Nonlinear Control and Planning in Robotics

Final Project

Planning and control of a quadrotor

in 3-D among obstacles

Soowon Kim, Hyungmu Lee

Johns Hopkins University

Abstract
In case of catastrophes such as earthquake, flood, it is difficult to reach out to find people quickly due to the uncertainty of the environment. Recently, quad-helicopter is more researched, and it is getting an alternative way to use it in catastrophes. In this project, we present planning and control of a quadrotor in 3-D among obstacles. Initializing starting point and destination point, we utilize Dijkstra and A* algorithm to find the shortest path in a map. On top of that, we also use the minimum snap-trajectory which is used to 9th polynomial and 4th derivative cost minimization. In this trajectory process, we emulate the time allocation by using gradient descent and add a task about collision test so that we are able to obtain an optimized path with lowest energy cost. Within geometric tracking control of a quadrotor UAV on SE(3) by Taeyoung Lee, we design a controller to track our desired-trajectory perfectly. This project demonstrates the possibility of use of quadrotors based on an optimized way to design a trajectory and track it under low-energy cost.

Introduction
Reconnaissance, monitoring, and gathering other information in a disaster scene is crucial and must be done in the timeliest manner possible. Often, the physicality of a quadrotor for this scenario is also limited due to the cluttered and dense environments. Therefore, from mapping, trajectory generation, to execution using robust controller must be done in a matter of seconds only using onboard computer. Among these challenges, the trajectory planning can be addressed by exploiting the fact that the system, in this case, a quadrotor, has the property of differential flatness, in other words, the state and input space of the dynamical system can be mapped into a set of flat output variables and their derivatives. In conjunction with utilizing polynomial basis functions, it enables fast algebraic calculation of the trajectory and the control inputs. Additionally, since polynomial trajectories are differentiable, the continuity of derivatives between two connected trajectories is obtainable. This fact is especially important since a non-smooth, or discontinuous trajectories can require the control inputs to be unnecessarily high. This trajectory planning technique is used in the work of Bry, A. et al [1], and Mellinger and Kumar [2].

The geometric controller which is utilizing geometric manifold, special Euclidean group, SE(3) can be used for the trajectory tracking. This approach demonstrated an acrobatic maneuver of a quadrotor with exponential stability when the attitude error is less than 90° [3]. Feedback linearization and robust back-stepping will be explored as well.

Method
A. Dijkstra and A algorithm*

Dijkstra's original variant found the shortest path between two nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree. Moreover, Dijkstra's algorithm uses a heuristic identically equal to zero [4].

The A* algorithm is a generalization of Dijkstra's algorithm that cuts down on the size of the subgraph that must be explored, if additional information is available that provides a lower bound on the "distance" to the target [5].

Here, we modified path route to more optimized route by decreasing points when there is no collision in between 2 nodes.

B. Minimum snap polynomial trajectory optimization

Consider a polynomial P of degree 9 such that

$$\mathrm{P(t)} = p_9 t^9 + p_8 t^8 + \cdots + p_1 t + p_0$$

Since input u_2 and u_3 have up to 4th derivative (snap) of the positions, we are interested in optimizing P by minimizing snap. Cost function J can be written as

$$\mathrm{J} = \int_0^\tau P^{(4)}(t)^2 dt = \mathbf{p}^T \mathbf{Q} \mathbf{p}$$

Where $\mathbf{p}$ is a column vector of the coefficients of the polynomial and $\mathbf{Q}$ is a cost matrix. Now we present equality constraints in a form of

$$\mathbf{A}\mathbf{p} = \mathbf{b}, \mathbf{b} = \left[x_0, \dot{x}_0, \ddot{x}_0, \dddot{x}_0, x_0^{(4)} \;\middle|\; x_\tau, \dot{x}_\tau, \ddot{x}_\tau, \dddot{x}_\tau, x_\tau^{(4)} \right]$$

Now we have a standard quadratic programming (QP).

C. Piecewise polynomial joint optimization

In general, multiple segments of the trajectory will be need. We will jointly optimize all polynomial segments at once by minimizing total sum of all cost function.

$$\mathrm{J_{tot}} = J_1 + \cdots + J_K = \begin{bmatrix} \mathbf{p_1} \\ \vdots \\ \mathbf{p_K} \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \mathbf{Q_1} & & \\ & \ddots & \\ & & \mathbf{Q_K} \end{bmatrix} \begin{bmatrix} \mathbf{p_1} \\ \vdots \\ \mathbf{p_K} \end{bmatrix}$$

With equality constraints

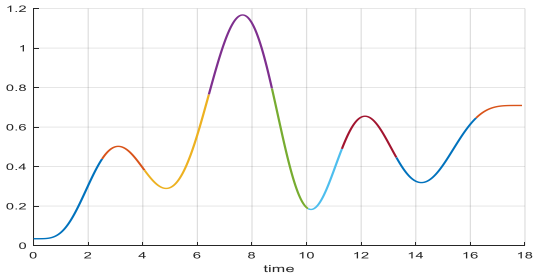
$$\begin{bmatrix} \mathbf{A_1} & & \\ & \ddots & \\ & & \mathbf{A_K} \end{bmatrix} \begin{bmatrix} \mathbf{p_1} \\ \vdots \\ \mathbf{p_K} \end{bmatrix} = \begin{bmatrix} \mathbf{b_1} \\ \vdots \\ \mathbf{b_K} \end{bmatrix}$$

In general, the positions and derivative terms are give at the start and end points. Only the positions are given at each waypoint, derivative terms are subjected to be optimized while ensuing the continuity so that all inputs are guaranteed to be continuous.

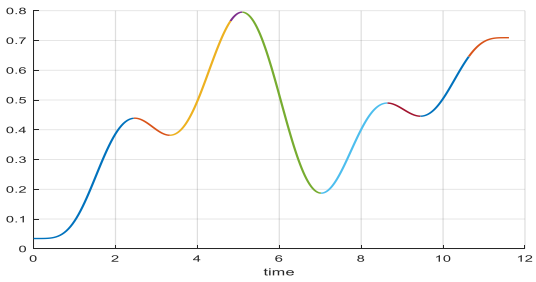
Constrained QP is prone to be ill conditioned when the dimension of a cost matrix gets large. Here, the dimension is 10K. To avoid this problem, we use an unconstrained QP by replacing $\mathbf{p}$ in the cost function with equality constraints, $\mathbf{A^{-1}b}$.

$$J_{tot} = \begin{bmatrix} \mathbf{b_1} \\ \vdots \\ \mathbf{b_K} \end{bmatrix}^T \begin{bmatrix} \mathbf{A_1} & & \\ & \ddots & \\ & & \mathbf{A_K} \end{bmatrix}^{-T} \begin{bmatrix} \mathbf{Q_1} & & \\ & \ddots & \\ & & \mathbf{Q_K} \end{bmatrix} \begin{bmatrix} \mathbf{A_1} & & \\ & \ddots & \\ & & \mathbf{A_K} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{b_1} \\ \vdots \\ \mathbf{b_K} \end{bmatrix}$$

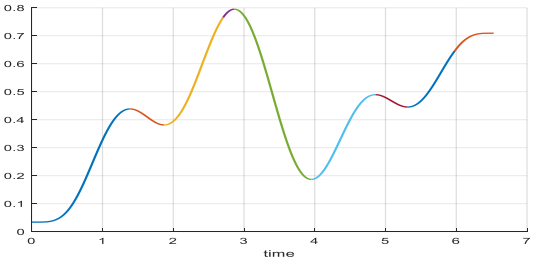
Now the decision variables are derivatives, where we already know some of elements including the positions



(a) Initial guess of τ



(b) Time allocation with $c_\tau = 50$



(c) Time allocation with $c_\tau = 5000$

Fig. 1 Path with 9 segments is generated randomly (from 0 to 1), as well as time interval, τ (from 1sec to 3sec).

at each waypoint and continuity. This fact can be exploited by rearranging the cost matrix so that known terms and unknown terms are separated.

$$J_{tot} = \begin{bmatrix} b_{fix} \\ b_{free} \end{bmatrix}^T CA^{-T}QA^{-1}C^T \begin{bmatrix} b_{fix} \\ b_{free} \end{bmatrix} = \begin{bmatrix} b_{fix} \\ b_{free} \end{bmatrix}^T \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix} \begin{bmatrix} b_{fix} \\ b_{free} \end{bmatrix}$$

By taking derivative of J_{tot} and set it equal to zero gives us minimum value, i.e. optimized b_{free}.

$$b^*_{free} = -\mathrm{R}_{22}^{-1}R_{12}^T b_{fix}$$

Optimized derivatives are reordered by applying $\mathbf{C}^{-1}$, then mapped back to get **p**. Constrained QP rendered 0% success rate with more than 5 segments in our case. Using an unconstrained QP successfully generated optimized values with even more than 30 segments.

C. Time allocation

Time interval on each segment will be optimized as well since we do not know the proper time to reach to the destination. This can be done by penalizing the total time, adding this penalty to the cost function.

$$\mathrm{J_{tot}} = p^T Q p + c_\tau \sum_{i=1}^{K} \tau_i$$

Now we use gradient descent method to update τ until it reaches to the minimum $\mathrm{J_{tot}}$. In figure, initial guess of

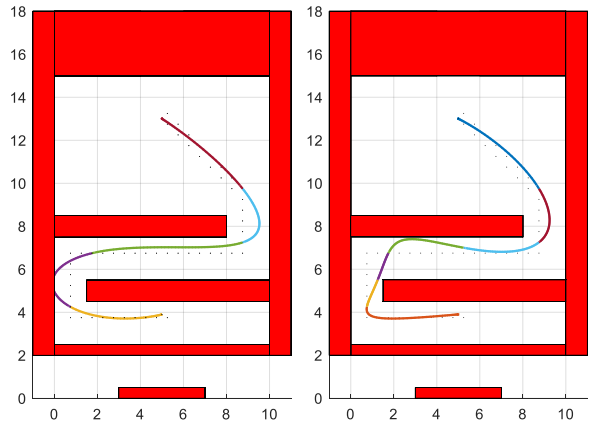


Fig. 2 Left trajectory intersects with an obstacle. After adding mid-way point, the trajectory becomes collision free (Right)

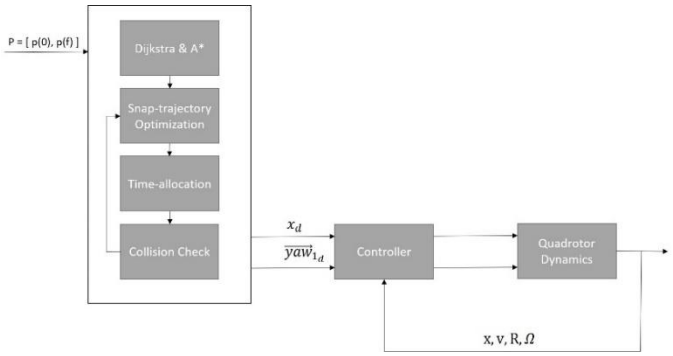


Fig. 3 The overall algorithm structure

time interval causes overshoot between two end points of each segment. After using time allocation, none of the segments overshoot between their two end points. Since penalty c_τ only penalizes the total time, the ratio of optimized each time interval remains the same as shown in fig. 1.

D. Collision free trajectory

If a certain segment intersects with an obstacle, a mid-way point is added, then new trajectory is made. fig. 2. shows the demonstration of adding new waypoints.

Dynamics

Complete state consists of the position and velocity of the center of mass and the orientation, which are parameterized by Euler angles, and the body angular velocity, ω^b.

$$\mathbf{x} = [x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, p, q, r]$$

Newton's equations of motion and Euler equation are

$$\dot{\boldsymbol{x}} = \boldsymbol{v}$$
$$m\dot{\boldsymbol{v}} = -mg\boldsymbol{z}_w + u_1\boldsymbol{z}_b$$
$$\dot{R} = R\hat{\omega}^b$$
$$I\dot{\omega}^b = -\omega^b \times I\omega^b + M$$

Thrust force generated by each rotor always have the direction same as body z-axis, $\boldsymbol{z}_b$. $R(\phi, \theta, \psi)$ is a

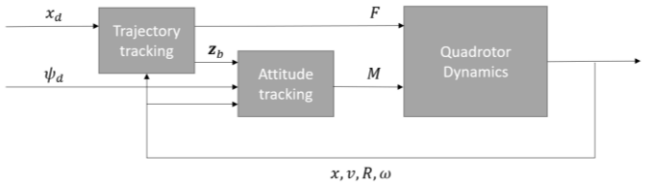


Fig. 4 Controller structure

rotation matrix with Z-X-Y Euler angles, yaw, pitch, and roll. We will use a geometric controller [3] since the attitude error function is defined within SO(3), avoiding singularities of Euler angles and ambiguities presented in quaternion. The overall controller structure is illustrated in fig. 4.

Desired position trajectories that are generated from previous section is fed into the trajectory tracking controller to track positions and compute $\boldsymbol{z}_{b,des}$. Then $\boldsymbol{z}_{b,des}$ and ψ_d are used to track attitude.
First, we define position and velocity error as

$$\mathbf{e}_\mathrm{p} = \boldsymbol{x} - \boldsymbol{x}_d$$
$$\mathbf{e}_\mathrm{v} = \dot{\boldsymbol{x}} - \dot{\boldsymbol{x}}_d$$

Then desired force vector is defined as

$$\boldsymbol{F}_{des} = -k_p \boldsymbol{e}_p - k_v \boldsymbol{e}_v + mg\boldsymbol{z}_w + m\ddot{\boldsymbol{x}}_d$$

The desired direction of the force is desired body z-axis.

$$\boldsymbol{z}_{b,des} = \frac{\boldsymbol{F}_{des}}{\|\boldsymbol{F}_{des}\|}$$

Using the fact that the first input u_1 is always aligned with z_b, we can compute u_1.

$$u_1 = \boldsymbol{F}_{des} \cdot \boldsymbol{z}_b$$

Desired body x and y-axis can be computed by knowing desired yaw angle and $\boldsymbol{z}_{b,des}$.

$$\boldsymbol{x}_{c,des} = [c\psi \; s\psi \; 0]^T$$
$$\boldsymbol{y}_{b,des} = \frac{\boldsymbol{z}_{b,des} \times \boldsymbol{x}_{c,des}}{\|\boldsymbol{z}_{b,des} \times \boldsymbol{x}_{c,des}\|}, \boldsymbol{x}_{b,des} = \boldsymbol{y}_{b,des} \times \boldsymbol{z}_{b,des}$$

Then the desired rotation matrix can be simply given by

$$R_{des} = [\boldsymbol{x}_{c,des}, \boldsymbol{y}_{b,des}, \boldsymbol{z}_{b,des}]$$

The error function is defined as

$$\Psi(R, R_{des}) = \frac{1}{2}\mathrm{tr}(I_3 - R_{des}^T R)$$

Then the attitude tracking error e_R is chosen to be

$$e_R = \frac{1}{2}(R_{des}^T R - R^T R_{des})^\vee$$

To define the angular velocity error, they must be compared in the same tangent plane in SO(3).

$$e_\omega = \omega^b - R^T R_{des} \omega_{des}^b$$

The control input M is chosen as

$$M = -k_R e_R - k_\omega e_\omega + \omega^b \times I\omega^b - I(\hat{\omega}^b R^T R_{des} \omega_{des}^b - R^T R_{des} \dot{\omega}_{des}^b)$$

Where

$$\omega_{des}^b = (R_{des}^T \dot{R}_{des})^\vee$$
$$\dot{\omega}_{des}^b = (\dot{R}_{des}^T \dot{R}_{des} + R_{des}^T \ddot{R}_{des})^\vee$$

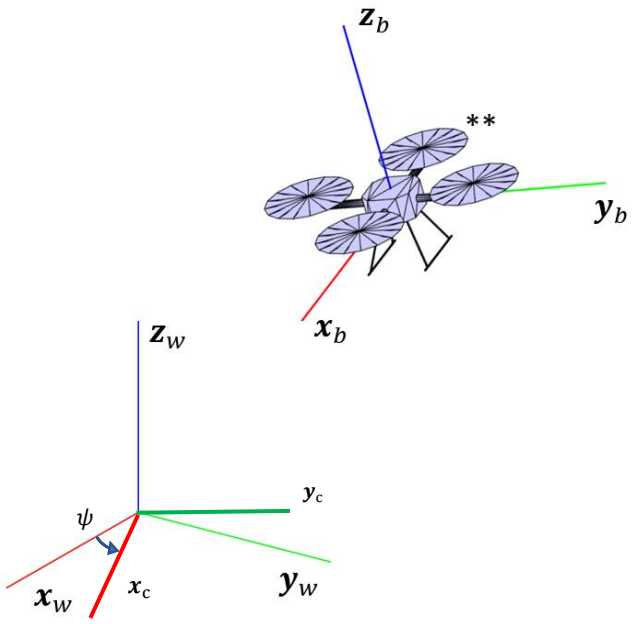


Fig. 5 Quadrotor model

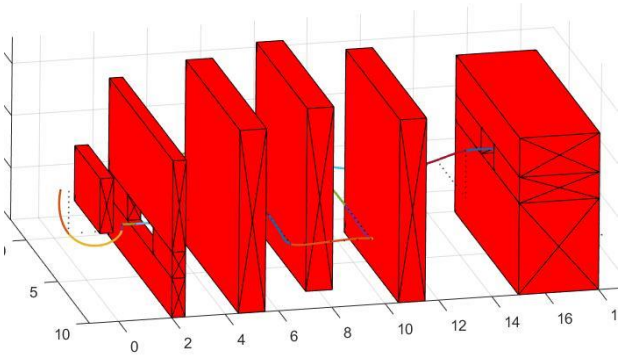
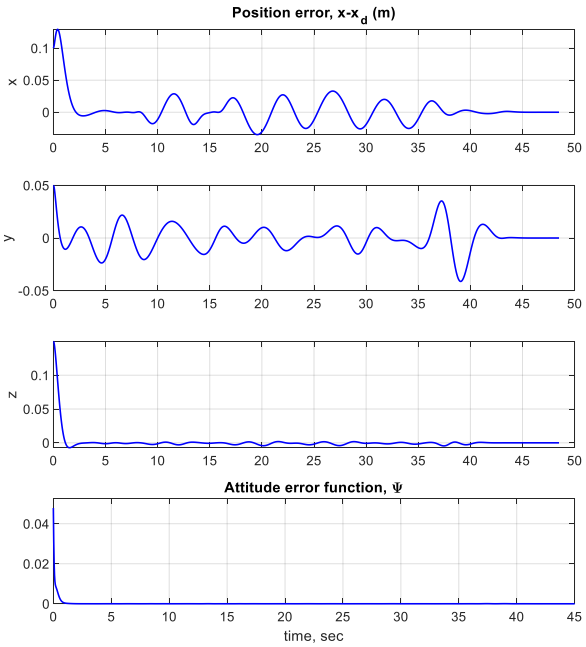


Fig. 6 3D structure of multiple obstacles

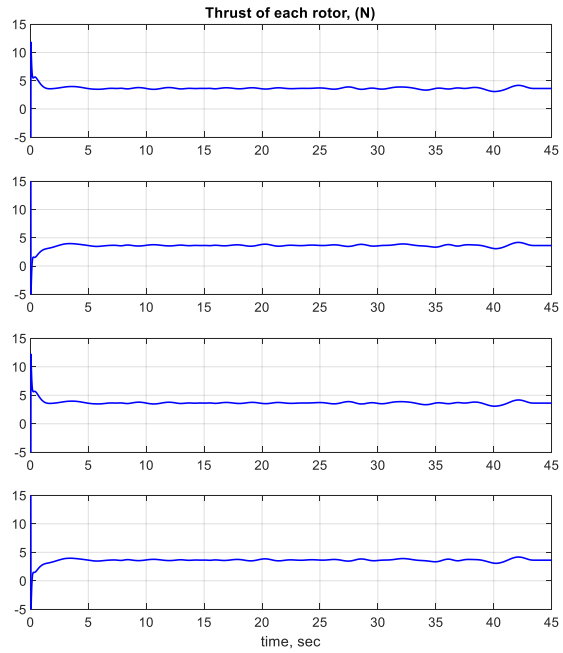
Using this controller guarantees asymptotic stability when the gain values are properly chosen. Details about proof can be found in [3].

Implementation

A platform to create 3D environment with obstacles [6] is used to create a dense terrain including a small window with shallow depth, three wide blocks standing in zigzag pattern, and a small window with deep depth as shown in fig. 6. We used Dijkstra algorithm, given by [6], to generate shortest opened waypoints once we feed start and end points into the algorithm. Start and end points are given as
start = [5.0 -1 3.5]; stop = [5.0 19.0 .5];
We first reduce the number of waypoints by discarding intermediate points which do not intersect with an obstacle.
With given path (start, way, and end points) polynomial optimization algorithm, which is directly implemented using the method in [1],[2], is initialized by initial guess of τ by randomly choosing the value ranging from 1 sec

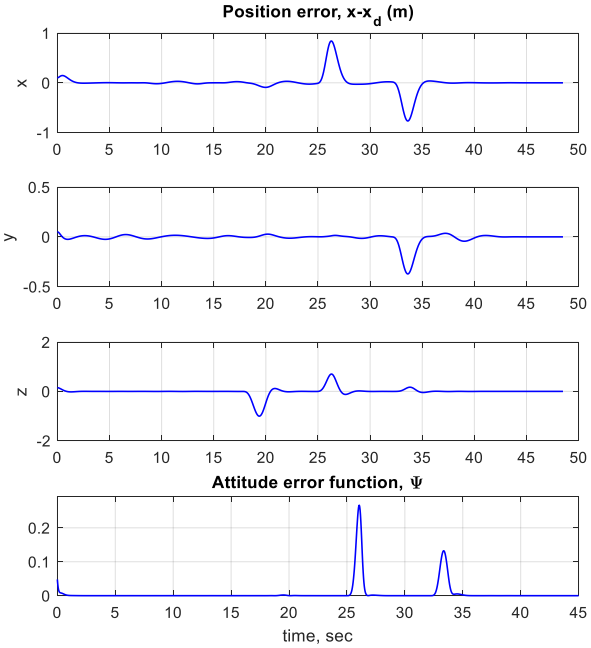


(a) Position error, e_p (m) and attitude error function, Ψ

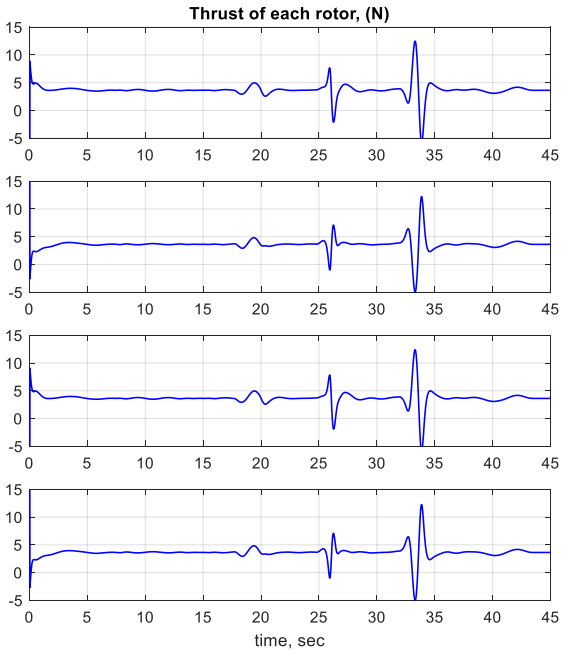


(b) Thrust of each rotor (N)

Fig. 7 Case 1. without external force.



(a) Position error, e_p (m) and attitude error function, Ψ



(b) Thrust of each rotor (N)

Fig. 8 Case 2. with external force.

to 3 sec. Then time allocation algorithm is used to update τ using gradient descent in a way that

$$\tau_{\text{new}} = \tau_{old} - \alpha 2^{m-1} \frac{\nabla J}{\|\nabla J\|}, \alpha = 10^{-6}$$

Positive integer m is increased while $J_{\tau_{new}} < J_{\tau_{old}}$. The stopping criteria is when the difference of 5 latest costs are less than 1% of the latest cost.

Once time is optimized, it checks collisions, which the algorithm is provided from [6] and iterates until the trajectory is collision free. Specifically, we use so called *inflated* map, where the obstacles are inflated so that it takes into account of the volume of the quadrotor. The resulting margin is 25 cm on xy direction and 15 cm on z direction.

The dynamics and the controller is directly implemented with the method in [3]. We need a mapping to get derivatives of Euler angles to have a complete differential equation of the current state.

$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}c\theta & 0 & -c\phi s\theta\\ 0 & 1 & s\phi\\ s\theta & 0 & c\phi c\theta\end{bmatrix}^{-1}\begin{bmatrix}p\\ q\\ r\end{bmatrix}$$

Result

The physical parameters of the quadrotors are given

$$\mathrm{m} = 1.477\ \mathrm{kg}, d = 0.263\ \mathrm{m}, \mathrm{c}_{\tau f} = 8.004 \times 10^{-4} m$$

$$I = [0.01152;\ 0.01152;\ 0.0218]\ \mathrm{kgm}^2$$

The gain values and time penalty are chosen as

$$\mathrm{kp} = 11.9, \mathrm{kv} = 4.443, \mathrm{k_R} = 10, \mathrm{k_\omega} = 6, \mathrm{c_\tau} = 50$$

Initial conditions are chosen including initial disturbances

$$\mathrm{x}_0 = [5.0\ \ -1\ 3.5] + [.1\ .05\ .15]$$

$$\dot{\mathrm{x}}_0 = [0\ 0\ 0] + [.01\ .01\ .02]$$

$$[\phi_0\ \theta_0\ \psi_0] = [0\ 0\ 0] + \left[\frac{\pi}{18}\ \frac{\pi}{18}\ \frac{\pi}{18}\right]$$

$$\omega_0^{\mathrm{b}} = [0\ 0\ 0]$$

Simulation results are presented in fig. 7, fig. 8. Without disturbances, as shown in fig. 7, after escaping initial disturbances, the maximum position error is less than 5 cm and with attitude error function, they go to zero. Maximum attitude error was less than 0.05. Even with external force disturbances, both position and attitude error reduced to zero as shown in fig. 8. External forces are chosen as

$$17.7\ \mathrm{sec} < \mathrm{t} < 20\ \mathrm{sec}, \mathrm{f}_1 = [0\ 0\ -6]\ (\mathrm{N})$$

$$24.8\ \mathrm{sec} < \mathrm{t} < 26.5\ \mathrm{sec}, \mathrm{f}_2 = [3\ 0\ 3]\ (\mathrm{N})$$

$$32\ \mathrm{sec} < \mathrm{t} < 34\ \mathrm{sec}, \mathrm{f}_3 = [-3\ \ -1.5\ 0]\ (\mathrm{N})$$

Reference

[1] Bry, A., Richter, C., Bachrach, A., & Roy, N. (2015). Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments. The International Journal of Robotics Research,34(7), 969-1002.

[2] Mellinger, D., & Kumar, V. (2011). Minimum snap trajectory generation and control for quadrotors. 2011 IEEE International Conference on Robotics and Automation.

[3] Lee, T., Leok, M., & Mcclamroch, N. H. (2010). Geometric tracking control of a quadrotor UAV on SE(3). 49th IEEE Conference on Decision and Control (CDC).

[4] Fredman, Michael Lawrence; Tarjan, Robert E. (1984). Fibonacci heaps and their uses in

improved network optimization algorithms. 25th Annual Symposium on Foundations of

Computer Science. IEEE. pp. 338–346

[5] Wikipedia Dijkstra's Algorithm, https://en.wikipedia.org/wiki_Dijkstra%27s

algorithm#cite note-Dijkstra1959-3

[6] S. (n.d.). Stormmax/quadrotor. Retrieved from https://github.com/stormmax/quadrotor

[7] T. (2017, March 05). Tu-darmstadt-ros-pkg/hector_quadrotor. Retrieved from https://github.com/tu-darmstadt-ros-pkg/hector_quadrotor

[8] Sreenath, K., Lee, T., & Kumar, V. (2013). Geometric control and differential flatness of a quadrotor UAV with a cable-suspended load. 52nd IEEE Conference on Decision and Control.

[9] Murray, R. M. (2017). Mathematical introduction to robotic manipulation. Taylor & Francis.