{\substack{v \in \mathbb{R}^{N(13(d+1))} \\ s \in \mathbb{R}^{13(N+1)} \\ q \in \mathbb{R}^{4(N)}}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} l_k(s_k, q_k, v_k) + \|s_N - s_{\mathrm{ref}}\|_{Q_{\mathrm{N}}}^2 \\ \text{subject to} \quad & x(0) - s_0 = 0 \\ & c_k(s_k, q_k, v_k) = 0, \quad k = 0, .., N \\ & p_k(t_{k+1}) - s_{k+1} = 0, \quad k = 0, .., N \\ & h(s_k, q_k, v_k) \leq 0, \quad k = 0, .., N \end{aligned} \quad (17)$$

IV. NMPC IMPLEMENTATION

A. NMPC

The implementation of the NMPC is done by solving the formulated NLP iteratively for each timestep, and sending the first control from the optimal solution over the entire horizon to the drone. The algorithm is as follows

Algorithm 1 Online NMPC for navigation to trajectory

Require: Reference state x_{ref}, Tolerance value ϵ, Max iterations $\mathrm{maxiter}$, current deviation from reference δ_x
Ensure: $\delta_x \leq \epsilon$

```
Create the NLP solver
x_current ← initial state of the drone
Initialise current guess for control and state
iter ← 0
while deviation between x_current and x_ref i.e. δ_x ≥ ε and iter < maxiter do
    Solve NLP with x_current as initial state
    u_current ← first optimal control u*_opt(0) from the solution
    Send u_current to the drone
    Wait for the drone to execute u_current and update x_current
    Update initial guess for the next solver iteration with u_current and x_current
    Update value of δ_x for current state
    if δ_x ≤ ε then
        break
    end if
    iter ← iter + 1
end while
return Drone has reached reference
```

The NLP is solved using the casADi [6] framework in Python.

V. CONTROL OF THE DRONE USING THE PYTHON API

Updating the angular speed of the drone as described in the NLP formulation requires a very short computation time owing to the nonlinear dynamics, which can be difficult to implement. The API provides a command to send a new control in the form of setpoints for the roll, pitch, yaw rate, and thrust, specifying the next value to be achieved by the drone. The implementation of this method is as follows:

Algorithm 2 Control Drone with NMPC

```
procedure CONTROLTODRONE(x_current, u_current obtained from solver)
    Extract quaternion values from x_current
    Convert quaternion to yaw, pitch, and roll in degrees
    Extract rotational speeds from u_current
    Convert rotational speeds to thrust value
    Extract yaw rate from x_current
    Concatenate roll, pitch, yaw rate, and thrust into the setpoint vector
    Send setpoint to the drone using API function
end procedure
```

VI. Results

For both the methods, different values for the state weights where used. For direct multiple shooting the value of Q was:

$$Q_{\text{DMS}} = \text{diag}\left[120, 100, 100, 10^{-2}, 10^{-2}, 10^{-2}, 10^{-2}, 0.07, 4, 1, 0.1, 0.1, 0.1\right] \tag{18}$$

and for direct collocation

$$Q_{\text{DMS}} = \text{diag}\left[120, 100, 100, 10^{-2}, 10^{-2}, 10^{-2}, 10^{-2}, 0.07, 4, 1, 1, 1, 1\right] \tag{19}$$

The reason for higher weighting of the attitude rates for direct collocation was that the simulation tended to make the drone circle around the reference point rather than reach it, so the attitude rates were penalized to prevent its oscillations. The control weight R was defined as

$$R = \text{diag}\left[10^{-2}, 10^{-2}, 10^{-2}, 10^{-2}, 10^{-2}\right] \tag{20}$$

This was used for both the methods. The terminal weight was simply the $Q_N = 50 \cdot Q$

A. Open Loop performance

The open loop performance of the direct methods was tested by checking the time taken for solution of the NLP for different horizon lengths. While for smaller horizon lengths, there is a signficant decrease in the solution time, the deviation from the final reference does not converge, indicating the need for a bigger timestep for the integration. As indicated in Fig. **??**, both solver have almost the same solution time for a timestep $h = 0.01$s. However Fig.**??** shows that for the same horizon length and timestep, Direct collocation leads to a higher deviation from the reference state at the end of the open loop simulation with a slower convergence rate. From fig. **??**, the horizon length N was taken to be 20 for the rest of the simulation owing to the low solution time. With N fixed, the timestep h was varied and the solution times for both the methods has been plotted in Fig. **??**. From the plot the inference was that a timestep of $h = 0.05$s was suited for direct collocation, and a timestep of $h = 0.01$s was suited for direct multiple shooting.

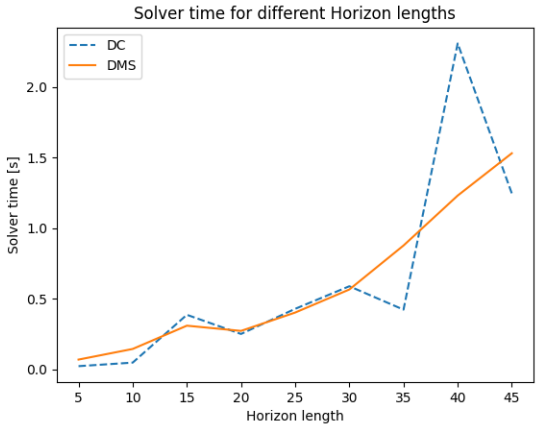


Fig. 3: Plot of horizon length N vs time taken to solve the NLP

Method	Timestep	Min Deviation	Method	Degree
DC	0.05s	0.05m	Collocation	2
DMS	0.05s	0.05m	Explicit RK4	3

TABLE II: Specifications of the methods

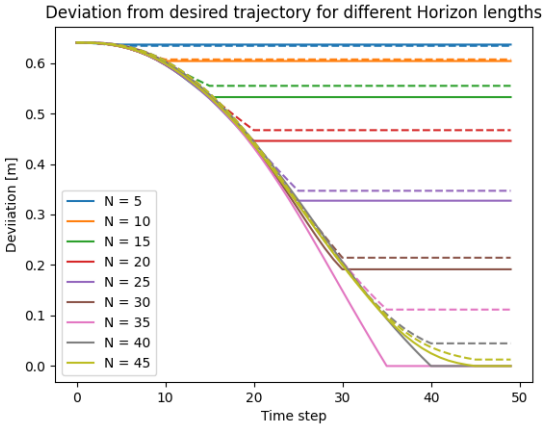


Fig. 4: Plot of the Deviation from the reference state for different horizon lengths. The straight line indicates that the NLP solved the problem only till that timestep and there is no deviation data after that timestep

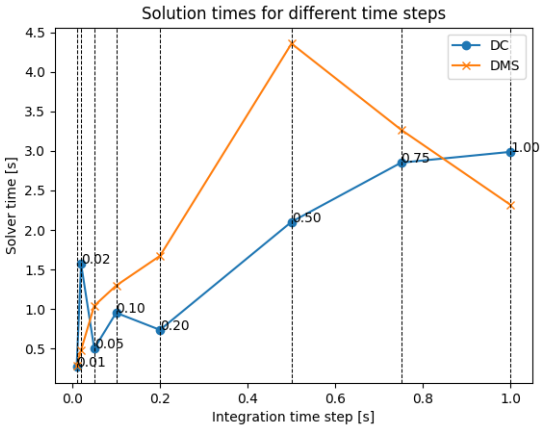
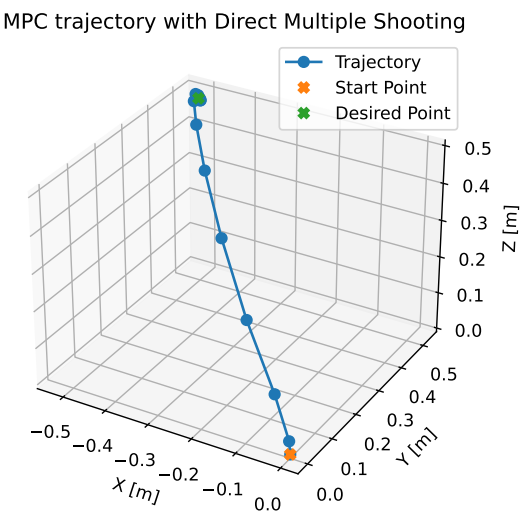


Fig. 5: Plot of the timestep vs the time taken to solve the NLP

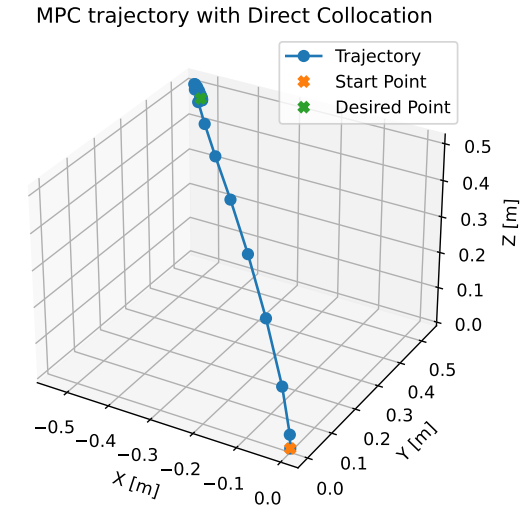
B. MPC simulation performance

The simulation was run considering the drone to start from $[0, 0, 0]$, and navigating it to a setpoint $x_{\text{ref}} = [-0.5, 0.5, 0.5]$. The plot of the trajectory is show in Fig. 6. The average solution time for direct multiple shooting was 75.80ms, while for direct collocation it was 31.39ms. The trajectory obtained obtained using direct collocation seems smoother than that for multiple shooting.

Fig.7 shows the deviation from the reference state. The direct collocation method converges long before the multiple shooting method for a shorter solution time.

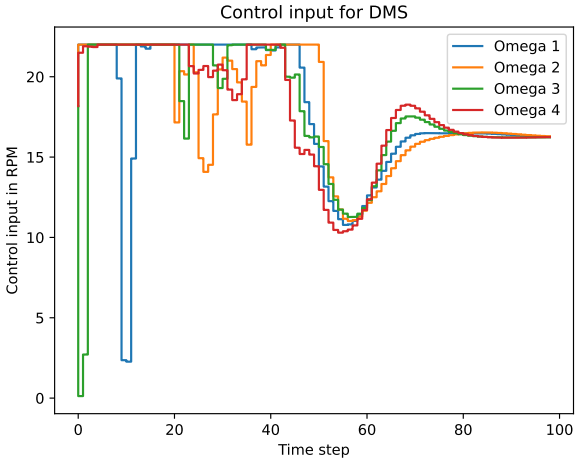


(a) Trajectory for Direct multiple shooting

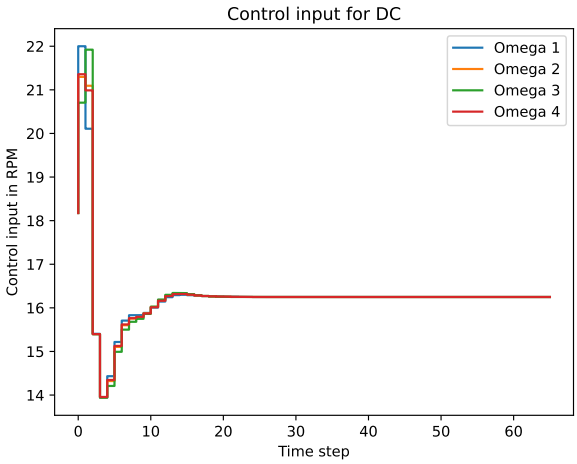


(b) Trajectory for Direct Collocation

Fig. 6: MPC trajectory



(a) Plot of MPC output for DMS



(b) Plot of MPC output for DC

Fig. 8: Plot of control output of MPC for the two methods.

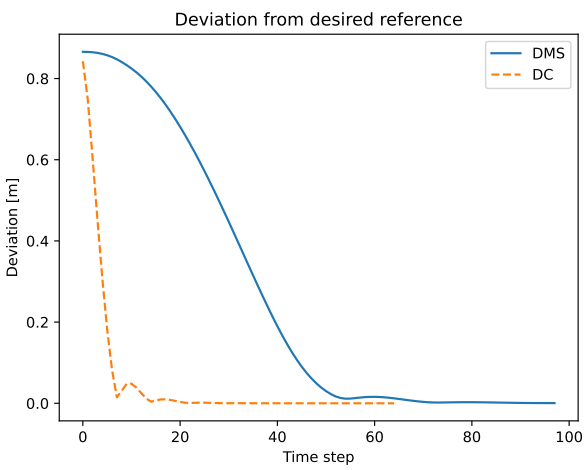
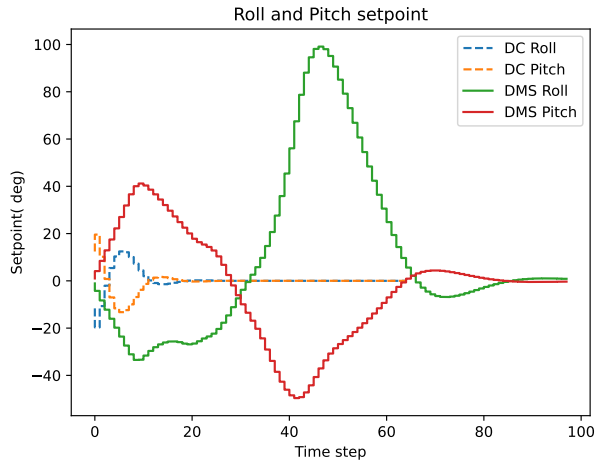


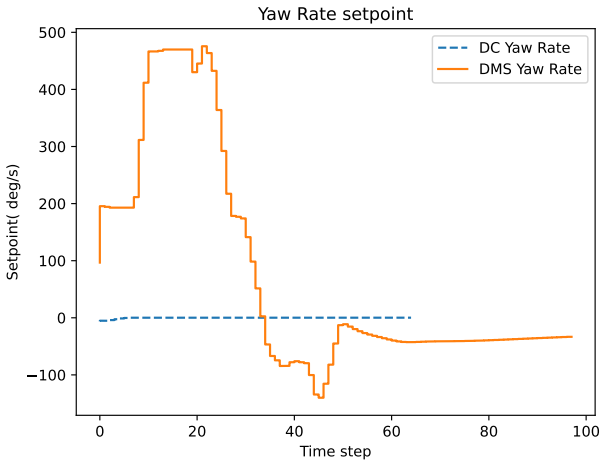
Fig. 7: Progress of deviation from the reference state

1) Output of NMPC: Fig 8a and Fig. 8b show the progression of the optimal control output of the MPC. There is a large amount of variation in the control for DMS, whereas the DC method tends to reach extremes but converge faster. This could be due to the lower accuracy in the approximation of the system dynamics by the explicit RK4 integrator for the DMS method. The DC method also converges in about 15 steps, contrasting the DMS method that takes 84 steps.

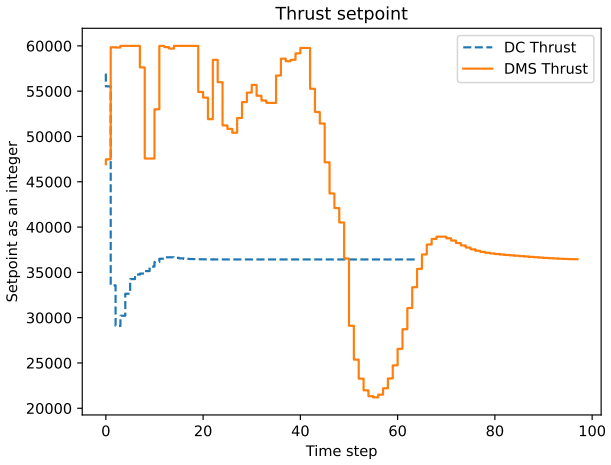
2) Control output to drone: The graph of the controls sent to the drone as show in Fig. 9 indicates that for the DC method, the MPC initially changes the attitude of the drone and accelerates it towards the setpoint. The DMS method adjusts the attitude over the duration of the total time, and the DMS MPC makes around step 50 reduces the thrust setpoint, in anticipation of reaching the reference.



(a) Plot of the actual roll and pitch setpoints sent to the drone



(b) Plot of the yaw rate setpoints sent to the drone



(c) Plot of the Thrust setpoints sent to the drone

Fig. 9: Comparison of the Controls sent to drone for the two methods.

VII. Discussion

While the simulation data for the Direct multiple shooting looks slower, I believe that it does a better job of controlling the drone. The direct collocation method seems too good to be true. In theory it should take a longer time for the same parameters as direct multiple shooting, owing to the formulation of the collocation polynomials for every time interval. The incredible difference in the speed up of direct collocation was only obtained once the time for testing in the lab was up. Unfortunately, Online MPC was not implemented on the drone owing to the large solution time per step. This could possibly be a limitation of using casADi with python as it is a general purpose IPOPT solver. Another possible issue could also be the coding of the NLP formulation, which could possibly made more efficient. Running Just-in-time compilation tends to decrease the computation time by a factor of 2, although the compilation time for this exceeds a few minutes, making it an unfavourable compromise.

VIII. Future scope

To implement an NMPC algorithm on the drone, future work would entail formulating this OCP as an SQP and using a fast embedded solver such as Acados.

IX. Acknowledgment

I extend my heartfelt gratitude to Prof. Dr. Moritz Diehl, without whose courses and material this project would not be possible. I also thank PhD advisors Mr. Andrea Ghezzi, Mr. Jakob Harzer and Mr. Florian Messerer for their help and advise. I would like to thank Mr Mohammed Hababeh for his dedication and enthusiasm as our tutor for this project. Finally I would like to thank Mr. Minh Quang Nguyen for our discussions during the early days of the project.

References

[1] Bitcraze, "Crazyflie 2.1 - bitcraze." https://www.bitcraze.io/products/crazyflie-2-1/, accessed 2023-03-05.

[2] E. Fresk and G. Nikolakopoulos, "Full quaternion based attitude control for a quadrotor," in *2013 European Control Conference (ECC)*, pp. 3864–3869, 2013.

[3] Bitcraze, "crazyflie-lib-python." https://github.com/bitcraze/crazyflie-lib-python#platform-notes, accessed 2023-03-05.

[4] Bitcraze, "Commander setpoints - bitcraze." https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/commanders_setpoints/, accessed 2023-03-05.

[5] S. G. Moritz Diehl, *Lecture Notes on Numerical Optimal Control*. November 2017.

[6] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.