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

Izzy Mones and Heidi Dixon

September 18, 2025

Introduction

This paper is part of a larger project inspired by the small-scale rocket prototype presented by Linsen et al. (2022) which aims to study thrust vector control (TVC) and vertical take of and landing (VTOL) using a gimballed thrust drone with electric motors and propellers.

Rockets are inherently unstable. They have highly nonlinear, coupled dynamics, and require fast control algorithms that respond quickly to disturbances and sensor noise. Unfortunately, validating control techniques on full scale rockets is prohibitively costly and high-risk. There is a need for inexpensive test vehicles that allow experimentation with rocket guidance, navigation, and control. A gimballed thrust drone with electric motors and propellers is a good choice for replicating rocket dynamics on a much smaller budget. A couple of very inexpensive designs have been preposed and implemented (Spannagl et al., 2021), (Linsen et al., 2022).

Thrust vector control, which uses a gimbal to angle the direction of engine thrust and control the rocket's position and attitude, is an important control method for rockets. Vertical takeoff and landing (VTOL) milestones like NASA's Lunar Landing Research Vehicle (LLRV) which allowed Apollo astronauts to practice lunar landings, SpaceX's Falcon 9 landings, and the future Artemis III mission that aims to land and return astronauts from the moon illustrate the value of propulsive VTOL with TVC. The drone design by Linsen et al. (2022) allows experimentation with of control algorithms for VTOL and VTC. We have designed a similar drone based on their images and design descriptions. Within this context there are still many details and choices to explore.

Our drone implements VTC and VTOL using a nonlinear model predictive control (NMPC) algorithm (Garcia et al., 1989). NMPC algorithms can handle nonlinear rocket dynamics and manage constraints like gimbal angle range, servo rates, and thrust limits. NMPC optimizes control inputs across a finite time horizon at each time step. In this sense it creates a *plan* of control inputs allowing it to anticipate future control requirements. Revising the plan at each time step allows it to respond to disturbances. The main downside of NMPC is its high computational load. NMPC solves a constrained nonlinear optimization problem (NLP) at every control step, and as a result it is much slower than simple control algorithms like proportional integral derivative (PID) controllers or linear quadratic regulators (LQR). There are a variety of ways to formulate an NMPC problem instance as a nonlinear programming problem. Different formulations may vary in their solution times and solution accuracy. The focus of this paper is experimenting with different choices of NLP formulations and evaluating their speed and accuracy. Our goal is to run our control algorithm on a Raspberry Pi 5 at 50Hz so solution speed will be very important.

We compare three approaches: a Runge-Kutta multiple-shooter method, an orthogonal collocation with Gauss-Radau nodes implemented in the **do-mpc** *library (Fiedler et al., 2023), and a Chebyshev pseudospectral collocation approach. We compare the relative accuracy of these techniques and their relative efficiency.*

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:

$$\dot{\vec{p}} = \vec{v} \tag{1}$$
$$\dot{\vec{v}} = \frac{1}{m} R(\vec{q}) \vec{F}_b + \vec{g} \tag{2}$$
$$\dot{\vec{q}} = \frac{1}{2} Q(\vec{\omega}) \vec{q} \tag{3}$$
$$\dot{\vec{\omega}} = I^{-1} \left(\vec{M}_b - \vec{\omega} \times (I\, \vec{\omega}) \right) \tag{4}$$

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

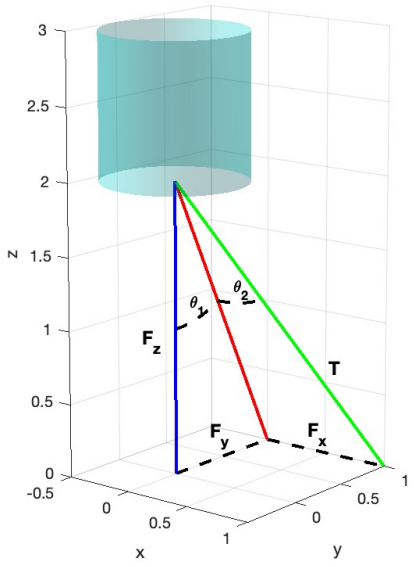
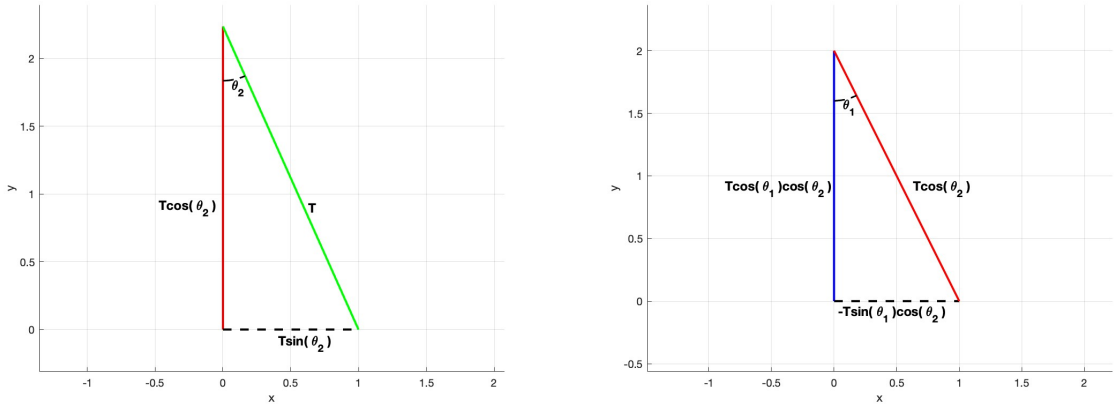


Figure 1: Three-dimensional thrust decomposition for a 2-axis gimballed drone.



(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.

$$R(\vec{q}) = \begin{bmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y - q_wq_z) & 2(q_xq_z + q_wq_y) \\ 2(q_xq_y + q_wq_z) & 1-2(q_x^2+q_z^2) & 2(q_yq_z - q_wq_x) \\ 2(q_xq_z - q_wq_y) & 2(q_yq_z + q_wq_x) & 1-2(q_x^2+q_y^2) \end{bmatrix}$$

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.

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

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

Nonlinear Model Predictive Control (NMPC)

Model predictive control (MPC) is a control algorithm that solves a linear optimization problem at each time step. An MPC solution yields a series of control inputs over a finite time horizon, creating 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 control values for the first step of the plan is applied to the system and then the plan is revised at the next time step, allowing the method to react to disturbances and noise. Nonlinear model predictive control (NMPC) is a variant of MPC that allows nonlinear constraints.

Generally, given a set of continuous time nonlinear differential equations $\dot{x} = f(x(t), u(t))$ and an initial state x_0, the finite horizon NMPC optimization problem is

$$\begin{aligned} \min_{u(t)} \quad & \int_0^{t_f} \ell(x(t), u(t))\, dt \\ \text{s.t.} \quad & \dot{x} = f(x(t), u(t)) \\ & x(0) = x_0 \end{aligned}$$

The solution $u(t)$ is a real valued function that minimizes the cost function over time interval $[0, t_f]$. The constraints require that $u(t)$ and $x(t)$ obey the system dynamics $\dot{x} = f(x(t), u(t))$ and the initial state $x(0)$ must be equal to the current state x_0. Given this framework, it is easy to add additional constraints that enforce system limitation such as minimum or maximum control values.

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. The NMPC framework also allows us to enforce the mechanical limitations of our thrust motors and gimbal servos. Libraries like CasADi (Andersson et al., 2018) and **do-mpc** (Fiedler et al., 2023) make NMPC easy to implement in practice.

An NMPC instance for the thrust vector drone is a minimization problem of the form

$$\begin{aligned} \min_{u(t)} \quad & \int_0^{t_f} \ell(x(t), u(t))\, dt \ + \ \phi(x_f) && (5) \\ \text{s.t.} \quad & \dot{x} = f(x(t), u(t)) \\ & x(0) = x_0 \\ & -\theta_{max} \le \theta_1 \le \theta_{max} && (6) \\ & -\theta_{max} \le \theta_2 \le \theta_{max} && (7) \\ & T_{min} \le \overline{P} + P_\Delta/2 \le T_{max} && (8) \\ & T_{min} \le \overline{P} - P_\Delta/2 \le T_{max} && (9) \\ & 0 \le z. && (10) \end{aligned}$$

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.

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

We also add a termial cost

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

The problem instance also enforces minimum and maximum gimbal angles (6,7), minimum and maximum values for each thrust motor (8, 9) and restrictons on the drone's vertical height (10).

NLP Formulations

To solve the continuous time optimal NMPC instance it is formulated as a discrete time finite dimensional 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 problems, 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.

An NLP problem over a finite set of decision variables w has the form

$$\min_{w \in \mathbb{R}^n} \; F(w) \tag{11}$$

$$\text{s.t.} \;\; G(x_0, w) = 0 \tag{12}$$

$$H(w) \leq 0 \tag{13}$$

It has an optimization function (11), a set of equality constraints (12) and a set of inequality constraints (13). In practice, the distinction between equality and inequality constraints is moot since an equality constraint can be written as two inequality constraints and an inequality constraint can be written as an equality constraints with the introduction of a slack variable.

To turn our NMPC problem into an NLP problem the continuous time functions $x(t)$ and $u(t)$ are parameterized into a finite set of discrete variables over the control trajectory. Then differential equations for the system dynamics are discretized using a numerical integration method. There are a many ways to do this and the choices we make will affect the size of the problem, the speed at which it can be solved, and the accuracy of the numerical approximation. Typically there is a tradeoff between speed and accuracy. Next we explore three different ways to formulate our NMPC problem as an NLP and compare and contrast their ability produce viable control trajectories and their relative efficiency. These three methods are in no way exhaustive. We chose to implement pseudospectral collocation because it was shown to be effective in an earlier paper on thrust vector drones (Linsen et al., 2022) and we chose multiple shooting and orthagonal collocation because they were very easy to implement.

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 and we make copies of our state x_k and control u_k variables for each time step. The initial state is initialized to the current state

$$x_0 - X_0 = 0.$$

Using the system dynamics $\dot{x} = f(x(t), u(t))$ and Runge-Kutta within each time step $[t_k, t_{k+1}]$ we forward integrate our differential equations

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

to create constraints (14) that enforce the change of state between time steps.

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.

$$x_r = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}^T, \qquad u_r = \begin{bmatrix} 0 & 0 & T_{\text{hover}} & 0 \end{bmatrix}^T.$$

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

Constraints on gimbal angles and thrust motors are encoded by adding a copy of the appropriate inequalities at each time step. To improve performance of our multiple shooter method we implemented warm starts, shifting the previous solution forward one time step and using that as an initial guess for the problem at the current time step.

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

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

This can easily become computationally demanding for long horizons and high-dimensional systems. For a thrust vector drone system with a horizon length of 100 time steps there are $13 * (100 + 1) + 4 * 100 = 1713$ decision variables. While the number of variables is very high, the constraint matrix is typically very sparse since each constraint references only a small number of variables.

Multiple shooting is easy to implement using CasADi. There are very few choices to toggle in producing the best multiple shooting implementation. Outside of specific NLP solver settings, the main choices are the length of your time steps and the length of your horizon. Working with a time step of $\Delta t = 0.02$ we found that time horizons with $N \leq 50$ often produced trajectories that were irregular or did not converge. An $50 \leq N \leq 80$ produced mostly smooth graphs that converged however the convergence was a bit slow. An $N \geq 80$ seemed to produce good quality graphs that converged promptly. However, our goal for all implementations is to have a horizon of 2 seconds.

Orthogonal Collocation using do-mpc

Of all the methods we implemented, orthogonal collocation was the easiest. We used the open source **do-mpc** Python library for model predictive control (Fiedler et al., 2023) which implements MPC and NMPC using orthogonal collocation with Radau nodes. The **do-mpc** library is ideal for prototyping MPC and NMPC algorithms and requires minimal understanding of MPC and no understanding of orthogonal collocation to build functioning algorithms.

Need to discuss our cost function and how to encode servo and thrust minimum and maximums. We should note that with do-mpc you don't need to know any of this because the NLP constraints are generated automatically from the system dynamics. Warm starts are done automatically in do-mpc. cost function and other constraints are entered using continuous model variables and do-mpc adapts them to the chosen trajectory variables. You can specify degree of collocation and time step size.

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:

Terms not defined: K, β, α

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

When P_K are the Gauss-Jacobi polynomials:

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

Undefined terms $z^K(\tau)$

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

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

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

How is $z^K(\tau)$ related to $x(t)$ and $u(t)$? Define h_i.

By differentiating the state trajectory and equating it with the dynamics s$\dot{x} = f(x(t), u(t))$, a dynamics constraint is formed:

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

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.

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

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

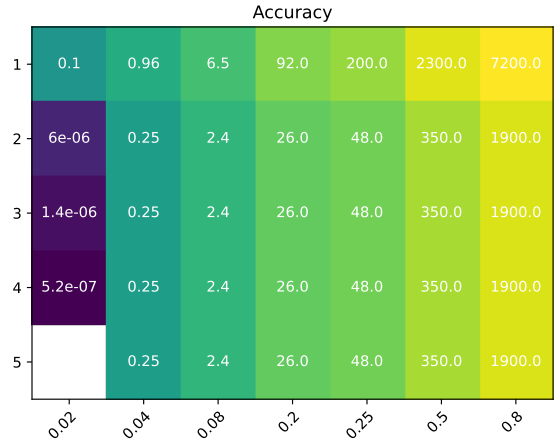
What is $i(\cdot)$. L, Φ? The cost function is defined as:

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

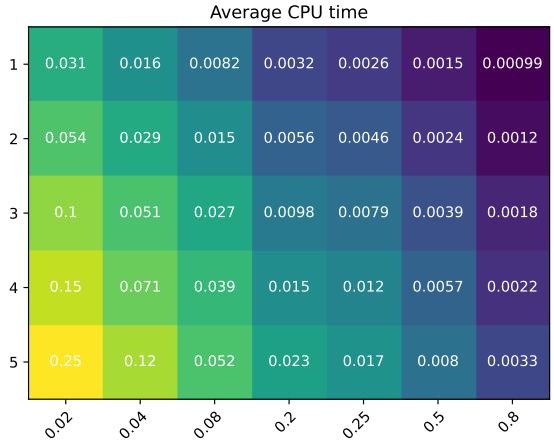
$$J = \Phi(z(t_f)) + \sum_{i=1}^{N} h_i \sum_{k=1}^{K} \omega_k L(z_{ik}, u_{ik}, t_{ik})$$

Where the weights ω_k are determined by Radau quadrature, which integrates all polynomials up to $2s - 1$.

Fix the length of the time horizon to 2 seconds and choices: number of finite elements, number of collocation points. Run a simulation for 200 iterations. Consider collocation degrees $c = 1, 2, \ldots, 5$ and time intervals $[0.02, 0.04, 0.08, 0.2, 0.25, 0.5, 0.8]$. Run with 5 collocation points and 0.02 time step and save state data. This is the most fine grained collocation scheme and has very nice curves that smooth and converge. To compute the accuracy of a run we look at the squared error from this run summed over each iteration. On visual inspection this works well and accuracy less then 50 are very good, smooth, converge promptly. So time step 0.25 with 2 collocation points is the best.



(a) Accuracy metric for orthogonal collocation with different size time steps and collocation degrees.



(b) Average CPU time for orthogonal collocation with different size time steps and collocation degrees.

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:

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

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:

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

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

warm starts. Need to summarize our choices. Spectral order, time horizon.

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 (**?**).

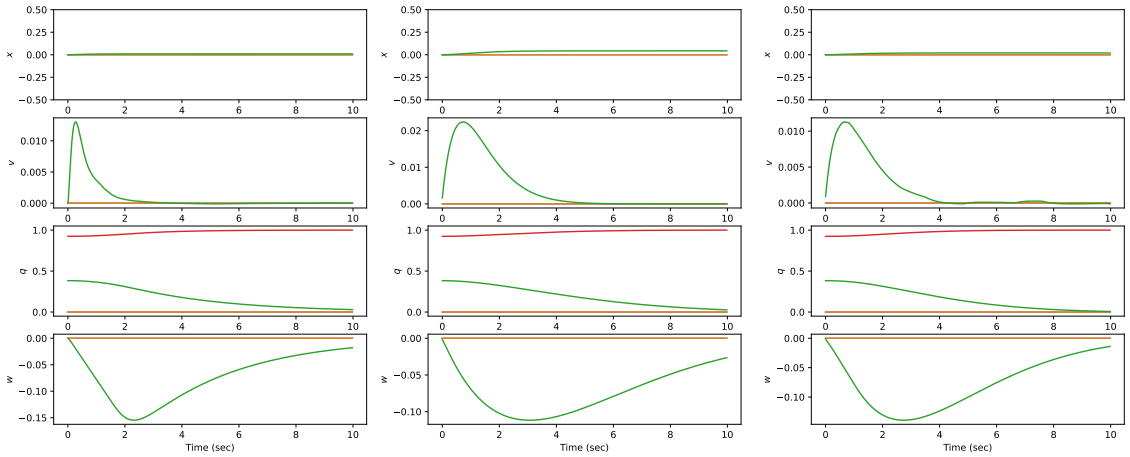
NLP Solver

To solve our non-linear programming (NLP) problems we used the `ipopt` solver (Wächter and Biegler, 2006) 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) (HSL, 2013). 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',
}
```

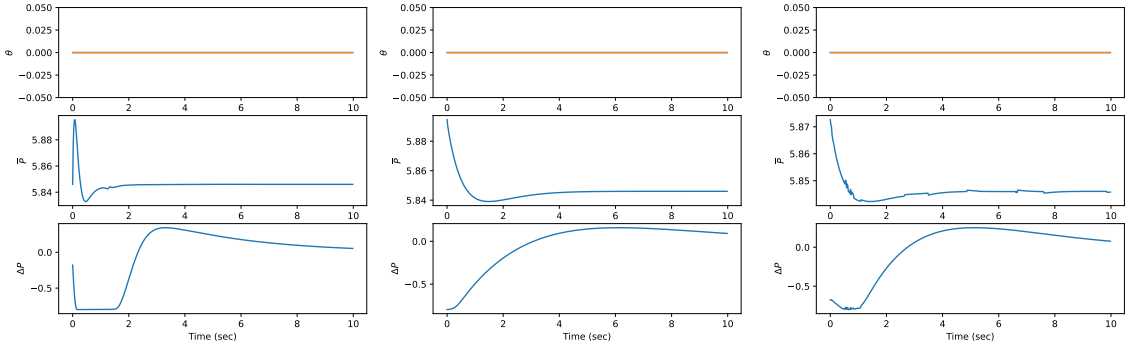
Results

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 (4) shows a graph of the drone's state overtime for each of our methods: orthagoal collocation, multiple shooter, and Chebyshev pseudospectral. The figures (7), (10) and (13) show similar graphs for our other three simulations. Comparison graphs of the control variables are given in figures (5), (8), (11) and (14). Finally, graphs comparing the CPU time for NLP calls throughout the simulations are given in figures (6), (9), (12) and (15).



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

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



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

Figure 5: Control data on a simulation requiring a 45° rotation about the z axis.

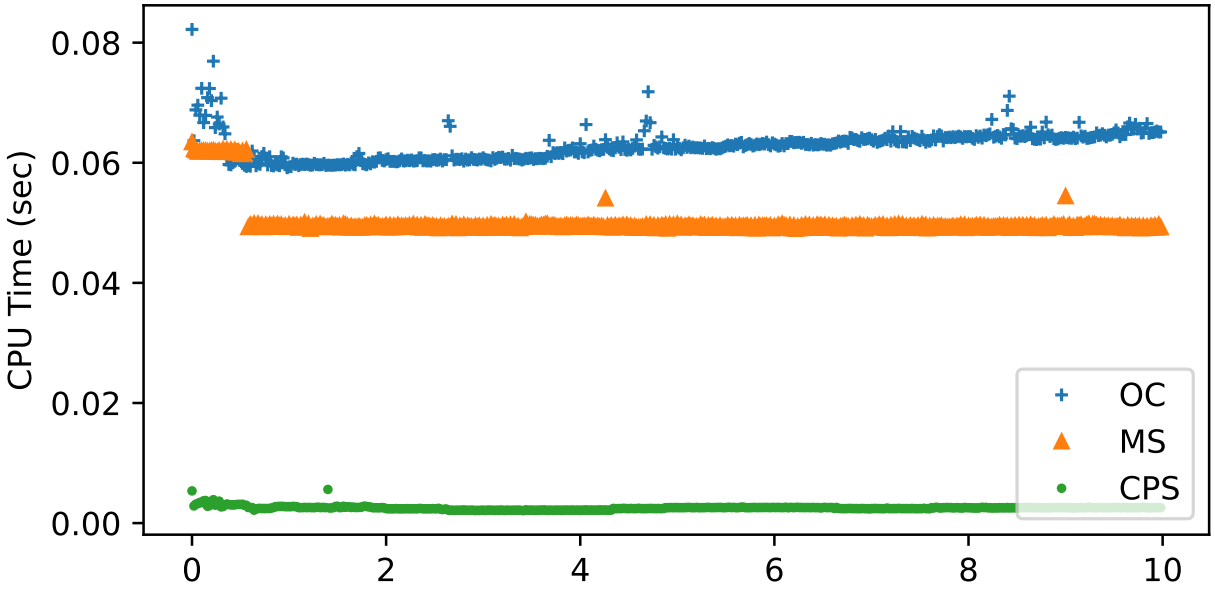
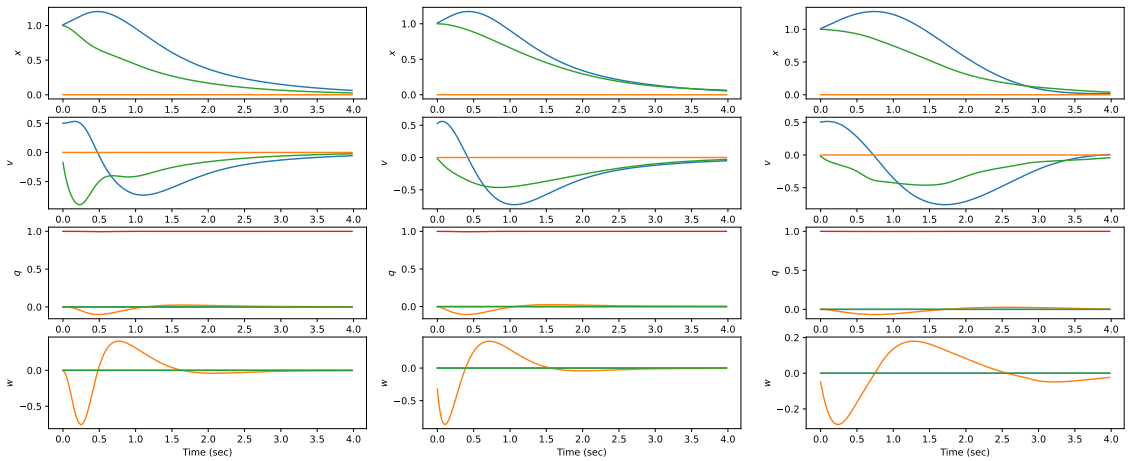
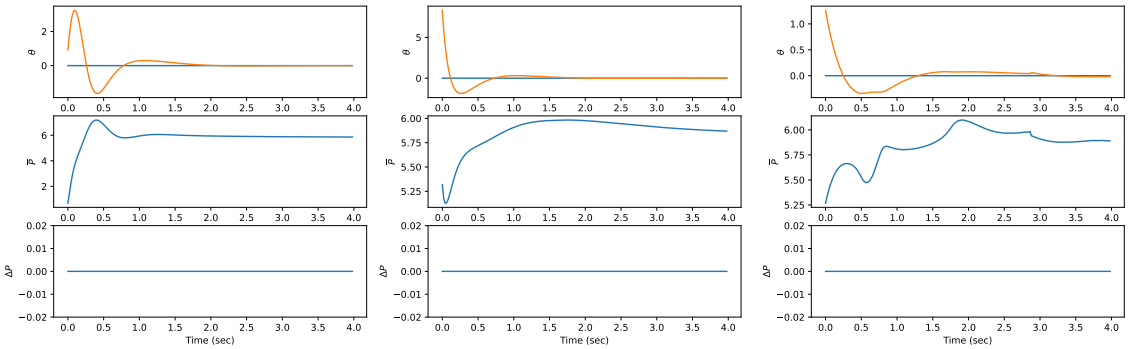


Figure 6: CPU time for simulation requiring a 45° rotation about the z axis.



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

Figure 7: 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 8: Control data on a simulation with initial conditions: $x = 1$, $z = 1$, $v_x = 0.5$

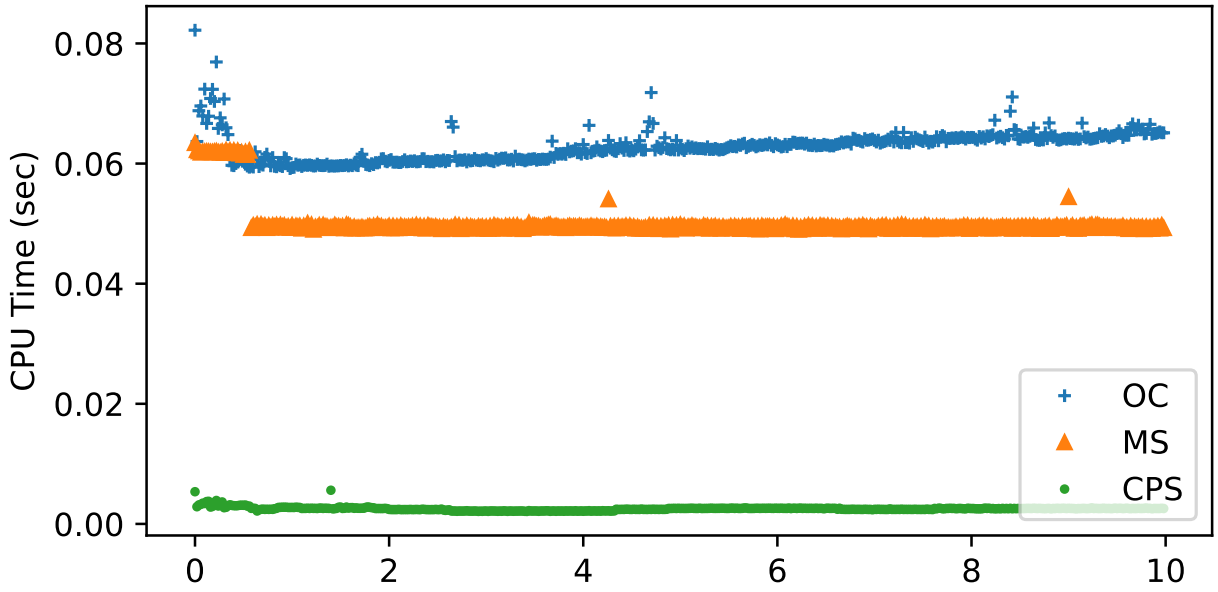
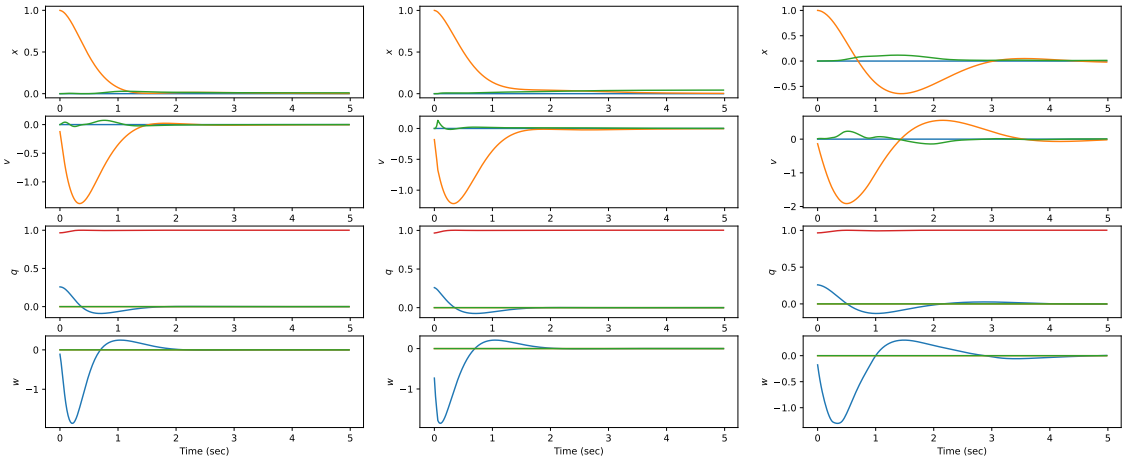
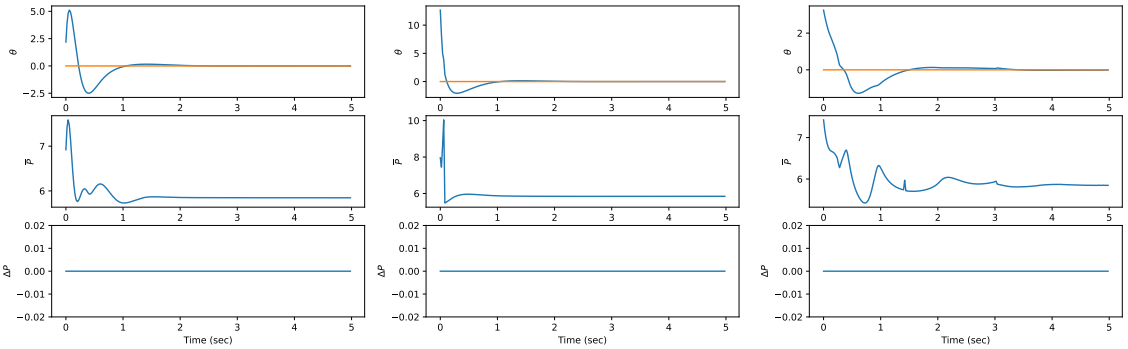


Figure 9: CPU time for a simulation with initial conditions: $x = 1$, $z = 1$, $v_x = 0.5$



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

Figure 10: 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 11: Control data on a simulation with initial conditions: $y = 1{,}15°$ rotation about x axis.

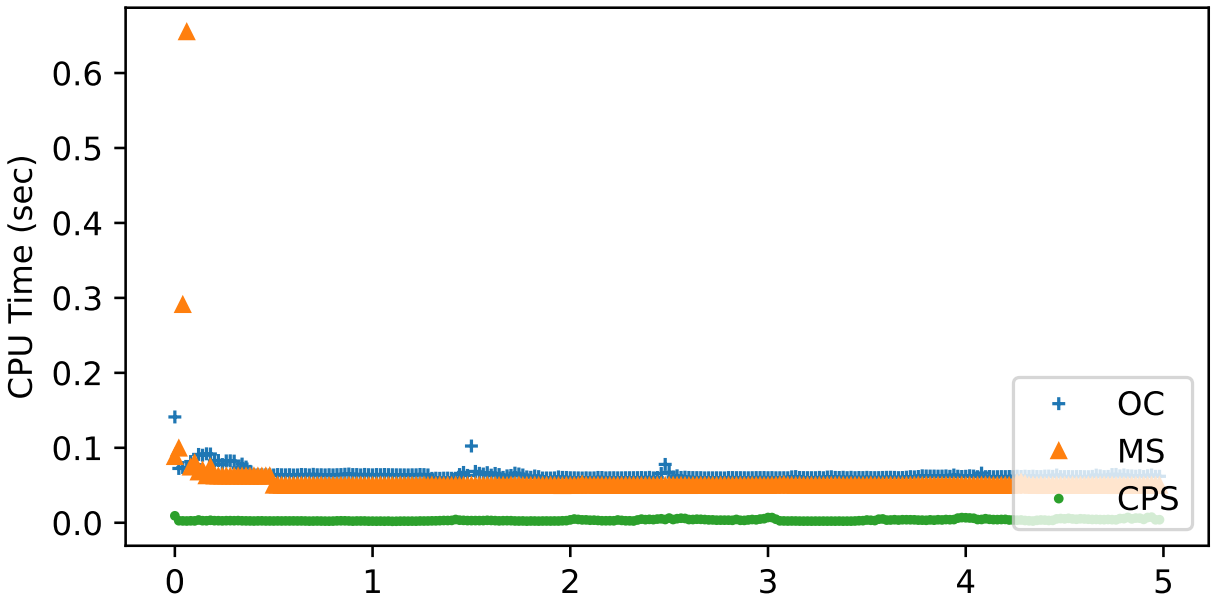
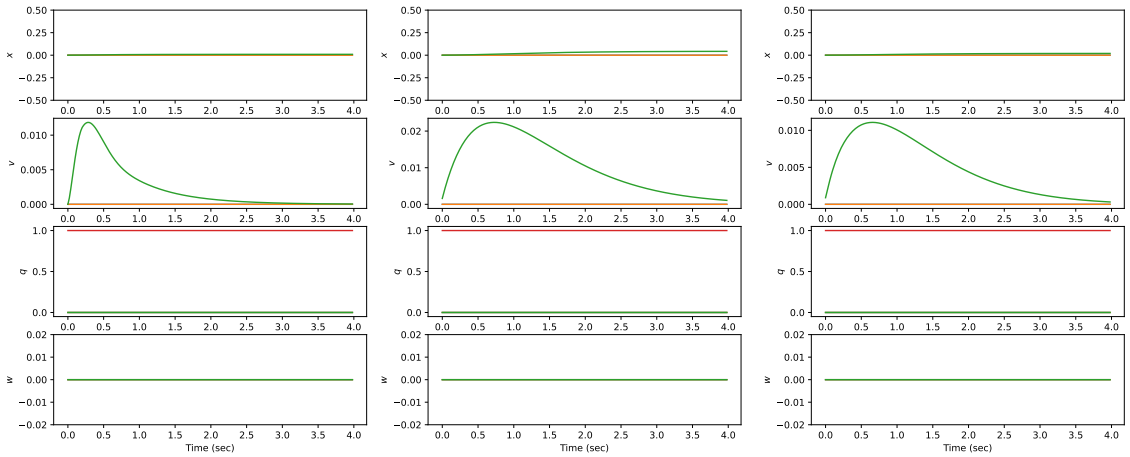
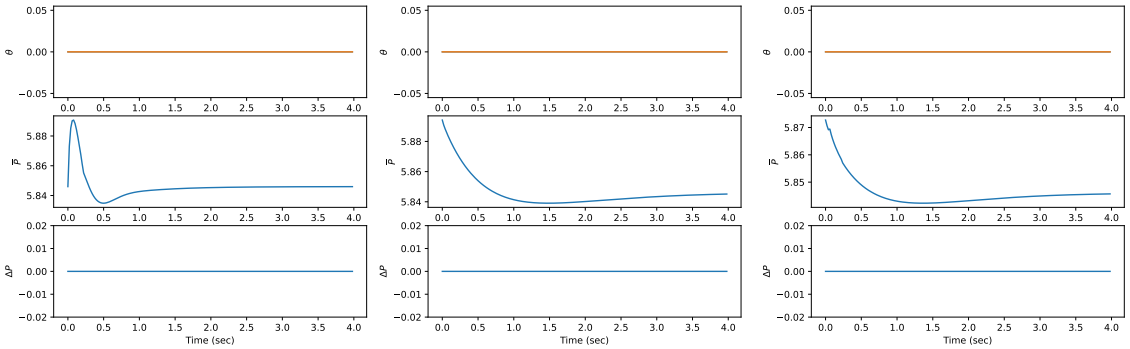


Figure 12: CPU time for simulation with initial conditions: $y = 1{,}15°$ rotation about x axis.



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

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



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

Figure 14: Control data on a simulation with initial state equal to goal state.

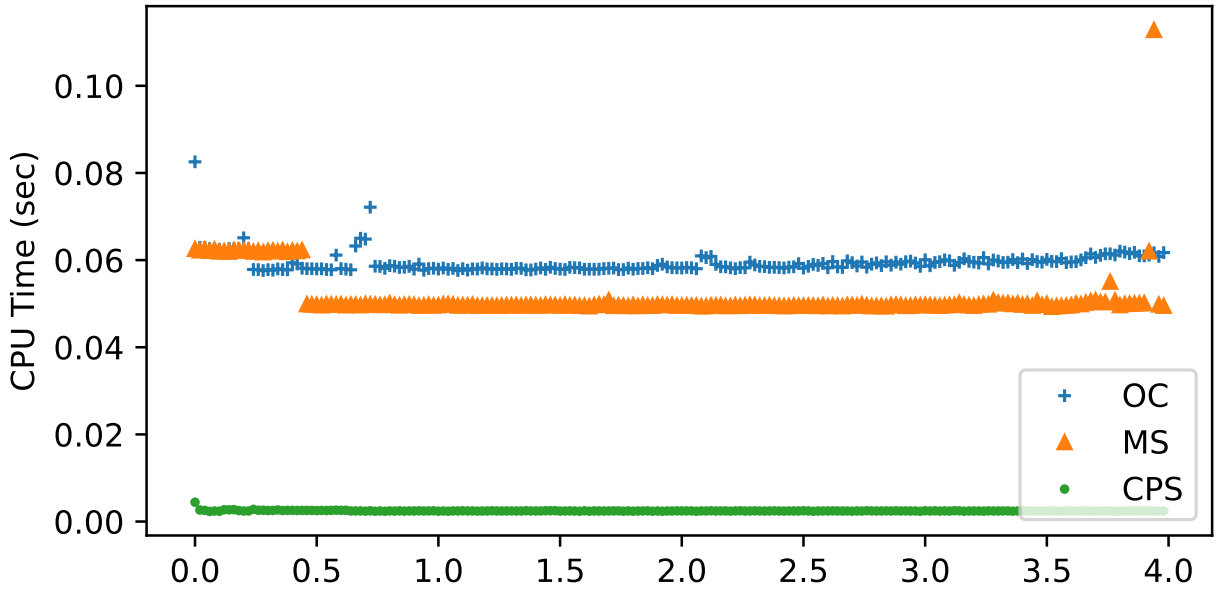


Figure 15: CPU time for simulation with initial state equal to goal state.

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

Raphaël Linsen, Petr Listov, Albéric de Lajarte, Roland Schwan, and Colin N. Jones. Optimal thrust vector control of an electric small-scale rocket prototype. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 1996–2002, 2022. doi: 10.1109/ICRA46639.2022.9811938.

Lukas Spannagl, Elias Hampp, Andrea Carron, Jerome Sieber, Carlo Pascucci, Aldo Zgraggen, Alexander Domahidi, and Melanie Zeilinger. Design, optimal guidance and control of a low-cost re-usable electric model rocket, 03 2021.

Carlos E. Garcia, David M. Prett, and Manfred Morari. Model predictive control: Theory and practice - a survey. *Autom.*, 25:335–348, 1989. URL `https://api.semanticscholar.org/CorpusID:3443742`.

Felix Fiedler, Benjamin Karg, Lukas Lüken, Dean Brandner, Moritz Heinlein, Felix Brabender, and Sergio Lucia. do-mpc: Towards fair nonlinear and robust model predictive control. *Control Engineering Practice*, 140:105676, 2023. ISSN 0967-0661. doi: https://doi.org/10.1016/j.conengprac.2023.105676. URL `https://www.sciencedirect.com/science/article/pii/S0967066123002459`.

Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 2018.

Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.*, 106(1):25–57, March 2006. ISSN 0025-5610.

HSL. A collection of fortran codes for large scale scientific computation., 2013. URL `http://www.hsl.rl.ac.uk`.