Comparison of Direct Methods for NMPC Applied to a Thrust Vector Drone

Izzy Mones and Heidi Dixon

September 14, 2025

Introduction

1. What is a Thrust Vector Drone and Why they are useful for Rocket Design

Thrust Vector Drone Equations of Motion

The system state $\vec{x}$ includes translational position $\vec{p} = [x, y, z]$ and velocity $\vec{v} = [v_x, v_y, v_z]$ in the world frame, the attitude represented as a unit quaternion $\vec{q} = [q_x, q_y, q_z, q_w]$ and the angular velocity in the body frame $\vec{\omega} = [\omega_x, \omega_y, \omega_z]$. The control variables include the two gimbal angles θ_1, θ_2, the average thrust between the two propellers $\overline{P}$ and the differential thrust between the propellers P_Δ to control the rotation about the body z axis.

$$\vec{x} = \begin{bmatrix} x & y & z & v_x & v_y & v_z & q_x & q_y & q_z & q_w & \omega_x & \omega_y & \omega_z \end{bmatrix}^T$$
$$\vec{u} = \begin{bmatrix} \theta_1 & \theta_2 & \overline{P} & P_\Delta \end{bmatrix}^T$$

The equations of motion are represented as a series of first-order vector differential equations:

$$\dot{\vec{p}} = \vec{v} \tag{1}$$
$$\dot{\vec{v}} = \frac{1}{m} R(\vec{q}) \vec{F_b} + \vec{g} \tag{2}$$
$$\dot{\vec{q}} = \frac{1}{2} Q(\vec{\omega}) \vec{q} \tag{3}$$
$$\dot{\vec{\omega}} = I^{-1} \Big(\vec{M_b} - \vec{\omega} \times (I\, \vec{\omega}) \Big) \tag{4}$$

where m is the mass of the drone and $\vec{g} = [0, 0, -9.81]^T$ is the acceleration due to gravity.

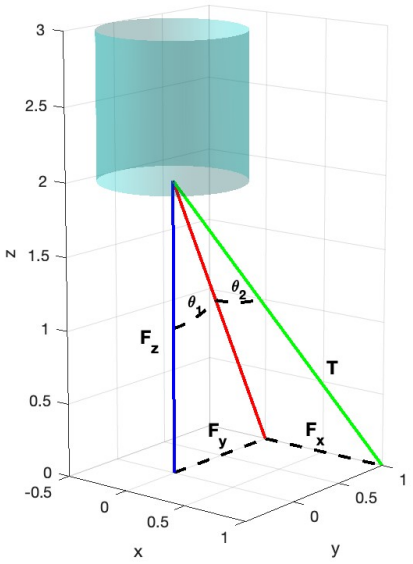
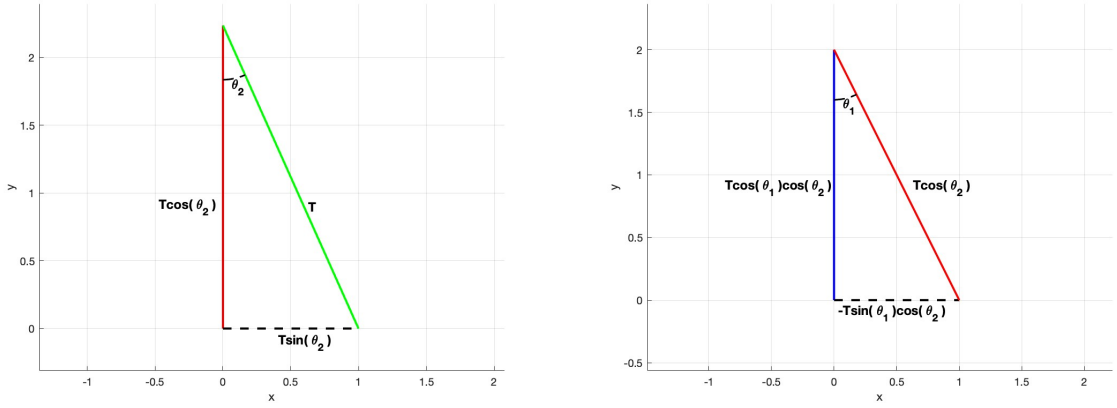


Figure 1: Three-dimensional thrust decomposition for a 2-axis gimballed drone.



(a) Thrust decomposition in the inner gimbal axis plane.

(b) Thrust decomposition in the x–z plane.

Figure 2: Two-dimensional thrust decompositions for a 2-axis gimballed drone.

Using vector decomposition we can find the thrust vector in body coordinates in terms of the gimbal angles θ_1 and θ_2 and magnitude of the thrust T acting on the drone.

$$\vec{F_b} = T \begin{bmatrix} \sin\theta_2 \\ -\sin\theta_1 \cos\theta_2 \\ \cos\theta_1 \cos\theta_2 \end{bmatrix}$$

$\vec{M}_z$ is the moment about the body z axis due to differential thrust.

$$\vec{M}_z = \begin{bmatrix} 0 \\ 0 \\ M_z \end{bmatrix}$$

The body frame moment vector is the sum of the torque due to thrust vector acting through the moment arm and the torque generated by the differential thrust. l is the length of the moment arm acting from the center of mass to the point $\vec{F}_b$ acts on the drone.

$$\vec{l} = \begin{bmatrix} 0 \\ 0 \\ -l \end{bmatrix}$$

$$\vec{M}_b = \vec{l} \times \vec{F}_b + \vec{M}_z = \begin{bmatrix} -lT \sin\theta_1 \cos\theta_2 \\ -lT \sin\theta_2 \\ M_z \end{bmatrix}$$

The drone is designed to be symmetrical and the principal axes align with the body frame. The products of inertia vanish and inertia tensor can be written with only diagonal terms.

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

The body to world matrix is used to compute the force in world coordinates due to the body-centric thrust vector given the quaternion orientation.

$$R(\vec{q}) = \begin{bmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_w q_z) & 2(q_x q_z + q_w q_y) \\ 2(q_x q_y + q_w q_z) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z - q_w q_x) \\ 2(q_x q_z - q_w q_y) & 2(q_y q_z + q_w q_x) & 1 - 2(q_x^2 + q_y^2) \end{bmatrix}$$

The quaternion propagation matrix computes the change in quaternions given a set of angular velocities.

$$Q(\vec{\omega}) = \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix}$$

The final extended dynamic equations are expressed as:

$$\dot{x} = v_x$$

$$\dot{y} = v_y$$

$$\dot{z} = v_z$$

$$\dot{v_x} = \frac{\sin\theta_2 * T(1 - 2(q_y^2 + q_z^2)) - \sin\theta_1 \cos\theta_2 * 2T(q_x q_y - q_w q_z) + \cos\theta_1 \cos\theta_2 * 2T(q_x q_z + q_w q_y)}{m}$$

$$\dot{v_y} = \frac{\sin\theta_2 * 2T(q_x q_y + q_w q_z) - \sin\theta_1 \cos\theta_2 * T(1 - 2(q_x^2 + q_z^2)) + \cos\theta_1 \cos\theta_2 * 2T(q_y q_z - q_w q_x)}{m} + g$$

$$\dot{v_z} = \frac{\sin\theta_2 * 2T(q_x q_z - q_w q_y - \sin\theta_1 \cos\theta_2 * 2T(q_y q_z + q_w q_x) + \cos\theta_1 \cos\theta_2 * T(1 - 2(q_x^2 + q_y^2))}{m}$$

$$\dot{q_x} = \frac{\omega_z q_y - \omega_y q_z + \omega_x q_w}{2}$$

$$\dot{q_y} = \frac{-\omega_z q_x + \omega_x q_z + \omega_y q_w}{2}$$

$$\dot{q_z} = \frac{\omega_y q_x - \omega_x q_y + \omega_z q_w}{2}$$

$$\dot{q_w} = \frac{-\omega_x q_x - \omega_y q_y - \omega_z q_z}{2}$$

$$\dot{\omega_x} = \frac{-lT \sin\theta_1 \cos\theta_2 - \omega_y \omega_z I_{zz} + \omega_y \omega_z I_{yy}}{I_{xx}}$$

$$\dot{\omega_y} = \frac{-lT \sin\theta_2 - \omega_x \omega_z I_{xx} + \omega_x \omega_z I_{zz}}{I_{yy}}$$

$$\dot{\omega_z} = \frac{M_z - \omega_x \omega_y I_{yy} + \omega_x \omega_y I_{xx}}{I_{zz}}$$

Note that these equations use the variable T representing the magnitude of the thrust vector acting on the drone. The actual control variables are defined as the average $\overline{P}$ and difference P_Δ between the two propeller thrust values T_1 and T_2.

$$\overline{P} = \frac{T_1 + T_2}{2}$$

$$P_\Delta = T_1 - T_2$$

T_1 and T_2 denote the normalized brushless motor command signals, ranging from 0 (no throttle) to 1 (full throttle).

$$T_1 = \overline{P} + P_\Delta/2$$

$$T_2 = \overline{P} - P_\Delta/2$$

To map the average thrust command signal value to the actual thrust force value in Newtons used in the thrust-vector decomposition a second-order polynomial fit is experimentally determined. Similarly, a linear relationship is used to map the differential thrust to the moment about the body frame z axis.

$$T = a\overline{P}^2 + b\overline{P} + c$$

$$M_z = dI_{zz}P_\Delta$$

NMPC

Model predictive control (MPC) is a control algorithm that solves a linear optimization problem at each time step. The set of instance constraints can be used to enforce the system dynamics and restrict the values of control inputs or state variables. For example, MPC constraints can be used to eliminate solutions that violate minimum or maximum control values or require control values to change faster than mechanically possible. This gives MPC algorithms more control over the types of solutions produced than linear quadratic regulators (LQR) or proportional integral derivative (PID) controllers. Unlike LQR and PID methods which are designed to find an optimal control input for a given moment in time, an MPC solution considers a series of control inputs over a finite time horizon. This allows MPC solutions to create a *plan* of actions that can anticipate and adjust to future issues. For example, the optimal speed for a car at a given instance might change if the algorithm knew that it was entering a curve in the road in the next few time steps. The ability to anticipate future conditions allows MPC to construct more robust solutions. The control plan is rebuilt at each timestep allowing the method to react to disturbances and noise. Nonlinear model predictive control (NMPC) is a variant of MPC that allows nonlinear constraints.

The biggest advantage of using NMPC to control our thrust vector drone is the allowance of nonlinear constraints. LQR and standard MPC methods require linear system dynamics. Equations (1-4) are strongly nonlinear, but can be linearized around the fixed point in which the drone is vertically balanced. However,

the linearized equations quickly become inaccurate if the drone moves away from the fixed point by tilting or rotating. Working directly with the nonlinear equations allows more accurate predictions of the drone behavior. Like MPC, NMPC allows us to enforce the mechanical limitations of our thrust motors and gimbal servos. An NMPC instance for the thrust vector drone is a minimization problem of the form

$$\min_{u(t)} \int_{t_0}^{t_f} \ell(x,u,t)\,dt \;+\; \phi(x_f) \tag{5}$$

$$\text{s.t. } \dot{\vec{x}} = f(\vec{x},\vec{u}) \tag{6}$$

$$-\theta_{max} \le \theta_1 \le \theta_{max} \tag{7}$$

$$-\theta_{max} \le \theta_2 \le \theta_{max} \tag{8}$$

$$T_{min} \le \overline{P} + P_\Delta/2 \le T_{max} \tag{9}$$

$$T_{min} \le \overline{P} - P_\Delta/2 \le T_{max} \tag{10}$$

$$0 \le z. \tag{11}$$

The cost function (5) minimizes both the squared error between the current state $x(t)$ and the goal state x_{ref} weighted by diagonal matrix Q and the error between the current control $u(t)$ and the goal control u_{ref} weighted by diagonal matrix R.

$$\ell(x,u,t) = (x(t)-x_{ref})^T Q(x(t)-x_{ref}) + (u(t)-u_{ref})^T R(u(t)-u_{ref}) \tag{12}$$

We also add a termial cost

$$\phi(x_f) = (x(t_f)-x_{ref})^T Q_f(x(t_f)-x_{ref}) \tag{13}$$

The constraint (6) enforces the equations of motion which in our case are differential equations defined in (1-4). The problem instance also enforces minimum and maximum gimbal angles (7,8), minimum and maximum values for each thrust motor (9, 10) and restrictons on the drone's vertical height (11).

To solve the NMPC instance at each time step, it is formulated as a nonlinear programming problem (NLP) and solved by a standard NLP solver. NLP solvers are well established, highly optimized methods that originated in the field of operations research. Unfortunately, despite the availability of efficient solvers for these problem, NLP solutions for NMPC problem instances may be too slow to produce control inputs for a thrust vector drone which requires a fast response time. This is a prime drawback of MPC and NMPC algorithms over LQR and PID methods which can be computed very quickly.

Ideally, we'd like to run our control algorithm at 50Hz on a Raspberry Pi 5. To meet this timing restriction we need to build an NMPC instance that can be solved efficiently. Given an NMPC problem instance, there are a variety of known ways to formulate the instance as nonlinear programming problem. Next, we present three different ways to formulate this instance as a nonlinear programming problem and then compare the quality of solutions produced and their relative computational efficiency.

NLP Formulations

To use NMPC the problem instance needs to be parameterized over the control trajectory, the differential equations for the system dynamics need to be integrated, and then the resulting problem can be optimized over control parameters. There are different ways to parameterize the problem and integrate it. These choices will affect the size of the problem and the speed at which it can be solved. All these methods are using numerical methods to estimate the behaviour of differential equations. The accuracy of the estimation will vary depending on the method used and the coarseness of the parameterization. Typically there is a tradeoff between speed and accuracy.

Orthogonal Collocation using do-mpc

Orthogonal collocation is a local method that divides the time horizon into finite elements and fits polynomials to each interval. The dynamics are enforced throughout each interval at a set of Radau nodes given by:

$$\gamma_0 = 1, \qquad \gamma_j = \gamma_{j-1}\frac{(K-j+1)(K+j+\alpha+\beta)}{j(j+\beta)}, \qquad j = 1,\dots,K$$

When P_K are the Gauss-Jacobi polynomials:

$$P_K^{(\alpha,\beta)}(\tau) = \sum_{j=0}^{K}(-1)^{K-j}\gamma_j\tau^j$$

Lagrange polynomials, $\ell_j(\tau)$, are used to interpolate the state trajectory $z^K(\tau)$:

$$\ell_j(\tau) = \prod_{\substack{m=0\\m\neq j}}^{K}\frac{\tau-\tau_m}{\tau_j-\tau_m}$$

$$z^K(\tau) = \sum_{j=0}^{K} z_{ij}\ell_j(\tau)$$

By differentiating the state trajectory and equating it with the dynamics ODEs, f, a dynamics constraint is formed:

$$\sum_{j=0}^{K} z_{ij}\frac{d\ell_j(\tau_k)}{d\tau} = h_i f(z_{ik}, t_{ik}), \qquad k = 1,\dots,K.$$

Continuity constraints are used to fit an interval polynomial's endpoint with the first point of the next interval. There are two additional continuity constraints for enforcing the overall horizon's start and endpoint continuity.

$$z_{i+1,0} = \sum_{j=0}^{K}\ell_j(1)z^{ij}$$

$$z_f = \sum_{j=0}^{K}\ell_j(1)z^{Nj}, \qquad z_{1,0} = z_0$$

The cost function is defined as:

$$\min_{i(\cdot)} J = \Phi(z(t_f), t_f) + \int_{t_0}^{t_f} L(z(t), u(t), t)dt$$

$$J = \Phi(z(t_f)) + \sum_{i=1}^{N} h_i \sum_{k=1}^{K}\omega_k L(z_{ik}, u_{ik}, t_{ik})$$

Where the weights ω_k are determined by Radau quadrature, which integrates all polynomials up to $2s-1$. For a horizon of 100 time steps each with an interpolant polynomial order of two the total number of decision variables per time step is $13*(100*(2+1)+1)+4*100 = 4313$.

Multiple Shooting (with Runge–Kutta discretization)

For multiple shooting, we divide our time horizon into N evenly spaced nodes $\{t_0, t_1, \ldots, t_N\}$ of length Δt. The initial state is constrained and at each node we make a copy of our state x_k and control u_k variables. Within each time step $[t_k, t_{k+1}]$ we forward integrate our differential equations f using Runge-Kutta to create a constraint that enforces the change of state between time steps with the system dynamics.

$$\begin{aligned} x_0 &= X_0 \\ k_1 &= f(x_k, u_k, t_k), \\ k_2 &= f\big(x_k + \tfrac{\Delta t}{2} k_1,\, u_k,\, t_k + \tfrac{\Delta t}{2}\big)\,, \\ k_3 &= f\big(x_k + \tfrac{\Delta t}{2} k_2,\, u_k,\, t_k + \tfrac{\Delta t}{2}\big)\,, \\ k_4 &= f(x_k + \Delta t\, k_3,\, u_k,\, t_k + \Delta t)\,, \\ x_{k+1} - \Big(x_k + \tfrac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)\Big) &= 0, \qquad k = 0, \ldots, N-1. \end{aligned} \tag{14}$$

The cost function is defined to penalize deviation from reference state as well as deviation from reference control input. There is also a terminal cost that incentivizes long term stability by weighting the final state more heavily.

$$\vec{x}_r = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}^T, \qquad \vec{u}_r = \begin{bmatrix} 0 & 0 & T_{\text{hover}} & 0 \end{bmatrix}^T.$$

$$\sum_{k=0}^{N-1} \Big[(x_k - x_r)^T Q (x_k - x_r) + (u_k - u_r)^T R (u_k - u_r) \Big] + (x_N - x_r)^T Q_f (x_N - x_r) \tag{15}$$

Multiple shooting considers the state and control variables at every time step to be optimization variables so the full NLP vector is:

$$\begin{bmatrix} x_0 & u_0 & x_1 & u_1 & \ldots & x_{N-1} & u_{N-1} & x_N \end{bmatrix}$$

This can easily become computationally demanding for long horizons and high-dimensional systems. For a horizon length of 100 time steps there are $13 * (100 + 1) + 4 * 100 = 1713$ decision variables per time step.

Chebyshev Pseudospectral Method

Multiple shooting is useful for its robustness to discontinuities and nonlinearity and its conceptual simplicity. However, thrust-vectoring dynamics are smooth and continuous, so many of the benefits of multiple shooting are unnecessary. Additionally, multiple shooting introduces a much larger NLP because it solves for the state and control variables at every point in the control horizon. By contrast, Chebyshev Pseudospectral Collocation is a global method that represents the trajectory of each state with a single interpolating polynomial of degree N over the entire horizon, defined by its values at a set of $N + 1$ nodes. It achieves spectral accuracy, with error decreasing exponentially as the number of nodes increases, while requiring far fewer decision variables than multiple shooting. For $N = 6$ there are seven nodes and $7 * 13 + 7 * 4 = 119$ decision variables per time step. The dynamics are enforced by constraining the derivative of the interpolant at each node to the dynamic equations. Chebyshev–Gauss–Lobatto nodes are used to cluster more of the nodes towards the beginning and end of the horizon, mitigating the Runge Phenomenon, where polynomial interpolation develops large oscillations near the endpoints. The cosine distribution ensures accuracy and stability:

$$\tau_j = \cos\left(\frac{j\pi}{N}\right), \qquad j = 0, 1, \ldots, N$$

These node values are defined from 1 to -1 instead of from time 0 to the end of the horizon. To determine the horizon time for a given tau a linear mapping is used:

$$t(\tau) = \frac{t_f - t_0}{2}\,\tau + \frac{t_f + t_0}{2}, \qquad \frac{dt}{d\tau} = \frac{t_f - t_0}{2}.$$

$$\frac{dx}{dt} = \frac{d\tau}{dt}\frac{dx}{d\tau} = \frac{2}{t_f - t_0}\frac{dx}{d\tau}$$

At each time step the dynamics are checked with:

$$\frac{dx}{d\tau}(\tau_j) \;\approx\; \sum_{j=0}^{N} D_{ij}\, x(\tau_j), \qquad i = 0, \ldots, N.$$

Using the chain rule the τ and time derivatives can be related allowing the node differentiation matrix to be checked against the system dynamics $f(x_i, u_i)$.

$$\dot{x}(t_j) = \frac{2}{t_f - t_0}\sum_{j=0}^{N} D_{ij}\, x(\tau_j),$$

$$\sum_{j=0}^{N} D_{ij}\, x_j = \frac{t_f - t_0}{2}\, f(x_i, u_i), \qquad i = 1, \ldots, N-1.$$

Where D is a Chebyshev Differentiation Matrix defined by:

$$D_{N_{00}} = \frac{2N^2 + 1}{6}$$

$$D_{N_{jj}} = \frac{-x_j}{2(1 - x_j^2)}$$

$$D_{N_{ij}} = \frac{c_i}{c_j}\frac{(-1)^{i+j}}{x_i - x_j}$$

$$D_{N_{NN}} = -\frac{2N^2 + 1}{6}$$

$$c_{i/j} = \begin{cases} 2 & i/j = 0 \text{ or } N, \\ 1 & \text{otherwise.} \end{cases}$$

For $N = 6$ the differentiation matrix is:

$$D_6 = \begin{bmatrix} \frac{73}{6} & -(8+4\sqrt{3}) & 4 & -2 & \frac{4}{3} & 4\sqrt{3}-8 & \frac{1}{2} \\ 2+\sqrt{3} & -\sqrt{3} & -(1+\sqrt{3}) & \frac{2}{\sqrt{3}} & 1-\sqrt{3} & \frac{1}{\sqrt{3}} & \frac{1-\sqrt{3}}{1+\sqrt{3}} \\ -1 & 1+\sqrt{3} & -\frac{1}{3} & -2 & 1 & 1-\sqrt{3} & \frac{1}{3} \\ \frac{1}{2} & -\frac{2}{\sqrt{3}} & 2 & 0 & -2 & \frac{2}{\sqrt{3}} & -\frac{1}{2} \\ -\frac{1}{3} & \sqrt{3}-1 & -1 & 2 & \frac{1}{3} & -(1+\sqrt{3}) & 1 \\ \frac{1-\sqrt{3}}{1+\sqrt{3}} & \sqrt{3} & \sqrt{3}-1 & -\frac{2}{\sqrt{3}} & 1+\sqrt{3} & \sqrt{3} & -(2+\sqrt{3}) \\ -\frac{1}{2} & 8-4\sqrt{3} & -\frac{4}{3} & 2 & -4 & 8+4\sqrt{3} & -\frac{73}{6} \end{bmatrix}$$

The chebyshev cost function is:

$$J = \frac{t_f - t_0}{2}\int_{t_0}^{t_f} \ell(x(t), u(t), t)\, dt \;+\; \phi(x(t_f))$$

$$J \;\approx\; \frac{t_f - t_0}{2} \sum_{j=0}^{N} w_j \, \ell(x_j, u_j, t_j) \;+\; \phi(x_N), \qquad t_j = \frac{t_f - t_0}{2}\,\tau_j + \frac{t_f + t_0}{2}.$$

where $\phi(x(t_f))$ is the terminal cost and w_j are the Clenshaw-Curtis quadrature weights:

$$w_{\text{j}_{\text{even}}} = \begin{cases} \dfrac{2}{N}\left(1 - \displaystyle\sum_{n=1}^{\frac{N}{2}-1} \frac{2}{4n^2-1}\cos\!\left(\tfrac{2nj\pi}{N}\right) - \frac{(-1)^j}{N^2-1}\right), & 1 \le j \le N-1, \\ \dfrac{1}{N^2-1}, & j = 0 \text{ or } j = N \quad (\text{since } N \text{ even}). \end{cases}$$

$$w_{\text{j}_{\text{odd}}} = \begin{cases} \dfrac{2}{N}\left(1 - \displaystyle\sum_{n=1}^{\frac{N}{2}-1} \frac{2}{4n^2-1}\cos\!\left(\tfrac{2nj\pi}{N}\right)\right), & 1 \le j \le N-1, \\ \dfrac{1}{N^2}, & j = 0 \text{ or } j = N. \end{cases}$$

Comparison of Methods

To compare our different formulations for the NLP problem, we ran a series of simulations each with the drone beginning in a unique and perturbed starting state. In all cases, the drone goal state was to be vertically balanced and motionless at the point $(x, y, z) = (0, 0, 0)$ in three dimensional space. We initially ran a set of seven different simulations. The results for all the simulations were similar in both quality of solutions and timing results, so we present only four of those simulations here. For each simulation, we computed the average time required to solve the NLP problem at each iteration. The quality of the solutions produced by each method was assessed by comparing the state and control graphs through time. All experiements were run on a Macbook Pro 3.49GHz Apple M2 chip with 8GB RAM. Code was implemented in Python 3.13 using CasADI 3.7.1.

NLP Solver

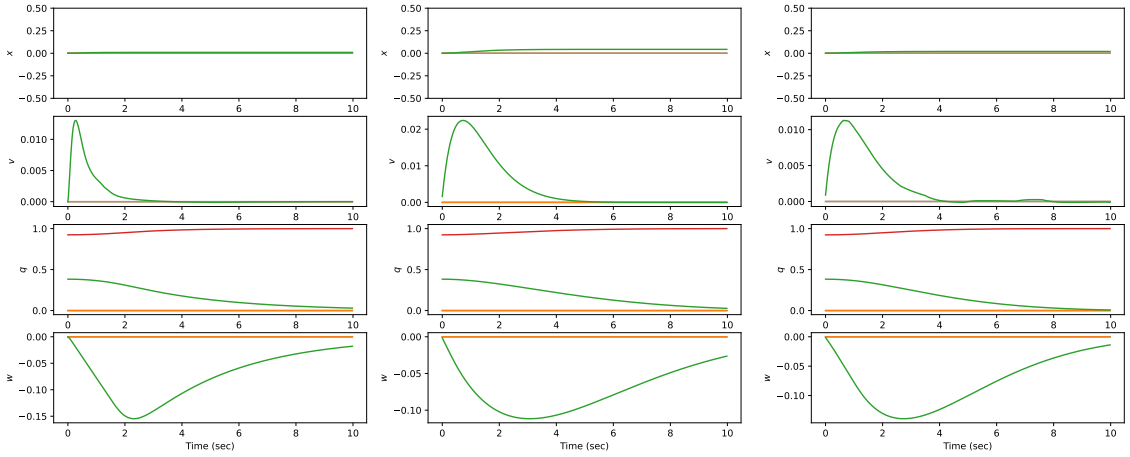
To solve our non-linear programming (NLP) problems we used the `ipopt` solver that comes installed with CasADi 3.7.1. This solver requires an additional subroutine to solve sparse matrix systems. We experimented with the `mumps` solver that comes with CasADi, and also tried using the `ma27` and `ma57` solvers from HSL (Harwell Subroutine Library) [HSL13]. All solvers were compiled with Apple clang version 17.0.0. The HSL `ma27` solver produced the most efficient solutions overall. All solvers and experiments were run with the same solver settings.

```
ipopt_settings = {
    'ipopt.max_iter': 100,
    'ipopt.tol': 1e-3,
    'ipopt.acceptable_tol': 3e-2,
    'ipopt.linear_solver': 'ma27',
}
```

Results

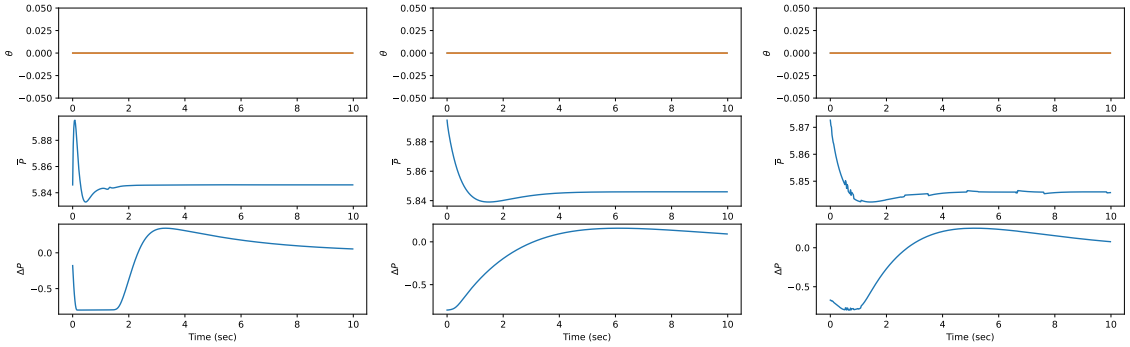
We begin with our first simulation which requires the drone to perform a 45° rotation about the z axis to achieve the goal state. Figure (3) shows a graph of the drone's state overtime for each of our methods: orthagoal collocation, multiple shooter, and Chebyshev pseudospectral. The figures (6), (9) and (12) show

similar graphs for our other three simulations. Comparison graphs of the control variables are given in figures (4), (7), (10) and (13). Finally, graphs comparing the CPU time for NLP calls throughout the simulations are given in figures (5), (8), (11) and (14).



(a) orthagonal collocation (b) multiple shooter (c) Chebyshev pseudospectral

Figure 3: State data on a simulation requiring a 45° rotation about the z axis.



(a) orthagonal collocation (b) multiple shooter (c) Chebyshev pseudospectral

Figure 4: Control data on a simulation requiring a 45° rotation about the z axis.

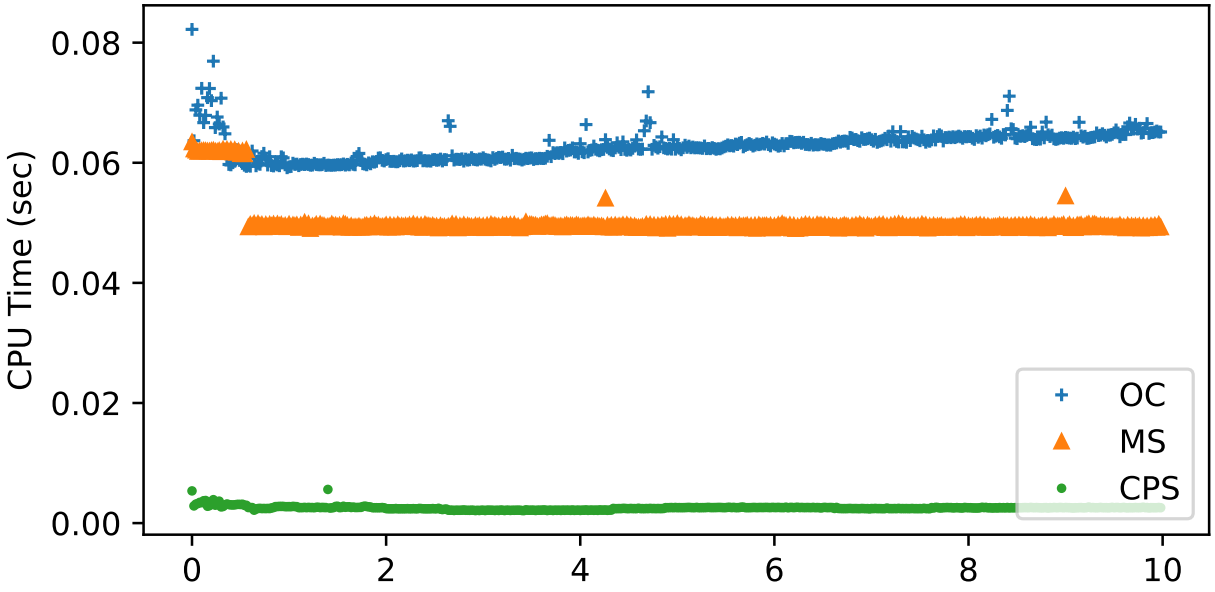
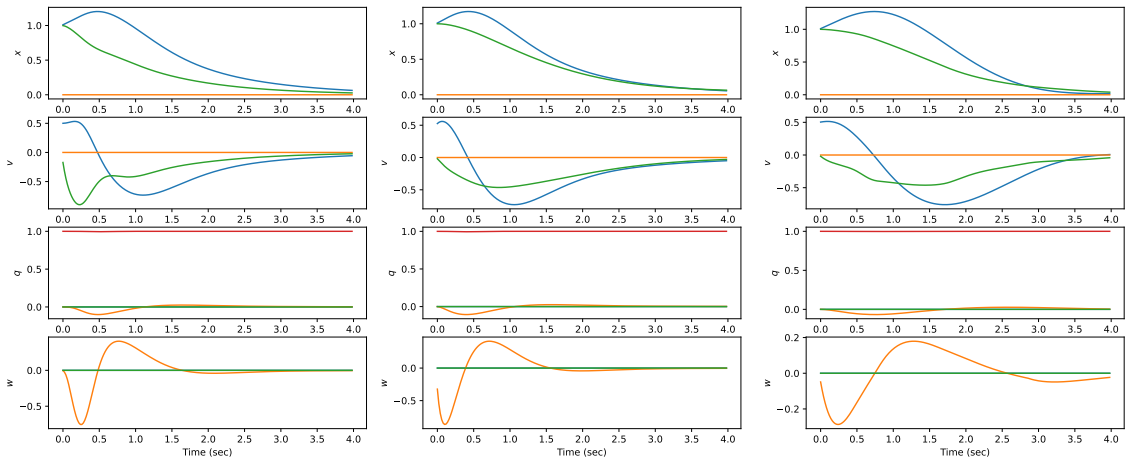
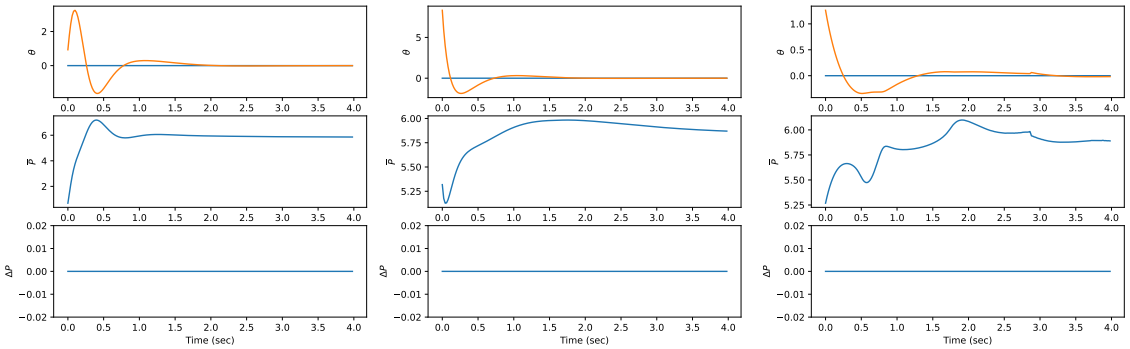


Figure 5: CPU time for simulation requiring a 45° rotation about the z axis.



(a) orthagonal collocation (b) multiple shooter (c) Chebyshev pseudospectral

Figure 6: State data on a simulation with initial conditions: $x = 1$, $z = 1$, $v_x = 0.5$



(a) orthagonal collocation (b) multiple shooter (c) Chebyshev pseudospectral

Figure 7: Control data on a simulation with initial conditions: $x = 1$, $z = 1$, $v_x = 0.5$

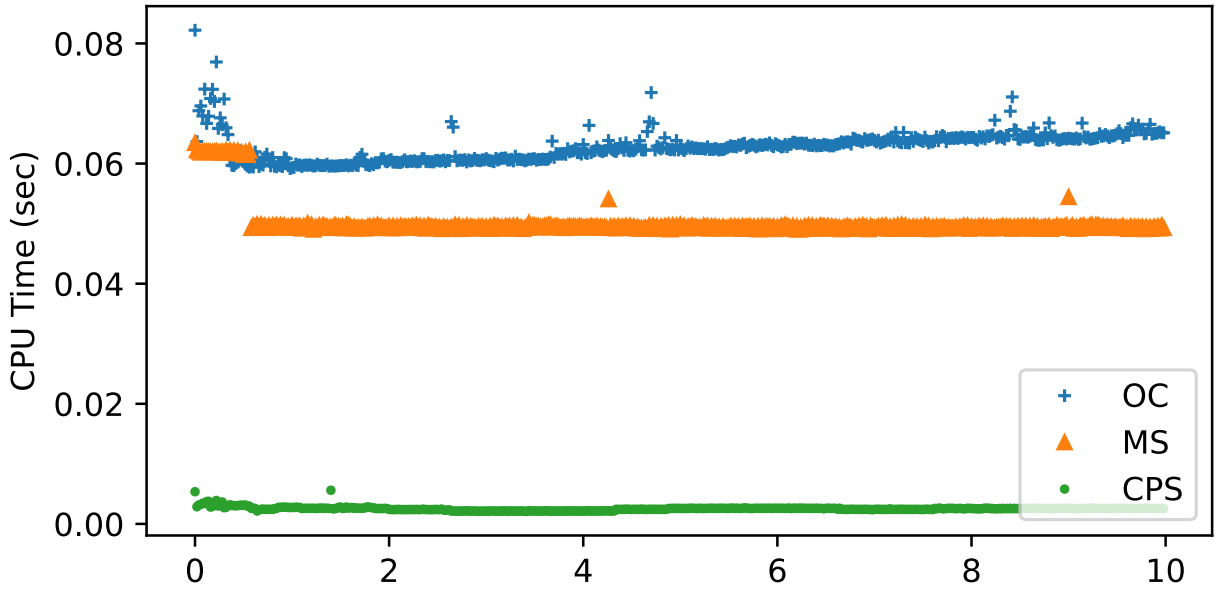
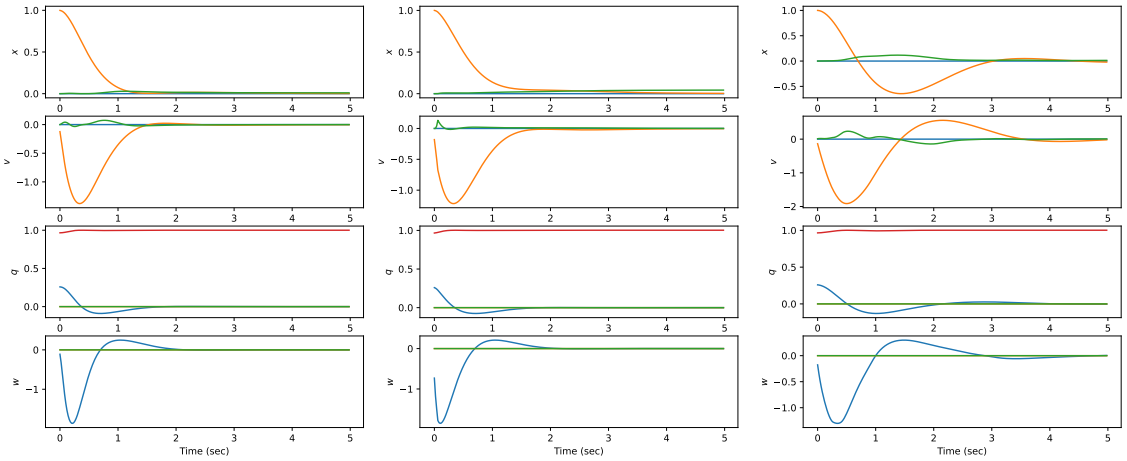
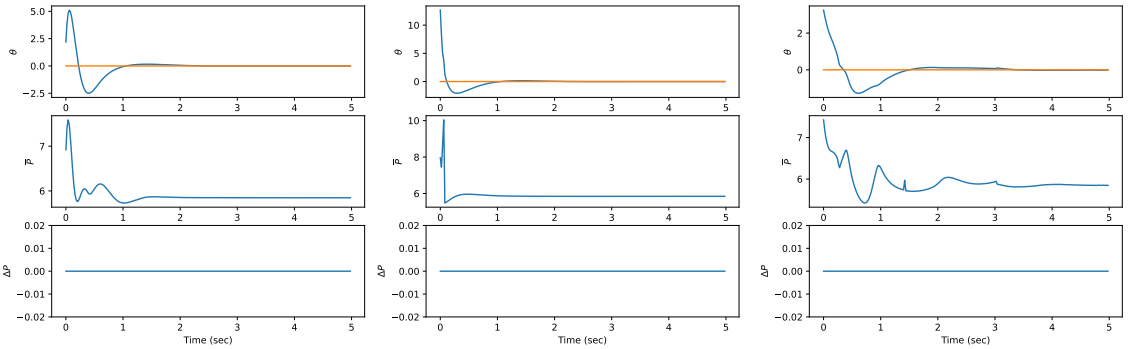


Figure 8: CPU time for a simulation with initial conditions: $x = 1$, $z = 1$, $v_x = 0.5$



(a) orthagonal collocation (b) multiple shooter (c) Chebyshev pseudospectral

Figure 9: State data on a simulation with initial conditions: $y = 1{,}15°$ rotation about x axis.



(a) orthagonal collocation (b) multiple shooter (c) Chebyshev pseudospectral

Figure 10: Control data on a simulation with initial conditions: $y = 1{,}15°$ rotation about x axis.

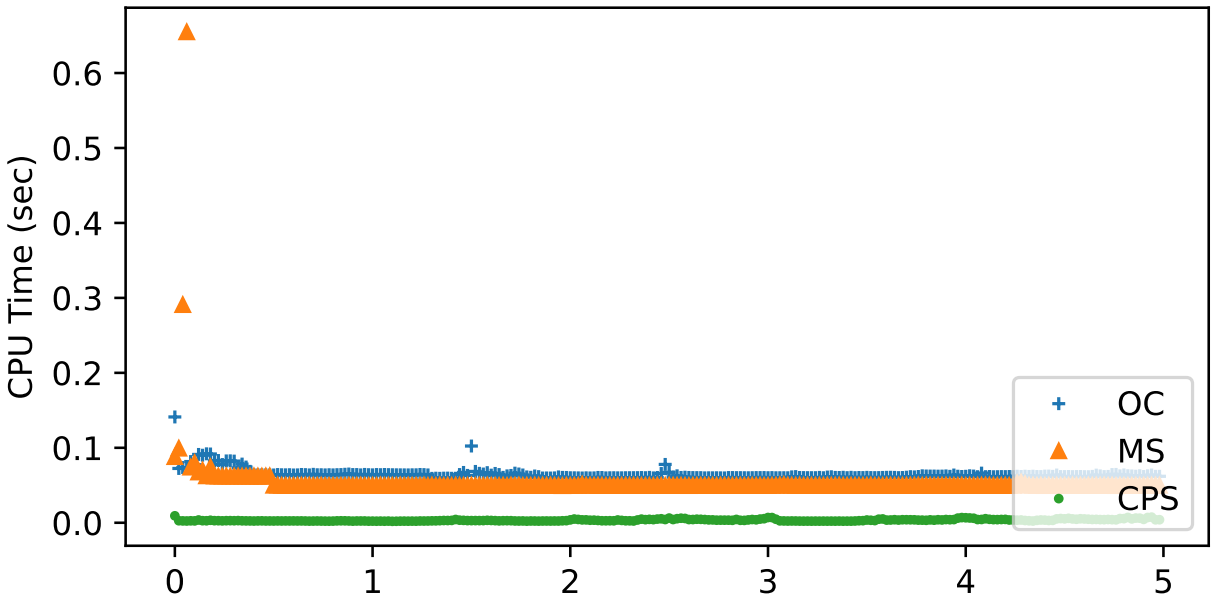
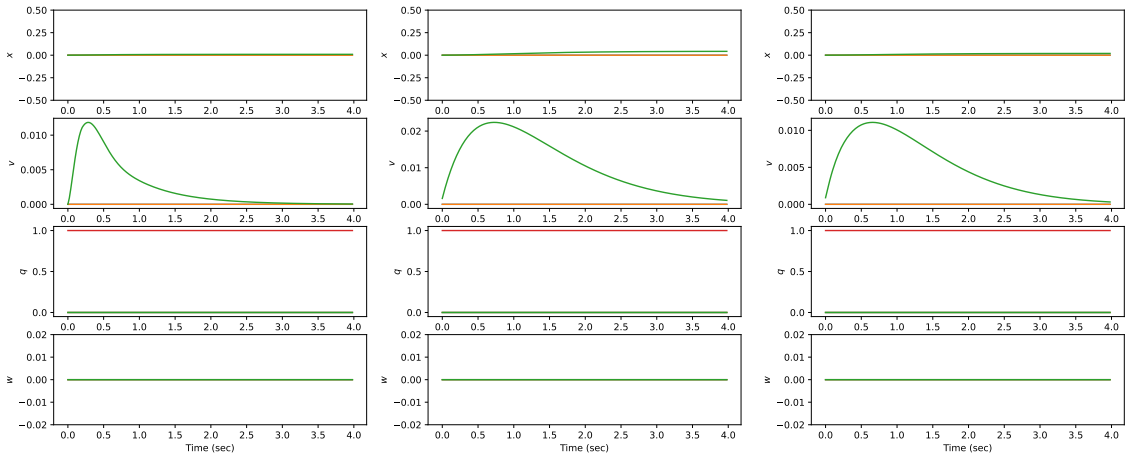
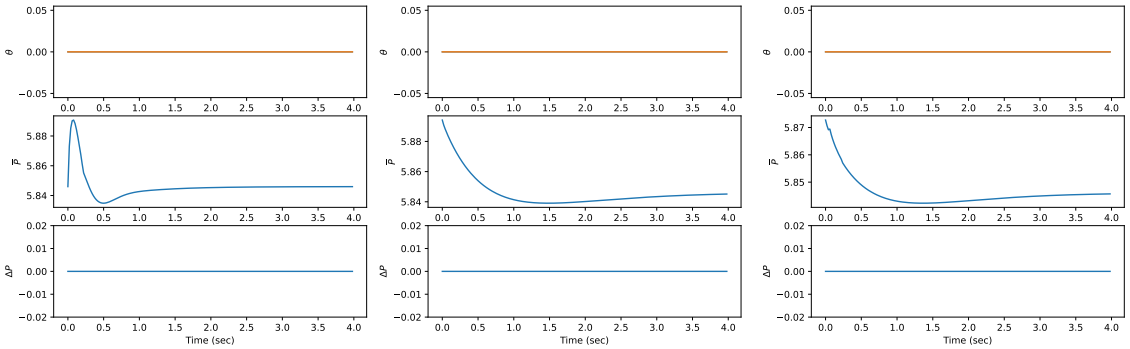


Figure 11: CPU time for simulation with initial conditions: $y = 1{,}15°$ rotation about x axis.



(a) orthagonal collocation (b) multiple shooter (c) Chebyshev pseudospectral

Figure 12: State data on a simulation with initial state equal to goal state.



(a) orthagonal collocation (b) multiple shooter (c) Chebyshev pseudospectral

Figure 13: Control data on a simulation with initial state equal to goal state.

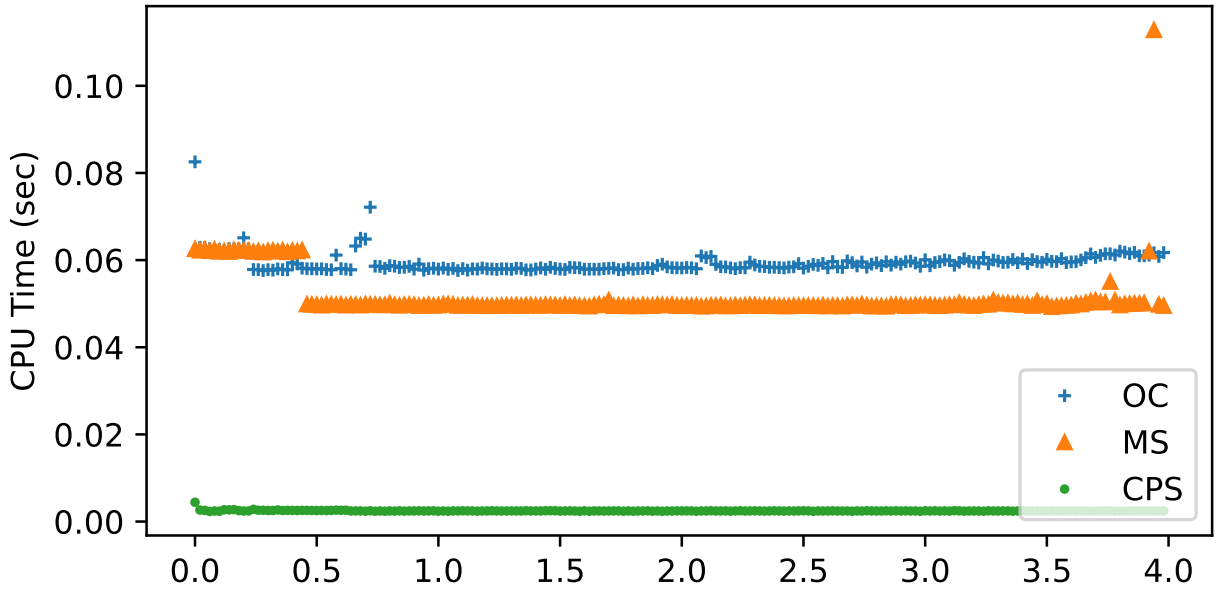


Figure 14: CPU time for simulation with initial state equal to goal state.

All of the methods provide good quality solutions with state values that behaved smoothly and converged to the desired goal state. The graphs of control variables were generally smooth, but all methods occasionally constructed solutions where the average thrust $\overline{P}$ seemed more irregular than required to reach the goal state. Real world experiments on the drone will be needed to determine if solutions with these irregularities will negatively affect performance in practice.

Our goal is to run our control algorithm at 50Hz on the raspberry Pi 5. The CPU speed of the Pi is 2.4GHz while our experiments were run on a machine with a CPU speed of 3.49GHz. To meet our goal of 50Hz on the Pi we need a solution time on the experiment machine of less than 0.014 seconds. Table (1) summarizes the average CPU time for solution of the NLP problem for each simulation and each method. The run times for the orthagonal collocation and multiple shooter are both too slow to meet our 50Hz target. Additional changes such as reducing the length of the horizon or relaxing convergence criteria might improve these times somewhat, but they are unlikely to provide the performance needed for this application

initial state	OC	MS	CPS
45° rotation about z axis	0.063	0.056	0.002
$x = 1$, $z = 1$, $v_x = 0.5$	0.059	0.046	0.003
y = 1,15° rotation about x axis	0.063	0.06	0.004
initial state equal to goal state	0.062	0.056	0.002

Table 1: Comparison of average CPU time in seconds for solution of NLP for a series of simulations. Methods include orthagonal collocation (OC), multiple shooter (MS), and Chebyshev pseudospectral method (CPS).

of NMPC. The Chebyshev pseudospectral method is substantially faster and comfortably meets our 0.014 solution time while providing solution graphs with comparable quality.

Conclusions

We implemented three different formulations of the NPL problem for an NMPC algorithm controlling a thrust vector drone and compared their solution quality and relative efficiency. All methods produced good quality solutions while only the Chebyshev pseudospectral method provided solutions that are efficient enough to control the drone on a Raspberry Pi 5. Further real-world experiments are required to determine if the solutions produced provide good control in practice.

References

[HSL13] HSL. A collection of fortran codes for large scale scientific computation., 2013.