Quadrotor Trajectory Planning and Control via Sequential Convex Programming

Jushan Chen

***Abstract*—Existing trajectory optimization and planning methods often decouple the controller from the planning. In many trajectory planning literature, a simple kinematics model is used to model a quadrotor and an onboard trajectory tracking controller is assumed a priori. Using a simplified model reduces the problem's complexity but at the same time hinders the possibility for optimizing over motor command signals to achieve desired behavior. In this work, we propose an optimal trajectory planner that relies on a high-fidelity nonlinear model for a quadrotor and evaluates the performance of trajectory generation and control via sequential convex programming.**

***Index Terms*—Trajectory optimization, sequential convex programming**

I. Introduction

A. Motivations and prior works

In this project, our primary object of interest is quadrotor due to its maneuverability in complex environments and huge popularity among the academic research community. Our objective is to come up with efficient trajectory generation algorithms that can be run online on the hardware via optimization. Recent trajectory optimization methods such as [1]–[3] rely on a simple kinematics model and only use the acceleration vector as the decision variables, and thus the returned trajectories consist only of the velocities and positions; they assume a priori a reliable onboard trajectory tracking controller, so the use of a simple kinematics model is justified. In other words, the trajectory planning algorithm is completely decoupled from controlling each motor on the quadrotor to achieve stability. Traditional control methods on quadrotors can be found in the early works of [4] and [5], which proposed a nonlinear geometric tracking controller on $SE(3)$. Such controllers can roughly be classified as PID (proportional-integral-derivative) since they directly regulate the error dynamics in the attitude and position induced between the quadrotor and its desired target trajectory. Since such traditional control methods have been proposed, later trajectory planning algorithms became a separate optimization problem that largely ignores the real dynamics of the drone. In this work, rather than comping up with a completely new approach for controlling a quadrotor or a trajectory generation algorithm decoupled from controlling the quadrotor's motion, we explore the capability of sequential convex programming to couple trajectory generation and control of quadrotors directly. We choose the torques and forces of the quadrotor as optimization decision variables and applies optimization on a 12-degree-of-freedom nonlinear quadrotor dynamics model. This facilitates the capability of agile and complex maneuvers because these decision variables (torques and forces) can be directly transformed to the motor command signals on the hardware through a linear transformation [5].

Jushan Chen is with the Department of Aerospace Engineering, University of Illinois Urbana-Champaign, 104 S Wright St, Urbana, IL 61801, USA, {jushanc2@illinois.edu

B. Sequential convex programming

The trajectory planning problem we consider involves solving the trajectory tracking problem using optimization techniques. There are various existing categories of optimization algorithms, such as convex optimization, nonconvex optimization, etc. The primary objective of this work is to design and implement a sequential convex programming algorithm (SCP) for quadrotor trajectory planning.

Sequential convex programming can be categorized as a nonconvex optimization algorithm. Such methods rely on constructing convex sub-problems via successive linearizations of the original (nonlinear) objective functions and constraints. The resulting sub-problems become convex optimization subproblems, which can be typically solved by existing convex programming solvers, such as OSQP [6]. Moreover, we aim to apply our sequential convex programming in a receding-horizon fashion (synonymous with model predictive control), which re-optimizes (in the outer loop) the state and control trajectories at each discrete time step k over a future horizon of length T, until the quadrotor reaches its goal objective.

II. Problem Formulation

We consider a collision-free navigation task for the control and trajectory planning problem, where the quadrotor must navigate to its designated goal position efficiently. A general mathematical formulation of the optimization problem can be expressed as the following:

$$\begin{aligned} \min \quad & \int_0^{t_f} c(\boldsymbol{x}(t), \boldsymbol{u}(t), t) dt \\ \text{s.t.} \quad & \dot{\boldsymbol{x}}(t) = f_{k+1}(\boldsymbol{x}(t), \boldsymbol{u}(t), t), t \in [0, t_f] \\ & \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ & \boldsymbol{x}\left(t_f\right) = \boldsymbol{x}_f \\ & \boldsymbol{u}(t) \in \mathcal{U}, t \in [0, t_f] \end{aligned} \tag{1}$$

where $c(x(t), u(t), t)$ is the running cost, $u(t)$ is the control variable, $x(t)$ is the state vector, f refers to the nonlinear dynamics, $\mathcal{U}$ refers to the input constraint set, x_0 is the initial state condition, x_f is the final state condition, and t_f is the designated final time stamp. The problem in Eqn. 1 is then discretized and solved with $k = 0$ initially, and this process continues iteratively until convergence. In addition, it

is empirically found that adding a convex trust region ensures the algorithm's stability because it constrains the deviation between each successive iteration. For the discretized problem, the convex trust region constraint is expressed as:

$$||s_k - \bar{s}_k||_\infty \leq \rho, ||u_k - \bar{u}_k||_\infty \leq \rho \tag{2}$$

where $\bar{s}_k$ and $\bar{u}_k$ refer to the nominal state and input trajectory at the previous iterate at time step k. Note that the state trajectory array s and input trajectory array u at the very first iterate $q = 0$ are initialized by rolling out the dynamics of the model, and subsequently, these two arrays at the latest iterate $q + 1$ are replaced by the corresponding quantities at the previous iterate q and fed back into the current convex sub-problem. This process then repeats until convergence. The pseudocode [3] for a vanilla sequential convex programming algorithm is presented in Alg. 1

Algorithm 1: Single-agent SCP

Input : Initial state s_0, final state s_f
Output: trajectory specified by waypoints $s(t)$

$\bar{s}, \bar{u} \leftarrow initializeNominalTrajectory$;
while *not converged* **do**
  $R \leftarrow$ linearizeAllConstraints
  $QP \leftarrow$ formQP
  $u \leftarrow$ solve(QP)
  $s \leftarrow$ rolloutStates
return s

To further understand Alg. 1, we note that the SCP algorithm – though not obvious at first glance – consists of one outer loop and one inner loop, where the outer loop detects whether or not the algorithm should terminate, and the inner loop leverages a QP solver of convex optimization algorithm to solve each convex sub-problem. The trust region constraint in each convex sub-problem serves to constrain the deviation in the state and input trajectory between successive iterates in the outer loop, q and $q + 1$, $\forall\ 0 \leq q \leq N$. Each convex sub-problem is initialized using a nominal trajectory of length T rolled out by the discrete dynamics of the model (a forward pass), $x_{k+1} = f(x_k, u_k)$, $\forall\ 0 \leq k \leq T$

The expected measurable results are chosen as the following:

- The proposed algorithm should execute in a computationally tractable manner. The total run time of the optimizer at each time step k prediction control horizon T should be reasonably short, i.e., $T \times dt$ seconds or less, where dt is the time discretization step
- The trajectory generated by the planner should be safe, i.e. if multiple quadrotors are involved in the environment, they should avoid a collision at all time steps k. Thus, the pairwise Euclidean distance d_{ij} between agent i and agent j can be cast as a nonlinear constraint:

$$d_{ij} \geq R \tag{3}$$

  where R is a constant threshold distance.
- The quadrotor should be able to reach within a proximity distance from the target position upon the convergence of the algorithm, i.e., within 0.1 meters away from the target position

In our implementations, we replace the terminal state constraint $x(t_f) = x_f$ with a terminal cost $(x(t_f) - x_f)^T P(x(t_f) - x_f)$, where P is a positive definite matrix whose elements are orders of magnitude larger than Q. The rationale is that an exact terminal constraint is not necessary and a terminal cost should suffice our requirement for convergence (the agent reaches 0.1 m of the goal position). We impose an input constraint $u(t) \in \mathcal{U}$ at each time step such that the control input has both an upper bound and a lower bound on each of its elements, i.e. $u(t)$ is a vector of control inputs.

III. Simulation Results

A. Motivational example of controlling a nonlinear cartpole

To provide a motivational example for applying SCP to a nonlinear control problem, we now consider controlling the swing-up motion for a classic 2D cart-pole system that moves in a horizontal plane with a movable pole mounted on the cart. The configuration of the system is shown in Fig. 1. The state variable $s(t)$ consists of the horizontal position of the cart $x(t)$, the angle of the pole relative to the downright position $\theta(t)$, and the time rate of change of these two variables $\dot{x}(t)$, $\dot{\theta}(t)$. The control variable is the torque that actuates the pole on the cart. The initial position of the cartpole is at the origin $x_0 = 0$, and the initial angle of the pole is in the downright position, where θ = 0 rad. We use a quadratic running cost given as:

$$\begin{aligned} c(x_k, u_k) =& (x_k - x_{ref})^T Q (x_k - x_{ref}) \\ &+ (u_k - u_{ref})^T R (u_k - u_{ref}) \end{aligned} \tag{4}$$

where Q and R are semi-definite weight matrices, x_{ref} is the designated reference state, and u_{ref} is the designated reference input. The quadratic terminal cost can be written as:

$$c_T(x_T) = (x_T - x_{ref})^T Q_f (x_T - x_{ref}) \tag{5}$$

We choose Q, R, Q_f to be diagonal semi-definite matrices. The result of applying SCP is summarized in Fig. 2:

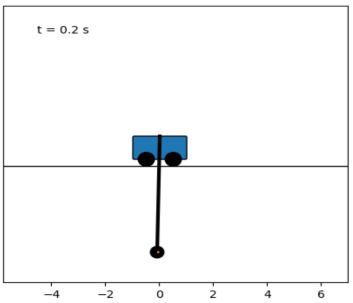

Fig. 1. Cartpole system

It is observed that at the end of the time span at t = 10, the angle θ reaches extremely close to π radians, which

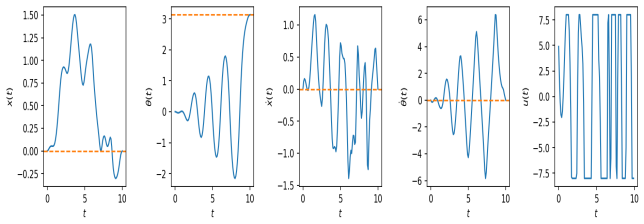

Fig. 2. Time evolution of state variables

Fig. 3. Crazyflie drone from Bitcraze

corresponds to the upright position. It is also noted that the input variable is upper and lower bounded by a scalar value, which satisfies the designated actuator constraint. [1]

B. Applying SCP to single-quadrotor trajectory generation

We apply the aforementioned SCP algorithm to the following 12-dimensional quadrotor model given in the following form:

$$\begin{aligned} \dot{x} &= v, \\ m\dot{v} &= mge_3 - fRe_3, \\ \dot{R} &= R\hat{\Omega}, \\ J\dot{\Omega} + \Omega \times J\Omega &= M \end{aligned} \tag{6}$$

where x is the position vector, v is the velocity, f is the net thrust, R is a rotation matrix from the body frame to the Earth frame, $\hat{\Omega}$ is the angular velocity in the body frame, and M is the total moment in the body frame [4], and J is the inertia matrix relative to the body frame. The exact model parameters are derived via system identification on a Crazyflie drone (see Fig. 3) and is given in Sec. VI. The complete closed-loop simulated trajectory using a one-shot SCP (over a single fixed-length horizon) is presented in Fig.4:

We observe that all state variables $x(t)$, $y(t)$, $z(t)$, $\psi(t)$, $\phi(t)$, $\theta(t)$, $v_x(t)$, $v_y(t)$, $v_z(t)$, $w_x(t)$, $w_y(t)$, $w_z(t)$ achieved asymptotic convergence upon their respective goal values. The trajectory of each state variable looks smooth. The 3-D position trajectory for the closed-loop simulation is given in Fig. 5 The final cost represents the numerical value of the

[1] https://github.com/labicon/distributed$_opt/tree/main/distributed_SCP$

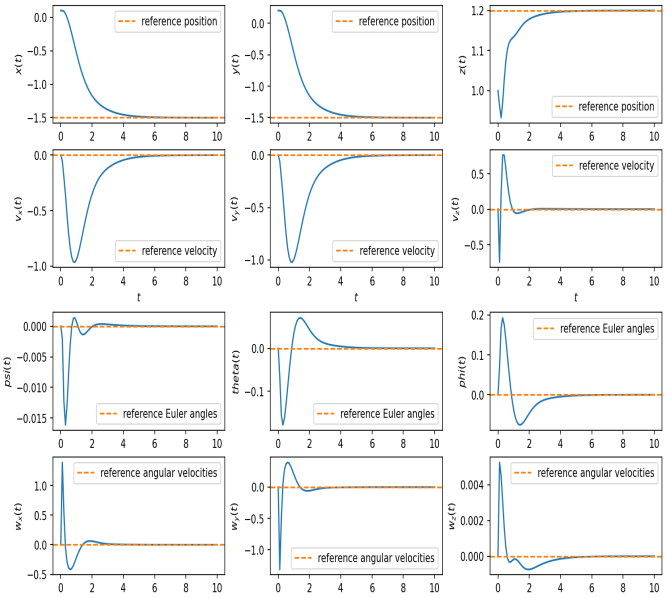

Fig. 4. State trajectory of a single drone

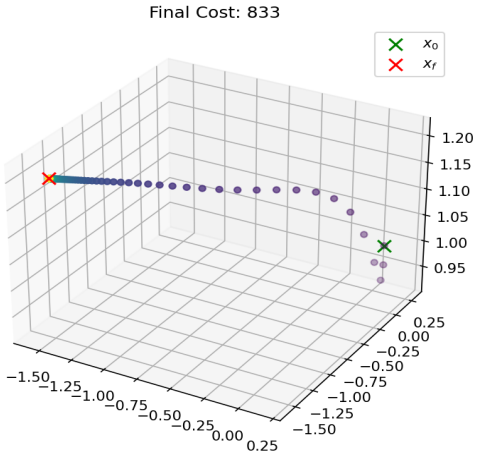

Fig. 5. 3-D trajectory of one-shot SCP (fixed-length horizon) for a single drone

overall quadratic cost at the final time step. The actual distance between the final position of the drone and its designated position is 4.1 $\times$ 10^{-5} m, orders of magnitude below the convergence threshold of 0.1 m. In addition, we show the control input trajectory returned by the SCP algorithm in Fig. 6

To further test the SCP algorithm, we run the simulation shown in Fig. 5 in a receding-horizon (RHC) fashion with a prediction horizon of length $T = 10$ and a discretization time step of dt = 0.1 s. The resulting 3-D trajectory is shown in Fig. 7 This time the final cost is a lot higher than the one-shot optimization in Fig. 5, and the distance between the final position and the goal position is 0.029 m. The time evolution of each state variable under the receding-horizon implementation is shown in Fig. 8

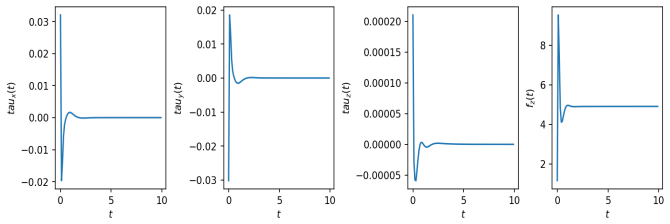

Fig. 6. Control input trajectory of a single drone. The upper and lower bounds of the control input are not shown because the returned values are well below the bounds.

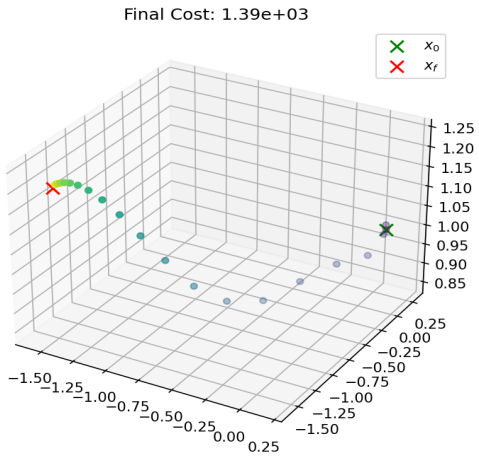

Fig. 7. Receding-horizon simulation for a single drone

By comparing the RHC implementation and one-shot optimization, we found that the RHC scheme is sub-optimal because the overall trajectory has a higher cost associated with it. This is an expected result because implementing RHC is a tradeoff between robustness to disturbance and optimality: when testing the algorithm in a real-world scenario, the one-shot optimization is highly susceptible to external disturbance, but RHC is more robust to such disturbances because it keeps re-optimizing at each time step. In addition, the RHC implementation can be executed online in real hardware testing, whereas the one-shot optimization has to follow a predetermined path without any position feedback at each time step. Our result shows that SCP can be done in an RHC setting without sacrificing much convergence performance.

C. Applying SCP to generate collision-avoidance trajectories

We further test the capability of our algorithm to handle collision avoidance when there is more than one quadrotor involved. SCP requires convex linearizations of all constraints. The original nonlinear collision avoidance constraint between any pair of agents i, j can be written as [3] :

$$\|\mathbf{p}_i[k] - \mathbf{p}_j[k]\|_2 \geq R \quad \forall i, j \quad i \neq j, \quad \forall k. \tag{7}$$

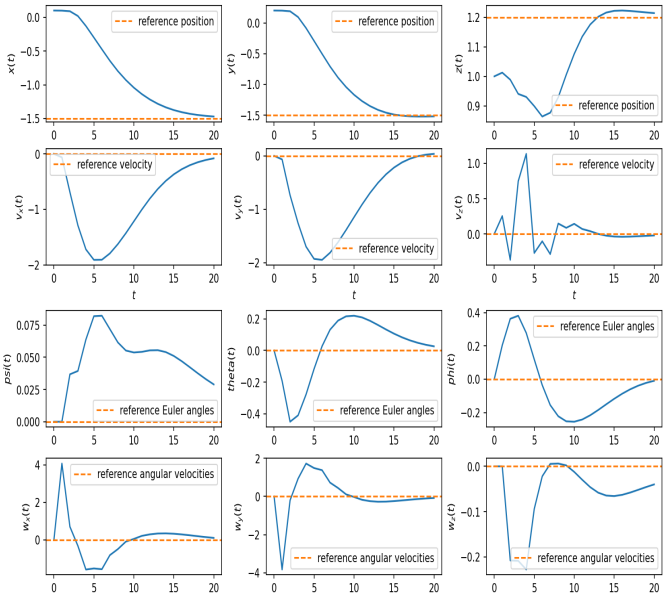

Fig. 8. State trajectory of a single drone in receding-horizon

where R is a collision threshold distance. We linearize Eqn .7 as the following:

$$\left\|\mathbf{p}_i^q[k] - \mathbf{p}_j^q[k]\right\| + \frac{\left(\mathbf{p}_i^q[k] - \mathbf{p}_j^q[k]\right)^T}{\left\|\mathbf{p}_i^q[k] - \mathbf{p}_j^q[k]\right\|} \left(\mathbf{p}_i[k] - \mathbf{p}_j[k]\right) \geq R. \tag{8}$$

where the superscript $p_i^q[k]$ refers to the position vector of agent i at time step k within the previous outer-loop iteration q. We run a 2-drone simulation in receding-horizon fashion with a prediction horizon of $T = 10$ and a collision radius of 0.35 m, and the initial and final positions of both drones are initialized randomly. The resulting trajectory is shown in Fig. 9: To verify that there is indeed no collision, we plot

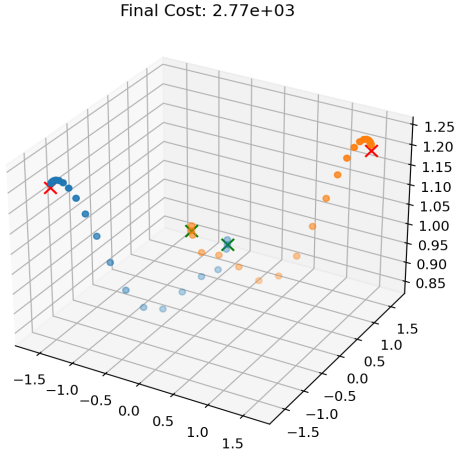

Fig. 9. Trajectory of 2-drone simulation

the distance between the two agents as shown in Fig. 10: We observe that no collision has occurred.

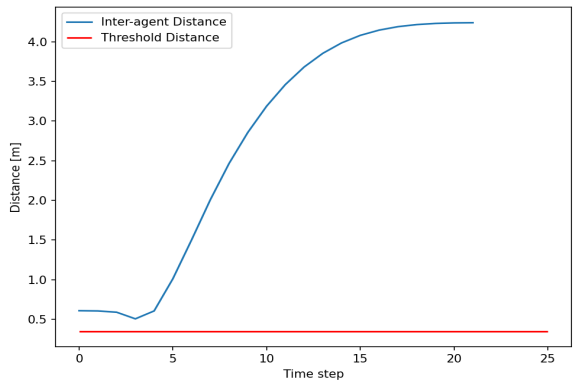

Fig. 10. Distance between 2 drones

D. Monte Carlo simulations

To accurately evaluate the performance of RHC-SCP, we run Monte Carlo simulations across several test cases ranging from 3 agents to 10 agents, and each test case is run 30 trials with randomized initial and final conditions. We record the average solve time at each time step time step $k \in [0, T]$ from each trial and then visualize the distributions of the solve time in a box plot fashion. The simulation is executed on a computer equipped with Intel i5-9600K CPU and 16GB of RAM. The result is presented in Fig. 11 The prediction horizon is chosen

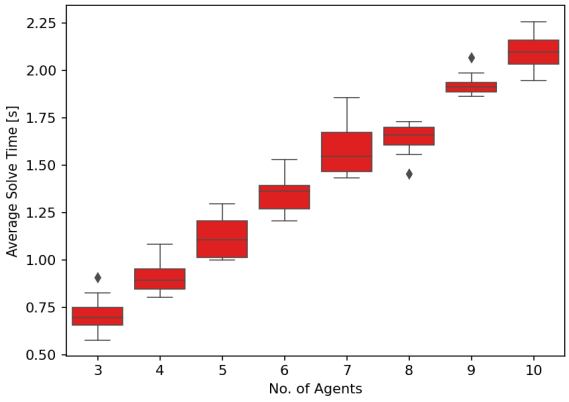

Fig. 11. Average solve time per time step for different test cases. Outliers are shown as diamonds in black

as $T = 10$, and the discretization time step is $dt = 0.1$, which yields a time duration of 1 second in reality per simulated time step k. Thus, for real-time online execution, we should require that the average solve time should be less than 1 second. It is observed from Fig. 11 that this requirement is satisfied when the number of agents is less than or equal to four; when there are more than four agents involved, the solve time exceeds our desired threshold of 1 second. However, from the trend shown in Fig. 11 we notice that the average solve time is approximately linear in the number of agents, and thus it is more scalable than nonlinear MPC methods that require $O(n_x^3)$ flops, where n_x is the dimension of the state space.

We also record the convergence probability for each test case: if the solution is feasible and the drone reaches its convergence criterion, it is a success; otherwise, it is a failure. The result is shown in Fig .12 When the number of agents is

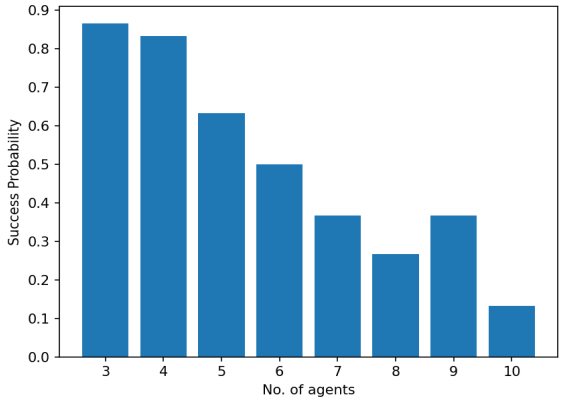

Fig. 12. Convergence probability for each test case

less than or equal to four, the convergence probability is above 80%. However, it drops significantly as the number of agents increases.

IV. Coupling between trajectory generation and control

Once the optimized trajectories are obtained, the corresponding control signals can be obtained in a straightforward manner. The control signals are different from the control input we use to model the drone dynamics as shown in Eqn. 6 because the actual control inputs from the hardware control's perspective are the motor command signals (pulse width modulations). We denote the motor power commands as m_1, m_2, m_3, and m_4 since there are four independent motors on a quadrotor. It is known that there exists a linear mapping P that maps the motor power commands to the input vector of the quadrotor dynamics $\tau_x, \tau_y, \tau_z, f_z$. P is given as a matrix :

$$\begin{bmatrix} -k_F l & -k_F l & k_F l & k_F l \\ -k_F l & k_F l & k_F l & -k_F l \\ -k_M & k_M & -k_M & k_M \\ k_F & k_F & k_F & k_F \end{bmatrix} \tag{9}$$

where l is the half-body length of the drone, and k_F, k_M are force and torque coefficients that are obtained empirically. To obtain the motor commands m_1, m_2, m_3, and m_4 from $\tau_x, \tau_y, \tau_z, f_z$, we compute the inverse of P as:

$$\begin{bmatrix} -\frac{1}{4k_F l} & -\frac{1}{4k_F l} & -\frac{1}{4k_M} & \frac{1}{4k_F} \\ -\frac{1}{4k_F l} & \frac{1}{4k_F l} & \frac{1}{4k_M} & \frac{1}{4k_F} \\ \frac{1}{4k_F l} & \frac{1}{4k_F l} & -\frac{1}{4k_M} & \frac{1}{4k_F} \\ \frac{1}{4k_F l} & -\frac{1}{4k_F l} & \frac{1}{4k_M} & \frac{1}{4k_F} \end{bmatrix} \tag{10}$$

Thus, we obtain the following expression for the actual motor commands:

$$\begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix} = \begin{bmatrix} -\frac{1}{4k_F l} & -\frac{1}{4k_F l} & -\frac{1}{4k_M} & \frac{1}{4k_F} \\ -\frac{1}{4k_F l} & \frac{1}{4k_F l} & \frac{1}{4k_M} & \frac{1}{4k_F} \\ \frac{1}{4k_F l} & \frac{1}{4k_F l} & -\frac{1}{4k_M} & \frac{1}{4k_F} \\ \frac{1}{4k_F l} & -\frac{1}{4k_F l} & \frac{1}{4k_M} & \frac{1}{4k_F} \end{bmatrix} \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \\ f_z \end{bmatrix} \tag{11}$$

Since our SCP algorithm 1 is considered a direct optimization where both control trajectory and state trajectory are returned, we simply need to apply the linear transformation in Eqn. 11 to the optimized control input u to get the actual motor commands. The motor command signals will be programmed into the firmware code that is embedded into the Crazyflie drone. Hardware testing is not performed in this project, but interested readers are encouraged to consult official resources from Crazyflie.

In reality, drift might be accumulated when the algorithm is implemented online in real time due to external disturbances. However, running in a receding horizon can help offset some influence of that disturbance.

V. Conclusion

Summary. We have demonstrated that a generic sequential convex programming approach can be applied to nonlinear cart pole control and quadrotor trajectory generation. Through Monte Carlo simulations, we have found that the computation required scales linearly with the number of agents.

Limitations. From the empirical results we collected, we note that the scalability of our sequential convex programming algorithm is limited: the convergence probability significantly drops when the number of agents exceeds four. For future work, this can be alleviated by making the algorithm decentralized such that each agent solves its own optimization problem in a parallel fashion.

References

[1] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," pp. 1917–1922, 2012.

[2] C. E. Luis and A. P. Schoellig, "Trajectory generation for multiagent point-to-point transitions via distributed model predictive control," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 375–382, 2019.

[3] Y. Chen, M. Cutler, and J. P. How, "Decoupled multiagent path planning via incremental sequential convex programming," pp. 5954–5961, 2015.

[4] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se(3)," pp. 5420–5425, 2010.

[5] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," pp. 2520–2525, 2011.

[6] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

VI. Appendix

To derive the full equations of motion, the mass is given as 0.5 kg, the moment of inertia along the x body axis is J_x = 0.0023 kgm^2, and similarly J_y = 0.0023 kgm^2, J_z = 0.0040 kgm^2, and g = 9.81 $\frac{m}{s^2}$. After substituting all the parameters into the equations of motion above, we obtain the following:

$$\begin{bmatrix} v_x\cos(\psi)\cos(\theta) \\ +\, v_y(\sin(\phi)\sin(\theta)\cos(\psi) - \sin(\psi)\cos(\phi)) \\ +\, v_z(\sin(\phi)\sin(\psi) + \sin(\theta)\cos(\phi)\cos(\psi)) \\ v_x\sin(\psi)\cos(\theta) \\ +\, v_y(\sin(\phi)\sin(\psi)\sin(\theta) + \cos(\phi)\cos(\psi)) \\ -\, v_z(\sin(\phi)\cos(\psi) - \sin(\psi)\sin(\theta)\cos(\phi)) \\ -\, v_x\sin(\theta) \\ +\, v_y\sin(\phi)\cos(\theta) \\ +\, v_z\cos(\phi)\cos(\theta) \\ \frac{w_y\sin(\phi)+w_z\cos(\phi)}{\cos(\theta)} \\ w_y\cos(\phi) - w_z\sin(\phi) \\ w_x + w_y\sin(\phi)\tan(\theta) + w_z\cos(\phi)\tan(\theta) \\ v_y w_z - v_z w_y + \frac{981\sin(\theta)}{100} \\ -v_x w_z + v_z w_x - \frac{981\sin(\phi)\cos(\theta)}{100} \\ 2f_z + v_x w_y - v_y w_x - \frac{981\cos(\phi)\cos(\theta)}{100} \\ \frac{10000\tau_x}{23} - \frac{17 w_y w_z}{23} \\ \frac{10000\tau_y}{23} + \frac{17 w_x w_z}{23} \\ 250\tau_z \end{bmatrix}$$