2023 European Control Conference (ECC)
June 13-16, 2023. Bucharest, Romania

Quadcopter Tracking Using Euler-Angle-Free Flatness-Based Control

Aeris El Asslouj[1] and Hossein Rastgoftar[1,2]

Abstract— **Quadcopter trajectory tracking control has been extensively investigated and implemented in the past. Available controls mostly use the Euler angle standards to describe the quadcopter's rotational kinematics and dynamics. As a result, the same rotation can be translated into different roll, pitch, and yaw angles because there are multiple Euler angle standards for the characterization of rotation in a 3-dimensional motion space. To address this issue, this paper will develop a flatness-based trajectory tracking control without using Euler angles. We assess and test the proposed control's performance in the Gazebo simulation environment and contrast its functionality with the existing Mellinger controller, which has been widely adopted by the robotics and unmanned aerial system (UAS) communities. Our simulations also show that, for both controllers, the main cause of loss of stability is not the theoretical domain of stability, but it is instead the inability of quadcopter rotors to provide negative thrust as is requested by controllers for aggressive trajectories.**

I. Introduction

Over the past few decades, multi-copter UAVs have been used for a variety of purposes, including crop management [1], rescue and disaster relief missions [2], aerial payload transport [3], [4], surveillance [5], piping inspections [6]. Trajectory tracking control of multi-copters have been extensively investigated by the researchers and multiple position-yaw controllers have been proposed. These include the cascaded Proportional–Integral–Derivative (PID) controller [7], the Incremental Nonlinear Dynamic Inversion (INDI) controller [8], and the Mellinger controller [9]. Additionally, we recently developed a feed-back linearization-based control for quadcopter trajectory tracking [10], [11] which is called "Snap" controller in this paper. All these controllers are based intentionally or not on the concept of differential flatness [12] which can facilitate designing controllers for non-linear systems.

The Mellinger controller is somewhat of an outlier in this list by the fact that it is able to follow aggressive trajectory far from the hover state while being simple and small. Meanwhile, the Snap controller offers a considerably wider domain of attraction as compared to the Mellinger controller, with the stability margin that is constrained to specific initial conditions [9], [10]. While these two controllers solve the same control problem using differential flatness, they are polar opposite solutions, each with their advantages and disadvantages.

*This work has been supported by the National Science Foundation under Award Nos. 2133690 and 1914581.

[1]Aeris El Asslouj is with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, Arizona, USA `aymaneelasslouj@arizona.edu`

[2]Hossein Rastgoftar is with with the Departments of Aerospace & Mechanical Engineering and Electrical & Computer Engineering, University of Arizona, Tucson, Arizona, USA `hrastgoftar@arizona.edu`

This paper compares the functionality of the Snap and Mellinger controllers. Compared to the authors' previous work and existing literature, this paper offers the following main contributions:

1) Rotation-Based Presentation of Flatness-Based Controllers: While [10] uses the $3-2-1$ Euler angle standard to model and control a quadcopter, this paper develops a rotation matrix-based form of the Snap controller without using Euler angles. This is particularly beneficial because there are more than 12 Euler angles conventions not counting all variations [13]. As a result, when implementing a controller relying on Euler angles, it is nearly always the case multiple Euler angle conventions exist for the same rotation. During our research, we found that the existing Snap controller paper [10], the Mellinger controller paper [14], and the Gazebo robot simulation software [15] each have a different convention for Euler angles requiring many conversions. However, all systems have a single convention for rotation matrices which they always provide and take as input. So the presented formulation of the Snap controller based on rotation matrices is universally compatible with all systems without a need for conversions. To accommodate systems that provide quaternions for orientation, we include a quaternion to rotation matrix conversion at the start. While a quaternion-only representation is possible, it would be more computationally expensive than the rotation-matrix representation as rotating a vector p by a quaternion q requires two matrix-like products $p' = qpq^{-1}$ [16]. Note that our proposed formulation still uses a yaw angle, but it is defined using a heading vector as opposed to an Euler angle convention. The definition is equivalent to that of the $3-2-1$ Euler angle standard.

2) Comparison of the Snap and Mellinger controllers: We compare the Snap and Mellinger controllers both from a theoretical point of view and in terms of tracking performance during simulations. We have found that the Mellinger controller is more versatile as it can be easily converted to an attitude controller. Meanwhile, the Snap controller performs better in terms of minimizing tracking error as shown in simulations.

3) Comparison between complex and real poles for tuning the controllers: We discovered that using real or complex poles does not affect the Snap Controller's performance. However, the Mellinger controller performs better when it is tuned using complex poles to a point where they are practically required. We also showed that complex poles did not lead to any non-negligible oscillations in the simulations.

In our simulations, the main cause of loss of stability for quadcopter controllers was not domain of stability but rotor speed bounds. Before the domain of stability is left, there is a smaller domain out of which controllers already lose stability. This is due to the controllers attempting to get thrust and torque values that correspond to non-valid rotor speeds (negative rotor speeds or beyond maximum rotor speeds). This is an important finding because it shows that in order to make quadcopter position control more aggressive, research should focus on creating quadcopter designs which allow for negative thrust. For example, an octo-copter where 4 of its rotors create downward forces, or a quadcopter with symmetric rotors that can spin in both directions.

In Section II, we provide a problem statement and present an overview of the solution strategy for both the Snap and Mellinger controllers. In Section III, we present the dynamics of the quadcopter system using rotation matrices instead of Euler angles and define the concept of quadcopter heading. Section IV builds on the dynamics to formulate the Snap and Mellinger controllers which are then compared from a theoretical point of view in Section V. In Section VI, we present the results of our simulations which are discussed in Section VII with a conclusion in Section VIII.

II. Problem Statement

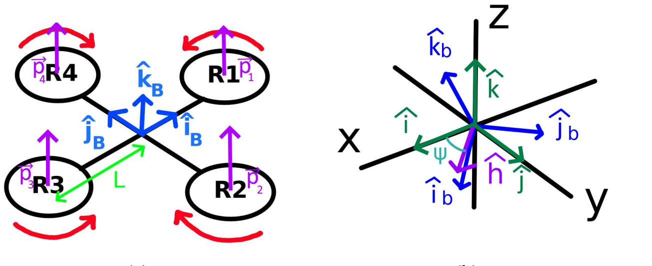
Both the Snap and Mellinger controllers solve the quadcopter yaw-position control. They use the sensor data to stably track both a given smooth trajectory $\boldsymbol{r}_T$ and a given smooth yaw as a function of time ψ_T. More specifically, for both Mellinger and Snap controllers, the quadcopter is equipped with sensors that provide the real-time data

$$\boldsymbol{s} = \{\boldsymbol{r}, \dot{\boldsymbol{r}}, q, \boldsymbol{\omega}\}$$

aggregating position $\boldsymbol{r}$, velocity $\dot{\boldsymbol{r}}$, quaternion q specifying the orientation of the quadcopter body frame with respect to the global frame [16], and angular velocity $\boldsymbol{\omega}$ of the body frame with respect to the global frame. In the Quadcopter Model section, Section III, we show that rotor speeds map to thrust and torque. As such, the controllers only need to provide desired thrust and desired torque to solve the tracking problem. In other words, to ensure position-yaw tracking, Snap and Mellinger need to map $\boldsymbol{s}$, $\boldsymbol{r}_T$, and ψ_T to desired thrust p and desired torque $\boldsymbol{\tau}$. In this paper, we add the requirement that the mapping should not use Euler angles for the reasons described in the Introduction Section.

As shown in the Quadcopter Control section, Section IV, the Snap and Mellinger controllers have different strategies for solving the tracking problem. Mellinger uses a cascaded pair of position and attitude controllers. Meanwhile, Snap uses a parallel pair of position and yaw controllers. Mellinger's controllers are "cascaded" because the output of the Mellinger position controller is given as an input to the Mellinger attitude controller. Snap also has the particularity that it stores previous values of thrust and change in thrust whereas the Mellinger controller is state-less with no stored values.

The Mellinger controller's cascaded form allows it to be easily modified to become an attitude controller making it more versatile as detailed in Section V. However, the results of Section VI show that it is reliant on using complex poles for tuning. The Snap controller on the other hand does not benefit from complex poles and outperforms the Mellinger controller in tracking precision as shown in Section VI.



(a) (b)

Fig. 1. (a) Quadcopter reference pose. Red arrows show the rotation direction of each rotor. Purple arrows show the rotor thrust forces. (b) Heading vector and heading constraints visualized. $(\hat{\mathbf{i}}_B, \hat{\mathbf{j}}_B, \hat{\mathbf{k}}_B)$ are the local frame of reference shown with respect to the inertial frame of reference $(\hat{\mathbf{i}}, \hat{\mathbf{j}}, \hat{\mathbf{k}})$.

III. Quadcopter Model

We denote the inertial reference frame with base vectors $\left(\hat{\mathbf{i}}, \hat{\mathbf{j}}, \hat{\mathbf{k}}\right)$ and the quadcopter's body frame with $\left(\hat{\mathbf{i}}_B, \hat{\mathbf{j}}_B, \hat{\mathbf{k}}_B\right)$. All vectors are represented in the inertial reference frame unless stated otherwise. The body frame base can be determined from the quaternion q through [16] :

$$\hat{\mathbf{i}}_B = 2\left[(q_0q_0 + q_1q_1) - 1 \quad (q_1q_2 + q_0q_3) \quad (q_1q_3 - q_0q_2)\right]^T, \tag{1a}$$

$$\hat{\mathbf{j}}_B = 2\left[(q_1q_2 - q_0q_3) \quad (q_0q_0 + q_2q_2) - 1 \quad (q_2q_3 + q_0q_1)\right]^T, \tag{1b}$$

$$\hat{\mathbf{k}}_B = 2\left[(q_1q_3 + q_0q_2) \quad (q_2q_3 - q_0q_1) \quad (q_0q_0 + q_3q_3) - 1\right]^T, \tag{1c}$$

where q_0, q_1, q_2, q_3 are the components of q. The rotation matrix $\boldsymbol{R}$ describing the orientation of the body frame with respect to the inertial frame and its derivative can be expressed as:

$$\boldsymbol{R} = \left[\hat{\mathbf{i}}_B \quad \hat{\mathbf{j}}_B \quad \hat{\mathbf{k}}_B\right], \tag{2a}$$

$$\dot{\boldsymbol{R}} = \left[\boldsymbol{\omega} \times \hat{\mathbf{i}}_B \quad \boldsymbol{\omega} \times \hat{\mathbf{j}}_B \quad \boldsymbol{\omega} \times \hat{\mathbf{k}}_B\right]. \tag{2b}$$

The quadcopter has a mass m and a diagonal inertia matrix $\boldsymbol{J}$ with entries (J_x, J_y, J_z). It is setup in a plus "+" formation with a distance from rotors to center of mass L as shown in Fig. 1 (a). Note that plus "+" formation means that the rotor arms are aligned with the body frame axes. This is in contrast to a cross "x" formation where the rotor arms are aligned with the diagonals of the body frame. The i-th rotor spins at angular speed $s_i \in [0, s_{max}]$ where s_{max} is the maximum rotor angular speed which is the same for every rotor $i \in \{1, \cdots, 4\}$. The i-th rotor creates a thrust $p_i = k_F s_i^2$ and a torque $\tau_i = \pm k_M s_i^2$ both in the $\hat{\mathbf{k}}_B$ direction where k_F and k_M are aerodynamic constants. Collectively, they create a net thrust force $p\hat{\mathbf{k}}_B$ and net torque $\boldsymbol{\tau}$ given by:

$$\begin{bmatrix} p \\ \boldsymbol{R}^T\boldsymbol{\tau} \end{bmatrix} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & -k_FL & 0 & k_FL \\ -k_FL & 0 & k_FL & 0 \\ -k_M & k_M & -k_M & k_M \end{bmatrix} \begin{bmatrix} s_1^2 \\ s_2^2 \\ s_3^2 \\ s_4^2 \end{bmatrix}. \tag{3}$$

The quadcopter's dynamics re given by:

$$m\ddot{\boldsymbol{r}} = p\hat{\mathbf{k}}_B - mg\hat{\mathbf{k}}, \tag{4a}$$

$$\boldsymbol{J\alpha} + \boldsymbol{\omega} \times (\boldsymbol{J\omega}) = \boldsymbol{\tau}. \tag{4b}$$

where $\boldsymbol{\alpha} = \dot{\boldsymbol{\omega}}$ is the quadcopter's angular acceleration. We express the body frame coordinates of $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$ with:

$$\boldsymbol{\omega} = \omega_i \hat{\mathbf{i}}_B + \omega_j \hat{\mathbf{j}}_B + \omega_k \hat{\mathbf{k}}_B, \tag{5a}$$

$$\boldsymbol{\alpha} = \alpha_i \hat{\mathbf{i}}_B + \alpha_j \hat{\mathbf{j}}_B + \alpha_k \hat{\mathbf{k}}_B. \tag{5b}$$

A. Heading model

We define the heading vector

$$\hat{\mathbf{h}} = \frac{(\hat{\mathbf{i}}_B \cdot \hat{\mathbf{i}})\hat{\mathbf{i}} + (\hat{\mathbf{i}}_B \cdot \hat{\mathbf{j}})\hat{\mathbf{j}}}{||(\hat{\mathbf{i}}_B \cdot \hat{\mathbf{i}})\hat{\mathbf{i}} + (\hat{\mathbf{i}}_B \cdot \hat{\mathbf{j}})\hat{\mathbf{j}}||}. \tag{6}$$

as the normalized projection of $\hat{\mathbf{i}}_B$ on the X-Y plane. As shown in Fig. 1 (b), $\hat{\mathbf{h}}$ is a unit vector containing the $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$ components of $\hat{\mathbf{i}}_B$. They are equal when $\hat{\mathbf{i}}_B$ has no $\hat{\mathbf{k}}$ component meaning that there is no pitch or roll, only yaw. Note that the denominator in Eq. (6) is zero only if the quadcopter is about to flip over, i.e. its front-facing vector $\hat{\mathbf{i}}_B$ is pointing fully upward in the $\hat{\mathbf{k}}$ direction. Heading $\hat{\mathbf{h}}$ is visualized in Fig. 1 and respects the following properties:

$$\left(\hat{\mathbf{k}} \times \hat{\mathbf{h}}\right).\hat{\mathbf{i}}_B = 0, \tag{7a}$$

$$\hat{\mathbf{h}}.\hat{\mathbf{i}}_B > 0. \tag{7b}$$

Yaw is then defined as the principal angle of $\hat{\mathbf{h}}$ such that:

$$\psi = \text{atan2}\left(\hat{\mathbf{h}} \cdot \hat{\mathbf{j}}, \hat{\mathbf{h}} \cdot \hat{\mathbf{i}}\right), \tag{8a}$$

$$\hat{\mathbf{h}} = \cos(\psi)\hat{\mathbf{i}} + \sin(\psi)\hat{\mathbf{j}}, \tag{8b}$$

where atan2 is the function which maps a 2d vector's y and x components to its principal angle [17]. It has the property:

$$\text{atan2}(A\sin(\theta), A\cos(\theta)) = \theta, \quad \forall\theta \in [-\pi, \pi], \forall A \in \mathbb{R}^+. \tag{9}$$

The definition of yaw is equivalent to that of the $3-2-1$ Euler angle standard [13]. This intuitively describes that the direction of the quadcopter's front-facing vector $\hat{\mathbf{i}}_b$ in the X-Y plane irrespective of pitch and roll. As such, if a camera is placed on the front of the quadcopter, yaw-tracking enables controlling the X-Y direction in which the camera is pointing at all times.

IV. Quadcopter Control

Both the Snap controller and the Mellinger controller are designed to allow quadcopters to track a trajectory position $\boldsymbol{r}_T$ and trajectory yaw ψ_T. These approaches are presented in in Sections IV-A and IV-B below.

A. Snap controller

The Snap controller's data flow is visualized in Fig. 2 (a). It can be decomposed into a position controller and a yaw controller which work in parallel.

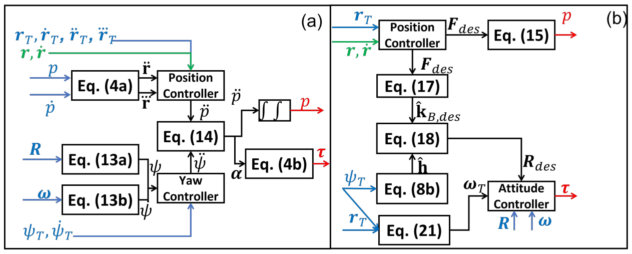


Fig. 2. Data flow of the Snap (a) and Mellinger (b) controllers.

1) Snap position controller: To ensure position tracking, the Snap position controller tries to achieve a snap $\ddddot{\boldsymbol{r}}$ determined by:

$$\begin{aligned}\ddddot{\boldsymbol{r}}_{des} = & -\boldsymbol{K}_1 \left(\dddot{\boldsymbol{r}} - \dddot{\boldsymbol{r}}_T\right) - \boldsymbol{K}_2 \left(\ddot{\boldsymbol{r}} - \ddot{\boldsymbol{r}}_T\right) \\ & -\boldsymbol{K}_3 \left(\dot{\boldsymbol{r}} - \dot{\boldsymbol{r}}_T\right) - \boldsymbol{K}_4 \left(\boldsymbol{r} - \boldsymbol{r}_T\right).\end{aligned} \tag{10}$$

where $(\boldsymbol{K}_1, \boldsymbol{K}_2, \boldsymbol{K}_3, \boldsymbol{K}_4)$ are position gain matrices.

The Snap position controller needs as inputs $\boldsymbol{r}$, $\dot{\boldsymbol{r}}$, $\ddot{\boldsymbol{r}}$, and $\dddot{\boldsymbol{r}}$. $\boldsymbol{r}$ and $\dot{\boldsymbol{r}}$ are given as sensor inputs. $\ddot{\boldsymbol{r}}$ and $\dddot{\boldsymbol{r}}$ can be computed from thrust p and its derivative $\dot{p}$ using the translational dynamics equation Eq. (4a) and its derivative:

$$\ddot{\boldsymbol{r}} = \frac{p\hat{\mathbf{k}}_B - mg\hat{\mathbf{k}}}{m}, \tag{11a}$$

$$\dddot{\boldsymbol{r}} = \frac{\dot{p}\hat{\mathbf{k}}_B - p(\boldsymbol{\omega} \times \hat{\mathbf{k}}_B)}{m}. \tag{11b}$$

As p and $\dot{p}$ are not sensor inputs, Snap stores their values after computing them in each control cycle to be used for the next control cycle.

2) Snap yaw controller: To ensure yaw tracking, the Snap yaw controller tries to achieve a second yaw derivative $\ddot{\psi}$ determined by:

$$\ddot{\psi}_{des} = -K_5\left(\dot{\psi} - \dot{\psi}_T\right) - K_6\left(\psi - \psi_T\right). \tag{12}$$

where (K_5, K_6) are positive yaw gain scalars. The Snap yaw controller needs as inputs ψ and $\dot{\psi}$. These can be obtained using:

$$\psi = \text{atan2}\left(\hat{\mathbf{i}}_B \cdot \hat{\mathbf{j}}, \hat{\mathbf{i}}_B \cdot \hat{\mathbf{i}}\right), \tag{13a}$$

$$\dot{\psi} = \frac{\omega_k\left(\hat{\mathbf{k}} \times \hat{\mathbf{h}}\right).\hat{\mathbf{j}}_B - \omega_j\left(\hat{\mathbf{k}} \times \hat{\mathbf{h}}\right).\hat{\mathbf{k}}_B}{\hat{\mathbf{h}}.\hat{\mathbf{i}}_B}. \tag{13b}$$

3) Mapping to thrust and torque: Once $\ddddot{\boldsymbol{r}}_{des}$ and $\ddot{\psi}_{des}$ are obtained, Snap maps them to $\ddot{p}_{des}$ and $\boldsymbol{\alpha}_{des}$. If we derivate twice and rearrange both the translational dynamics equation Eq. (4a) and the first property of the heading vector (7a) we obtain:

$$\hat{\mathbf{h}}_p = m\ddddot{\boldsymbol{r}} - p\left(\boldsymbol{\omega} \times \left(\boldsymbol{\omega} \times \hat{\mathbf{k}}_B\right)\right) - 2\dot{p}\left(\boldsymbol{\omega} \times \hat{\mathbf{k}}_B\right), \tag{14a}$$

$$\ddot{p} = \hat{\mathbf{h}}_p.\hat{\mathbf{k}}_B, \tag{14b}$$

$$\hat{\mathbf{h}}_\alpha = \frac{\hat{\mathbf{h}}_p - \ddot{p}\hat{\mathbf{k}}_B}{p}, \tag{14c}$$

$$\alpha_i = -\hat{\mathbf{h}}_\alpha.\hat{\mathbf{j}}_B, \tag{14d}$$

$$\alpha_j = \hat{\mathbf{h}}_\alpha.\hat{\mathbf{i}}_B, \tag{14e}$$

$$\hat{\mathbf{k}}' = \left(\hat{\mathbf{k}} \times \hat{\mathbf{h}}\right), \tag{14f}$$

$$V = \left(\ddot{\psi}\hat{\mathbf{h}} + \dot{\psi}^2\hat{\mathbf{k}}'\right).\hat{\mathbf{i}}_B + 2\dot{\psi}\hat{\mathbf{h}}.\left(\boldsymbol{\omega} \times \hat{\mathbf{i}}_B\right)), \tag{14g}$$

$$\alpha_k = \frac{V + \alpha_j\hat{\mathbf{k}}'.\hat{\mathbf{k}}_B - \hat{\mathbf{k}}'.\left(\boldsymbol{\omega} \times \left(\boldsymbol{\omega} \times \hat{\mathbf{i}}_B\right)\right)}{\hat{\mathbf{k}}'.\hat{\mathbf{j}}_B}. \tag{14h}$$

The denominator in Eq. (14h) is zero only if the quadcopter is about to flip over. Also, these equations assume thrust p is non-zero. Then Snap integrates $\ddot{p}_{des}$ twice to get desired thrust p_{des} and gets desired torque $\boldsymbol{\tau}_{des}$ from $\boldsymbol{\alpha}_{des}$ using the rotational dynamics equation Eq. (4b).

B. Mellinger controller

The Mellinger controller's data flow is visualized in Fig. 2 (b) and cascaded position and attitude controllers. To ensure position tracking, the Mellinger position controller tries to achieve a force determined by:

$$\boldsymbol{F}_{des} = -\boldsymbol{K}_p\left(\dot{\boldsymbol{r}} - \dot{\boldsymbol{r}}_T\right) - \boldsymbol{K}_v\left(\boldsymbol{r} - \boldsymbol{r}_T\right) + mg\hat{\mathbf{k}} + m\ddot{\boldsymbol{r}}_T, \tag{15}$$

where $\boldsymbol{K}_p$ and $\boldsymbol{K}_v$ are positive definite gain matrices. The force $\boldsymbol{F}_{des}$ is projected onto the body frame base vector $\hat{\mathbf{k}}_B$ to get the desired thrust force by

$$p_{des} = \boldsymbol{F}_{des}.\hat{\mathbf{k}}_B, \tag{16}$$

Additionally, we normalize $\boldsymbol{F}_{des}$ to obtain the desired orientation of $\hat{\mathbf{k}}_B$ by

$$\hat{\mathbf{k}}_{B,des} = \frac{\boldsymbol{F}_{des}}{\|\boldsymbol{F}_{des}\|}. \tag{17}$$

Assuming the quadcopter is not flipped over (i.e. $\hat{\mathbf{k}}_B.\hat{\mathbf{k}} > 0$), desired unit vectors $\hat{\mathbf{i}}_{B,des}$ and $\hat{\mathbf{j}}_{B,des}$ are obtained by

$$\hat{\mathbf{i}}_{B,des} = \frac{\left(\hat{\mathbf{k}} \times \hat{\mathbf{h}}\right) \times \hat{\mathbf{k}}_{B,des}}{\left\|\left(\hat{\mathbf{k}} \times \hat{\mathbf{h}}\right) \times \hat{\mathbf{k}}_{B,des}\right\|}, \tag{18a}$$

$$\hat{\mathbf{j}}_{B,des} = \hat{\mathbf{k}}_{B,des} \times \hat{\mathbf{i}}_{B,des}. \tag{18b}$$

If the quadcopter were to flip over, the sign of $\hat{\mathbf{i}}_{B,des}$ would flip. We do not account for this case as a flipped quadcopter has already irrecoverably lost control.

Note that the denominator in Eq. (18a) is zero only if the quadcopter is about to flip over. Note that the desired body frame base vectors, denoted by $\hat{\mathbf{i}}_{B,des}$, $\hat{\mathbf{j}}_{B,des}$, and $\hat{\mathbf{k}}_{B,des}$, are all obtained form the rotation matrix $\boldsymbol{R}_{des}$ and used to compute attitude error as defined by:

$$\boldsymbol{e}_R = \frac{1}{2}\left(\boldsymbol{R}_{des}^T\boldsymbol{R} - \boldsymbol{R}^T\boldsymbol{R}_{des}\right)^\vee, \tag{19}$$

where $\square^\vee$ is the vee map which maps skew-symmetric matrices to vectors:

$$\begin{bmatrix} 0 & a & b \\ -a & 0 & c \\ -b & -c & 0 \end{bmatrix}^\vee = \begin{bmatrix} -c \\ b \\ -a \end{bmatrix}, \qquad \forall(a,b,c) \in \mathbb{R}^3. \tag{20}$$

By taking the time-derivative of the translational dynamics equation Eq. (4a) and the first property of the heading vector Eq. (7a), we obtain the components of trajectory angular velocity $\boldsymbol{\omega}_T = \omega_{i,T}\hat{\mathbf{k}}_{B,T} + \omega_{j,T}\hat{\mathbf{j}}_{B,T} + \omega_{k,T}\hat{\mathbf{k}}_{B,T}$ as follows:

$$\mathbf{h}_\omega = \frac{m\dddot{\boldsymbol{r}}_T - \dot{p}_T\hat{\mathbf{k}}_{B,T}}{p_T}, \tag{21a}$$

$$\omega_{i,T} = -\mathbf{h}_\omega \cdot \hat{\mathbf{j}}_{B,T}, \tag{21b}$$

$$\omega_{j,T} = \mathbf{h}_\omega.\hat{\mathbf{i}}_{B,T}, \tag{21c}$$

$$\omega_{k,T} = \frac{\omega_{j,T}\left(\hat{\mathbf{k}} \times \hat{\mathbf{h}}_T\right).\hat{\mathbf{k}}_{B,T} + \dot{\psi}_T\hat{\mathbf{h}}_T.\hat{\mathbf{i}}_{B,T}}{\left(\hat{\mathbf{k}} \times \hat{\mathbf{h}}_T\right).\hat{\mathbf{j}}_{B,T}}. \tag{21d}$$

To ensure both yaw and position tracking, the attitude controller then tries to achieve a torque determined by: We use $\boldsymbol{\omega}_T$ to compute angular velocity error as defined by:

$$\boldsymbol{\tau}_{des} = -\boldsymbol{K}_R\boldsymbol{e}_R - \boldsymbol{K}_\omega\boldsymbol{e}_\omega, \tag{22}$$

where $\boldsymbol{K}_R$ and $\boldsymbol{K}_\omega$ are diagonal gain matrices, and

$$\boldsymbol{e}_\omega = \boldsymbol{\omega} - \boldsymbol{\omega}_T. \tag{23}$$

V. Theoretical Comparison

In this section, we compare the stability and versatility of the Snap and Mellinger controllers in order to assess their performance.

A. Stability

If the desired force and torques can be achieved, the domain of stability for the Snap controller is unbound as proven in [10]. For the Mellinger controller, no proof of stability was provided, but a very similar controller was proven to be stable in [18] given that the initial conditions respect certain constraints.

Assuming reasonable initial conditions, both controllers are able to maintain stability if they can achieve the desired thrust and torques in simulations or experiments as shown in their respective papers. What can cause loss of stability for both of them, as shown later in the simulation section, is that their desired thrust and torques cannot be achieved because they map to rotor speeds which are outside the valid range. For example, desired thrust cannot be negative or higher than what the maximum rotor speeds can provide. In these cases, the usual approach is to "clamp" or restrict the rotor speeds to the valid range by rounding them up to zero if they are negative or rounding them down to the maximum rotor speed.

Based on this, the main measures of stability performance between the two controllers is their ability to track trajectories while staying within the valid motor range and recovering from unexpected thrust/torque values caused by rotor speeds being clamped.

B. Versatility

The Mellinger controller is separated into a position controller that feeds into an attitude controller. This allows it to easily be reconfigured to an attitude controller where the position controller only controls altitude. Attitude control is useful because it allows the drone to have a manual mode where orientation and attitude, or orientation and thrust, are given by the user. On the other hand, the Snap controller cannot be modified to be used as an attitude controller without a near complete rewriting.

Both controllers are assumed to be tuned with pole placement [19]. For stability, the used poles need to be in the left half of the complex plane (negative real part). As shown in the simulation section, the Snap controller performs well with negative real poles, while the Mellinger controller practically requires complex poles for its attitude sub-controller. Complex poles allow the system to converge to the desired state faster. Convergence from value 1 to value 0 with both types of poles is shown in Fig. 3. As shown, the cumulative average for complex poles converges significantly faster than for real poles. A disadvantage of complex poles is that they introduce oscillations which are undesirable for position control. However, the simulation section also shows that the oscillations from complex poles seem to be negligible.

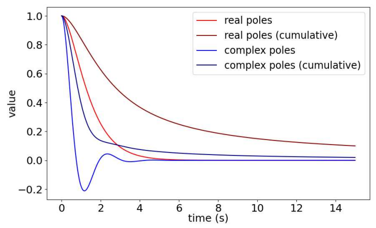


Fig. 3. Convergence from value 1 to value 0 with real and complex poles along with the cumulative averages for an arbitrary one-dimensional linear system with a simple linear controller.

VI. Simulation Results

To compare the Snap and Mellinger controllers. We implemented a discrete version of both in the Python scripting language with a time step of $\Delta t = 10ms$. Then we created a simulation model in the open-source 3D robotics simulator Gazebo [15] based on the following parameters: $m = 0.5kg$, $g = 9.81m/s^2$, $L = 0.25m$, $J_x = J_y = 0.0196kg\ m^2$, $J_z = 0.0264kg\ m^2$, $k_F = 3 \times 10^{-5} N\ s^2/rad^2$, $k_M = 1.1 \times 10^{-6} N\ s^2/rad^2$. Fig. 4 (a) shows the Gazebo quadcopter model used. Finally, we bridged between the controllers and the simulation using ROS, the Robotics Operating System [20], which allows processes to communicate. For the tests, we used helix trajectories for a duration of $T = 10s$ with vertical velocity of $0.1m/s$, radius $1m$, and angular speeds $\omega \in [0rad/s, 2rad/s]$:

$$\mathbf{r}_\omega(t) = \begin{bmatrix}\cos(\omega\sigma_T(t)) & \sin(\omega\sigma_T(t)) & 0.1\sigma_T(t)\end{bmatrix}, \quad (24)$$

where $\sigma_T(t) = T \times \sigma\left(\frac{t}{T}\right)$ for any time $t \in [0, T]$, and

$$\sigma(t) = -20t^7 + 70t^6 - 84t^5 + 35t^4 \qquad \forall t \in [0, 1] \quad (25)$$

ensures that initial and final velocities, accelerations, and jerks are 0 to match the initial takeoff condition.

TABLE I

Used poles for the Snap and Mellinger controllers.

Sub-controller	real poles	complex poles
Snap: position	-10, -10, -10, -10	-
Snap: yaw	-10, -10	-
Mellinger: position	-5, -5	-5, -5
Mellinger: attitude	-1, -1	-0.5 + 3j, -0.5 - 3j

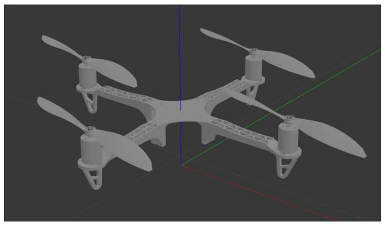
For values of ω separated by $\Delta\omega = 0.1rad/s$, we tested each controller in simulation and recorded the maximum tracking error δ. The poles used for tuning the controllers are shown in Table I. Fig. 4 (b) shows the results of the tests. Figs. (c) and 4 (d) show the real (simulation) and desired trajectories for $\omega = 0.5rad/s$ for the Snap and Mellinger controllers (complex poles for Mellinger).

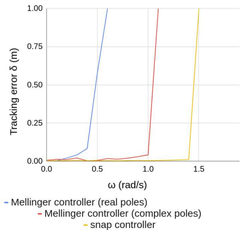
VII. Discussion

As expected, in all cases, the tracking error became larger as ω increased. A phenomena in all cases that these graphs show is that both controllers with either sets of poles have breaking points after which they fully lose stability and spiral out of control. The reason for this seems to be controllers reaching the limits of the rotors' angular velocity range. Fig. 5 shows the rotor speeds for a trajectory near the breaking point for the Mellinger controller with real poles. If the required force and torques are too high in magnitude, they require angular velocities that are too high for certain rotors (reaching the maximum rotor speed), while requiring angular velocities that are too low for the other rotors (rotor speeds cannot be negative). When this is the case, rotor speeds get clamped to the valid range leading to thrusts/torques that are different from the desired ones. This will likely make the tracking error larger which the controller will likely respond to by attempting even more out of bound forces and torques and so on. This cycle seems to lead to the loss of stability at breaking points and beyond in all cases. For the pole types, the Mellinger controller sees a very significant increase in its performance when the poles of its attitude controller are allowed to be complex. Adding complex poles to the Mellinger position controller or to the Snap controller did not have any impact. The reason why the Mellinger controller's performance is dependent on having complex poles for its attitude controller could be because the Mellinger controller's sub-controllers are cascaded. The position controller's output is passed to the attitude controller. While the attitude controller always works, the position controller only works if the drone has the right attitude which the attitude controller is supposed to guarantee. In the case where poles are real, the attitude will exponentially decay toward the desired attitude. In the case where poles are complex, the attitude decays exponentially while oscillating around the desired attitude. The difference is the average attitude during the decay process reaches the desired faster significantly faster when oscillation is added. Using complex poles would therefore allow the position controller to function properly as the average attitude is closer to the desired attitude than with real poles. If this is the reason complex poles affect the Mellinger attitude controller, then the same pattern should appear for all cascaded controllers with a position controller that feeds into an attitude controller. On the other hand, any non-cascaded quadcopter controllers should not be affected by complex poles.

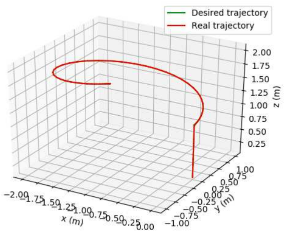
VIII. Conclusion

We compared two quadcopter position-yaw controllers, the Snap controller and the Mellinger controller. We rewrote their equations to use only rotation matrices instead of Euler angles with a vector-based definition of yaw through heading.

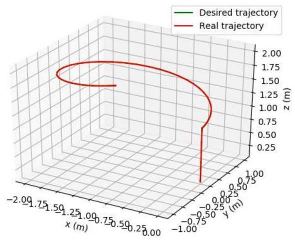

(a)



(b)



(c)



(d)

Fig. 4. (a) Gazebo quadcopter model used for our simulation. (b) Tracking error δ as a function of helix angular velocity ω for the Snap controller and the Mellinger controller with real and complex poles. (c,d) Real (simulation) and desired trajectories for $\omega = 0.5rad/s$ for the Snap controller (c) and the Mellinger controller (d) (Green is desired trajectory which is hidden behind red which is real trajectory).

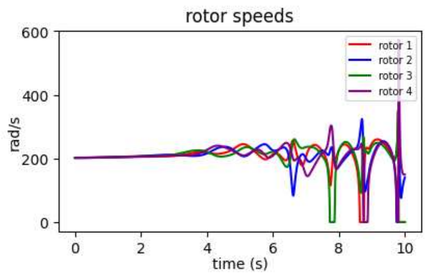


Fig. 5. Rotor speeds for a trajectory near the breaking point for the Mellinger controller with real poles.

Both of them were implemented and tested on single drone simulations. We also tried both real poles and complex poles for tuning the controllers. We found that the Snap controller outperformed the Mellinger controller in terms of minimizing tracking error. We also found that complex poles only affect the Mellinger attitude controller and greatly enhance the Mellinger controller's performance without leading to non-negligible oscillations. Our main finding is that both controllers seem to lose stability because their desired thrust and torques map to rotor speeds outside the valid range.

In terms of possible future work, there are two main problems that are still not solved. The first is the question of whether our reasoning for why complex poles affect the Mellinger attitude controller is right. This can be tested by verifying if the same behavior can be observed with other cascaded controllers. The second problem is creating a multi-copter which can create negative thrust and testing quadcopter controllers with it. This type of design would not have the rotor speed limit issue of regular quadcopter (assuming maximum rotor speeds are big enough) and should be able to perform trajectories that are significantly more aggressive than what a regular quadcopter can do.

References

[1] S. Ahirwar, R. Swarnkar, S. Bhukya, and G. Namwade, "Application of drone in agriculture," *International Journal of Current Microbiology and Applied Sciences*, vol. 8, no. 1, pp. 2500–2505, 2019.

[2] B. Rabta, C. Wankmüller, and G. Reiner, "A drone fleet model for last-mile distribution in disaster relief operations," *International Journal of Disaster Risk Reduction*, vol. 28, pp. 107–112, 2018.

[3] A. Otto, N. Agatz, J. Campbell, B. Golden, and E. Pesch, "Optimization approaches for civil applications of unmanned aerial vehicles (uavs) or aerial drones: A survey," *Networks*, vol. 72, no. 4, pp. 411–458, 2018.

[4] H. Rastgoftar and E. M. Atkins, "Cooperative aerial lift and manipulation (calm)," *Aerospace Science and Technology*, vol. 82, pp. 105–118, 2018.

[5] A. Mir and D. Moore, "Drones, surveillance, and violence: Theory and evidence from a us drone program," *International Studies Quarterly*, vol. 63, no. 4, pp. 846–862, 2019.

[6] A. Alharam, E. Almansoori, W. Elmadeny, and H. Alnoiami, "Real time ai-based pipeline inspection using drone for oil and gas industries in bahrain," in *2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT)*. IEEE, 2020, pp. 1–5.

[7] N. Bao, X. Ran, Z. Wu, Y. Xue, and K. Wang, "Research on attitude controller of quadcopter based on cascade pid control algorithm," in *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. IEEE, 2017, pp. 1493–1497.

[8] E. Tal and S. Karaman, "Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 3, pp. 1203–1218, 2020.

[9] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.

[10] H. Rastgoftar and I. V. Kolmanovsky, "Safe affine transformation-based guidance of a large-scale multiquadcopter system," *IEEE Transactions on Control of Network Systems*, vol. 8, no. 2, pp. 640–653, 2021.

[11] H. Rastgoftar, "Real-time deployment of a large-scale multi-quadcopter system (mqs)," *arXiv preprint arXiv:2201.10509*, 2022.

[12] M. J. Van Nieuwstadt and R. M. Murray, "Real-time trajectory generation for differentially flat systems," *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, vol. 8, no. 11, pp. 995–1020, 1998.

[13] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors," *Matrix*, vol. 58, no. 15-16, pp. 1–35, 2006.

[14] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 2520–2525.

[15] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[16] J. C. Hart, G. K. Francis, and L. H. Kauffman, "Visualizing quaternion rotation," *ACM Trans. Graph.*, vol. 13, no. 3, p. 256–276, jul 1994. [Online]. Available: https://doi.org/10.1145/195784.197480

[17] F. De Dinechin and M. Istoan, "Hardware implementations of fixed-point atan2," in *2015 IEEE 22nd Symposium on Computer Arithmetic*, 2015, pp. 34–41.

[18] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se(3)," in *49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 5420–5425.

[19] M. Kinnaert and V. Blondel, "Discrete-time pole placement with stable controller," *Automatica*, vol. 28, no. 5, pp. 935–943, 1992.

[20] A. Koubâa *et al.*, *Robot Operating System (ROS)*. Springer, 2017, vol. 1.