Drone Acrobatics: Autonomous Flip
Team 3

Khai Nguyen, Shravan Kumar Gulvadi, Yash Jain, Sourojit Saha, Akshay Antony
Carnegie Mellon University
Pittsburgh, USA

***Abstract*—Drones are popular due to their ability to take off and land vertically, high maneuverability, and hovering capability. However, they are highly unstable and susceptible to external disturbances. In this project, we examine flip maneuver which exploits the quadrotor's agility while maintaining stability simultaneously. The flipping pipeline consists of three main modules: trajectory planner, tracking controller and stabilizing controller. A smooth three-stage flipping rate profile is generated in account for the limited sensing and actuating capabilities of the micro quadrotor. A PD controller is used for executing the rapid trajectory for the flip, then switched to a cascaded PID to re-stabilize the quadrotor during normal flight. Beside, a full-state feedback LQR controller is designed to test with the stabilization phase. These modules provide the drone with an ability to perform autonomous flip. Simulation and experimental tests on a Crazyflie quadrotor are successful. However, hardware performance is not consistent and in need of further works.**

***Index Terms*—Quadrotor, LQR, PID, Autonomous Flip, Trajectory Tracking**

I. Introduction

Quadrotors have gained popularity in several decades. Due to its high maneuverability, hovering capability, and ability to take off and land vertically, it poses as the ideal candidate for various tasks such as payload delivery, searches and rescue, and surveillance. The high maneuverability of a quadrotor is due to its 6 DOF, which allows it to move in complex trajectories. However, quadrotors are highly unstable and susceptible to disturbances. One needs a robust controller to fully utilize the quadrotor's ability to track complex trajectories and be stable simultaneously. Many on-going projects are still focusing on manipulate quadrotors to their fullest capabilities, pushing the limit of this interesting subject.

Drone acrobatic movement is one of the best ways to showcase complicated design and impressive manipulation. This is not only for entertainment but also very useful for numerous challenging applications as listed above.

Here, we have implemented autonomous flip maneuver on the micro-quadrotor as it is an interesting and solid way to test its flight capabilities. We use the Crazyflie quadrotor for this objective. Not only is this flyer smaller and lighter than usual quadrotors used in high-speed aerobatic research, but it also has a larger motor time constant, a lower thrust-to-weight ratio, and substantially less computational power [1]. Additionally, we do not use motion capture systems like Vicon or powerful instrumented flyers in outdoor arenas in this research; rather, we simply use the quadrotor's on-board sensor capabilities. The procedure of synthesizing controllers for the scenario presented here is extremely difficult experimentally due to the aforementioned factors.

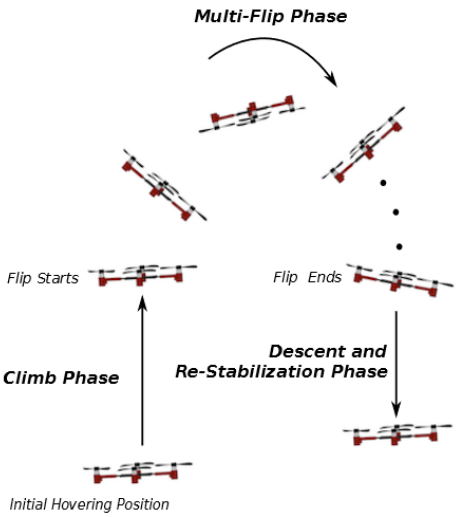


Fig. 1: The whole process of a multi-flip maneuver is composed of: the climb phase, the flip phase, and the descent and re-stabilization phase. [1]

A single flip is typically executed in 0.5 seconds and demonstrates the controllability of the quadrotor's movement to follow a rapid trajectory. This maneuver is accomplished by using a flipping planner that generates the angle rate profile for the maneuver and two specialized controllers: flipping and stabilizing controllers. The flipping planner outputs a trajectory for the quadrotor to follow. This is divided into five stages, the Climb stage, the Increasing angular velocity stage, Max angular velocity stage, Decreasing angular velocity stage, and Stabilizing stage (see Fig. 1). The controllers switch during flight to flip and stabilize the quadrotor. The PD controller is used for executing the flip, and cascaded PID and LQR stabilize the quadrotor in normal flight mode after the flip execution is completed.

The rest of paper is organized as follows. Section II describes relevant works and our base reference, Section III models the dynamics of the flying system for both nonlinear and linearized models, Section IV presents our design including speed planner, tracking controller and stabilizing controller. Section V demonstrates several simulation results. Section VI describes the micro unmanned aerial vehicle and experimental

setup. Experimental results are presented and discussed in Section VII. Lastly, some conclusions are drawn in Section VIII.

II. Literature Review

Aggressive maneuvers for drones have been a topic studied for a long time. The limited computing power and sensor inputs for a quadrotor make the task challenging and interesting. These were achieved mainly using classical controls and recent learning-based methods. Article [2] introduced a way in which any aggressive maneuvers can be divided into phases, and various control strategies can be employed to complete each phase. They design safe regions for any action using Hamilton-Jacobi differential game formulation. This paper focuses on guaranteeing safe regions of operation. They demonstrate the results on a back-flip, where the flip is divided into 3 regions: recovery, drift, and impulse. Although the paper suggests an extremely safe method for acrobatic maneuvers, the complexity of the regions makes it hard to implement on real hardware. Paper [3] introduces a technique for trajectory generation and control for aggressive routines in quadrotors. The core idea of the paper is to compose any maneuver as cascaded controls: Attitude control, Hover control, and 3D path following control. They show the generalizability of the idea by experimenting with 4 diverse aggressive scenarios. We decided to use the idea of generating trajectories as phases and switching between different controllers in our project.

Article [4] suggests a learning-based approach for performing highly aggressive maneuvers on drones. They train a Reinforcement Learning network that takes Camera and IMU data as input and generates the trajectory required for the operation. The reward function gives a positive reward based on an MPC-generated trajectory. They demonstrate the ability of this network using various experiments based on a swarm of quadrotors. Although the paper recommends an end-end to end trajectory generator, the complexity and time requirements for training the RL model made the method inapplicable.

Paper [1] aims to perform single and multiple flips on a Crazyflie using two controllers and a time-dependent trajectory. They employ a cascaded PID controller for normal operations like hovering while it uses a PD-like controller while performing the flip. The trajectory function gives the angular velocities about the flipping axis based on the time inputted. The angular velocities across non-flipping axes are zeros. So the flipping controller takes the three angular velocities and their derivatives as input. The paper provided a simple yet powerful technique for drone flipping, but the parameters for trajectory and controllers in the entire paper were based on Crazyflie 1.0. This work is chosen as our base reference to develop a novel flipping pipeline in Crazyflie 2.1.

III. System Modeling

The first step in this project was to derive a mathematical model of the drone that could accurately represent the dynamics and constraints of the drone. This model is then used for the simulation and designing of the state feedback controller (LQR). The mathematical model of the drone is derived using Newton-Euler equations of motion provided in [5].

A. Quadrotor Dynamics

When considering a complex body like the quadrotor, which has 6 DOF, it is important to consider all the possible frames of reference to understand the complex motion. In the case of a quadrotor, we consider two frames: the drone body frame and the inertial frame (also known as the world frame). Newton's equations of motion are given in the inertial frame, whereas the aerodynamic forces and torques are given in the body frame.

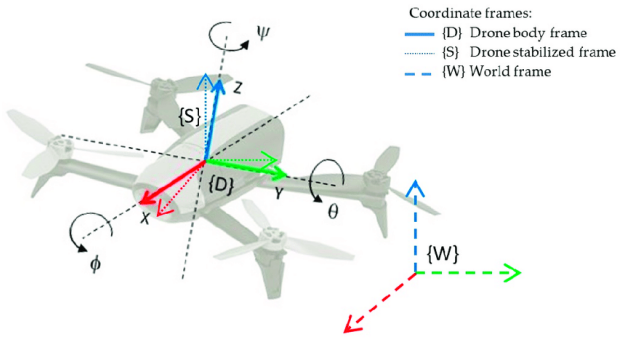


Fig. 2: Frames of reference.

We transform one frame of reference into another through rotations. We use the Euler angle rotations to form a rotational matrix that can convert the inertial frame to a body frame. The matrix is as follows

$$R_b^i = \begin{bmatrix} c\theta c\psi & c\theta c\psi & -s\theta \\ s\phi s\theta c\psi - c\phi s\psi & s\phi s\theta s\psi + c\phi c\psi & s\phi c\theta \\ c\phi s\theta c\psi + s\phi s\psi & c\phi s\theta c\psi - s\phi c\psi & c\phi c\theta \end{bmatrix} \quad (1)$$

where $c(\cdot) = \cos(\cdot)$ and $s(\cdot) = \sin(\cdot)$. The rotational matrix for converting the body frame to an inertial frame can be obtained by taking the inverse of R_b^i, which in this case is equal to the transpose of R_b^i, $R_i^b = (R_b^i)^\intercal$. Also, the rotational matrix for converting the rate of change of Euler angles from the body frame to the inertial frame is given as

$$R_r^i = \begin{bmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi)sec(\theta) & \cos(\phi)sec(\theta) \end{bmatrix} \quad (2)$$

The 12 states of the drone are defined as follows.
x, y, z = position in x-axis, y-axis and z-axis (height)
u, v, w = body frame velocity in x-axis, y-axis and z-axis
ϕ, θ, ψ = roll angle, pitch angle and yaw angle
p, q, r = roll rate, pitch rate and yaw rate
Using the kinematics and dynamics equations of the quadrotor, the nonlinear model obtained is as follows-

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = R_i^b \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad (3)$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{bmatrix} + \begin{bmatrix} g\sin(\theta) \\ -g\cos(\theta)\sin(\phi) \\ -g\cos(\theta)\cos(\phi) \end{bmatrix} + \frac{1}{m}\begin{bmatrix} 0 \\ 0 \\ F + mg \end{bmatrix} \quad (4)$$

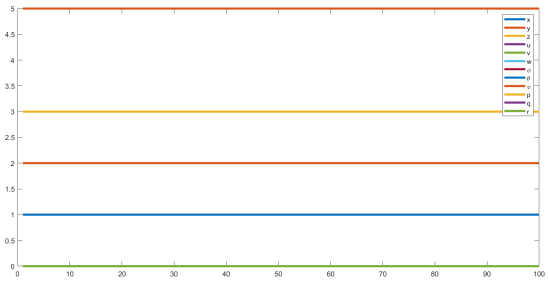


Fig. 3: The system is stable

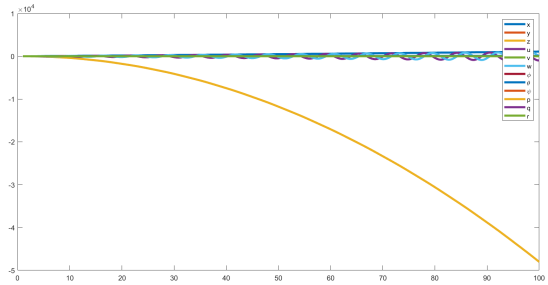


Fig. 4: The system is unstable

$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta)\\ 0 & \cos(\phi) & -\sin(\phi)\\ 0 & \sin(\phi)sec(\theta) & \cos(\phi)sec(\theta)\end{bmatrix}\begin{bmatrix}p\\ q\\ r\end{bmatrix} \tag{5}$$

$$\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix}\frac{I_y-I_z}{I_x}qr\\ \frac{I_z-I_x}{I_y}qr\\ \frac{I_x-I_y}{I_z}qr\end{bmatrix} + \begin{bmatrix}\frac{T_\phi}{I_x}\\ \frac{T_\theta}{I_y}\\ \frac{T_\psi}{I_z}\end{bmatrix} \tag{6}$$

The mg term is added to account for the gravitational force. Here, I_x, I_y, I_z, T_ϕ, T_θ and T_ψ are the moments of inertia and torques in the respective directions.

B. Linearized Model

For designing the LQR controller, we need to derive the linearized model of the drone. For this, the equilibrium points of the system are identified and the nonlinear system model is linearized around one of the equilibrium points. To find the equilibrium points, the LHS of the nonlinear model is set to zero and then we solve for the values of the states. From the obtained results, it can be inferred that all the equilibrium points of the drone are in the following form-

$$X_{eq} = [x, y, z, 0, 0, 0, 0, 0, \psi, 0, 0, 0]^\intercal$$

Hence, infinite equilibrium points exist with various combinations of x, y, z, and ψ and all other states as zeros. This inference is tested through a MATLAB simulation where the drone was simulated at an equilibrium point of the form mentioned above and also at a point where some states other than x, y, z, and ψ had some non-zero value. The simulation results are provided in Fig. 3 and Fig. 4.

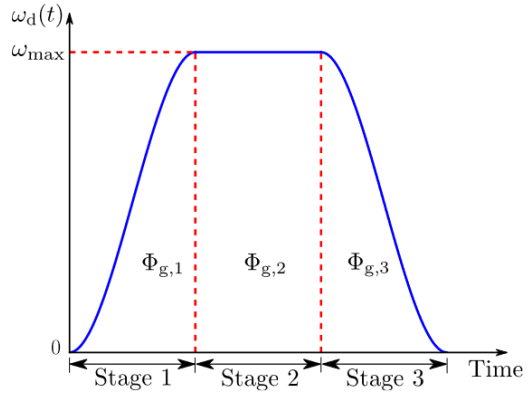


Fig. 5: Generalized flipping speed reference. [1]

We observe that the system is unstable at the point where states other than x, y, z, and ψ have a non-zero value. Hence, the intuition about the equilibrium point is right. The point with all states as zero is chosen as the equilibrium point around which the non-linear model is linearized. The linearization is done by computing the Jacobian of the nonlinear model and then substituting the state values as zero. The obtained A and B matrices are shown below.

$$A = \begin{bmatrix}
0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&g&0&0&0&0\\
0&0&0&0&0&0&-g&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}, B = \begin{bmatrix}
0&0&0&0\\
0&0&0&0\\
0&0&0&0\\
0&0&0&0\\
0&0&0&0\\
\frac{1}{m}&0&0&0\\
0&0&0&0\\
0&0&0&0\\
0&0&0&0\\
0&\frac{1}{I_x}&0&0\\
0&0&\frac{1}{I_y}&0\\
0&0&0&\frac{1}{I_z}
\end{bmatrix}$$

The values of mass, inertia in all directions and motor constants can be found in [6].

IV. Controller Design

This section is split into 3 main parts- flipping trajectory planner, stabilizing controller design, and flipping controller design. The overall control architecture is designed to switch between the two controllers based on the time and reference trajectory. We plan to use the LQR controller during the normal flight and stabilization phase and the flipping controller during the flip stages of the trajectory.

A. Flipping Trajectory Planner

First, reference trajectory is defined to represent a flip in mathematical form. This is the reference trajectory the controllers need to track to perform the flip. The design of the trajectory needs to factor in the dynamics of the drone and the limitations of the hardware as well. The flip trajectory is defined in the form of angular speeds instead of angles. This has two benefits; namely, the trajectory can be designed to better account for the actuating limits of the drone, and also lower chances of stall [1].

We split the complex maneuver of flip into five stages to for easier representation in mathematical forms and switch

between them based on time. The primary three middle planning stages can be seen in Fig. 5.

The flipping planner outputs a trajectory for the quadrotor to follow. This is divided into five stages as follows:

- Ascent phase: Here the quadrotor is boosted to obtain moments and inertia before executing the flip.
- Increasing angular velocity phase: In this stage, the roll rate slowly increases from 0 to ω_{max}.
- Max angular velocity phase: At this stage the ω_{max} is commanded.
- Decreasing angular velocity phase: In this stage, the roll rate decreases from ω_{max} to 0.
- Stabilizing phase: This phase is activated after a complete flip. The quadrotor is stabilized and enters normal flight mode.

Before performing a flip maneuver, we define a hovering height. To start, the ascent phase ensures that the quadrotor keeps climbing higher until the next stage reference is provided. This is an essential part of the trajectory as we want the drone to maintain a certain height while flipping, and this ascent helps us achieve this goal. We provide this climbing trajectory for a fixed amount of time and then switch to the increasing angular velocity phase.

In this stage, an angular velocity is provided in the axis we wish to perform the flip motion about. We gradually increase this angular velocity to obtain a smooth and feasible motion. The angular velocity for this phase is calculated by the equation below. [1]

$$\omega_d = \frac{\beta_1}{3}(\gamma_1)^3 - \beta_1\gamma_1^2 t + \frac{\beta_1\gamma_1^3}{3} \tag{7}$$

where,

- $\gamma_1 = \omega_{max}^{-1}\Phi_g$
- $\beta_1 \frac{-3}{4}\gamma_1^{-3}\omega_{max}$
- $\dot{\omega}_{max,1} = \frac{3}{4}\cdot\Phi_{max}^{-1}\omega_{max}^2$
- $\Phi_{g,1}$ is the desired amount of rotation generated during the increasing angular velocity stage
- $\dot{\omega}_{max,1}$ is the maximum flipping acceleration in the first section of ω_d
- The time duration of the first section ω_d, $\omega_{d,1}$ is $\delta_1 = 2\gamma_1$

Now comes the max angular velocity stage. During this stage, the desired trajectory is provided as the maximum angular velocity as defined in the following equation.

$$\omega_{d,2}(t) = \omega_{max} \tag{8}$$

with,

- The associated time duration is $\delta_2 = \omega_{max}^{-1}\Phi_{g,2}$
- $\Phi_{g,2}$ is the desired amount of rotation generated during max angular velocity stage.

The next stage is the decreasing angular velocity stage. During this stage, we gradually decrease the angular velocity to end the flip and set the drone up for the stabilizing stage. The following equation describes the angular velocity in this phase.

$$\omega_{d,3}(t) = \frac{\beta_3}{3}(\gamma_3 + \delta_2^{'} - t)^3 - \beta_3\gamma_3^2(2\gamma_3 + \delta_2^{'} - t) + \frac{\beta_3\gamma_3^3}{3} \tag{9}$$

where,

- $\delta_2^{'}$ is the ending time of previous stage
- $\gamma_3 = \omega_{max}^{-1}\Phi_{g,3}$
- $\beta_3 = \frac{-3}{4}\gamma_3^{-3}\omega_{max}$
- $\dot{\omega}_{max,3} = \frac{3}{4}\Phi_{g,3}^{-1}\omega_{max}^2$
- $\Phi_{g,3}$ is the desired amount of rotation generated during decreasing angular velocity stage
- The time duration for this stage is $\delta_3 = 2\gamma_3$

Once we enter the stabilizing stage, we set the angular velocity reference as zero and provide a hovering height as the reference trajectory to the drone. Since this is an equilibrium point, the drone should be able to stabilize around this hovering point.

B. Flipping Controller

Following [1], a PD-like controller is designed for tracking the angular velocity from the planner. For simplicity, we propose a pure PD controller as follows

$$K(s) = k_p + k_d s \tag{10}$$

where k_p is the proportionality constant of the error term, and k_d is the proportionality constant of the derivative of the error term. The control effort is calculated using the following relation

$$u(t) = k_p e(t) + k_d \frac{d}{dt}e(t) \tag{11}$$

The PD controller can be tuned to obtain a desired behavior from the controller by varying the k_p and k_d terms. This controller provides the control efforts in the form of three torques, and a thrust is set as the drone's weight to maintain a constant height and not to saturate the actuators.

C. Stabilizing Controller

An LQR controller is designed to re-stabilize the drone after a complete flip. It is noteworthy that we require a robust controller because initial errors for this controller can be significant. LQR controller is a full-state feedback controller and is designed using the linearized mathematical model of the quadrotor. An optimization problem is solved to obtain the gains of the LQR controller. The cost function for the optimization problem is as follows

$$J = \int_0^\infty (x^\intercal Q x + u^\intercal R u)dt \tag{12}$$

Here, x represents the states of the quadrotor, u represents the control signals, Q is a diagonal matrix that represents the weights of the states, and R is a diagonal matrix that represents the weight on each control effort.

The gain matrix K is calculated from the following relation -

$$K = R^{-1}B^\intercal P \tag{13}$$

where P is the solution of the Riccati equation written below.

$$A^\intercal P + PA - PBR^{-1}B^\intercal P + Q = 0 \tag{14}$$

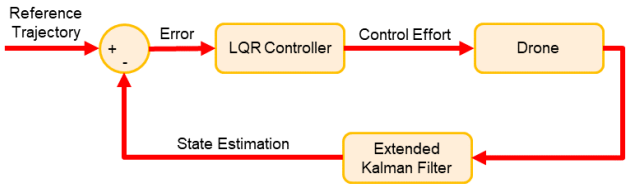


Fig. 6: LQR controller architecture.

Once the gain matrix is found, the control effort is calculated using the following relation

$$u(t) = -Ke(t) \tag{15}$$

where $e(t)$ is the error between the reference state and the actual state (obtained from the Extended Kalman filter) at each time step.

The controller architecture can be represented from the Fig. 6.

The LQR controller is tuned by varying the Q and R matrices to get the desired performance from the controller.

V. Simulation

To speed up the process of controller designing and tuning, we have developed simulation models for both controllers. The first simulation model developed is for the LQR controller. This model is developed in MATLAB using the linearized dynamics model around the hovering point. The Q and R matrices were tweaked to tune the LQR controller. The performance of this controller is simulated using the nonlinear dynamics model of the drone with different starting points and trying to stabilize around the hovering point. The simulation is done using the ode45 solver at each time step to calculate the states of the drone. The following Q and R matrices calculate the gains for the LQR controller.

$$Q = \text{diag}([2000, 2000, 2000, 1000, 1000, 1, 1700, 1700, 1700, 800, 800, 0.5]) \tag{16}$$

$$R = \text{diag}([1e^4, 6e^9, 6e^9, 1e^4]) \tag{17}$$

The performance upon simulation is represented in Fig. 7. We can see that most of the states do settle pretty quickly and some states, mainly x, do not settle fast enough. This means that there would be some drift in the x direction if this controller is implemented on the drone.

We see some deviation in performance when using the controller designed using this simulation on the hardware. This could be because the system parameters used in these simulations might not be close to the real parameters.

During our literature review, we came across [7]. During the initial experimentation, we observed that the model used here is closer in performance to the actual hardware. This might be because the system parameters used in this model might be closer in value to the actual hardware system parameters. Thus, we decide to switch to this Python-based simulation environment. One thing to note is that the order of states in

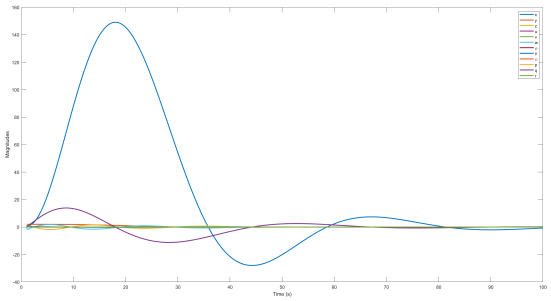


Fig. 7: LQR controller performance in Matlab simulation.

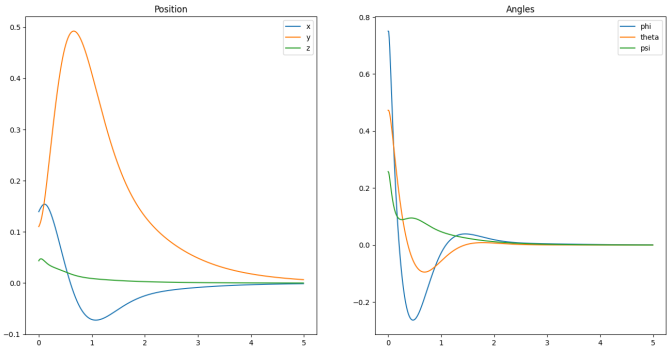


Fig. 8: LQR performance simulation in Python

this model is different from the model we have derived. We obtain the following gain matrix for the same Q and R matrices as above.

$$K^\intercal = \begin{bmatrix} 0.0 & 0.0 & 8e^4 & 0.0 \\ 0.0 & -8e^{-4} & 0.0 & 0.0 \\ -0.6 & 0.0 & 0.0 & 0.0 \\ 0.0 & -3.6e^{-3} & 0.0 & 0.0 \\ 0.0 & 0.0 & -3.6e^{-3} & 0.0 \\ 0.0 & 0.0 & 0.0 & -1.4e^{-2} \\ 0.0 & 0.0 & 1.1e^{-3} & 0.0 \\ 0.0 & -1.1e^{-3} & 0.0 & 0.0 \\ -0.5 & 0.0 & 0.0 & 1e^{-2} \\ 0.0 & -9e^{-4} & 0.0 & 0.0 \\ 0.0 & 0.0 & -9e^{-4} & 0.0 \\ 0.0 & 0.0 & 0.0 & -1e^{-2} \end{bmatrix} \tag{18}$$

The performance of this controller is represented by the Fig. 8. We see that the states converge better in this simulation. This is similar to the performance we see on the hardware.

Simulink is used to simulate the flipping controller. We have tuned the gains to achieve reasonable trajectory tracking performance in the flipping stage. Nonlinear dynamics, the trajectory generator, the stabilizing LQR controller, and the flipping controller are integrated into this Simulink module to test the system's performance for executing single, double, and triple flips. The Simulink block is represented in Fig. 9, and the results are represented in Fig. 10, 11, and 12. These results show that the proposed controllers work very well and can perform multiple flips successfully.

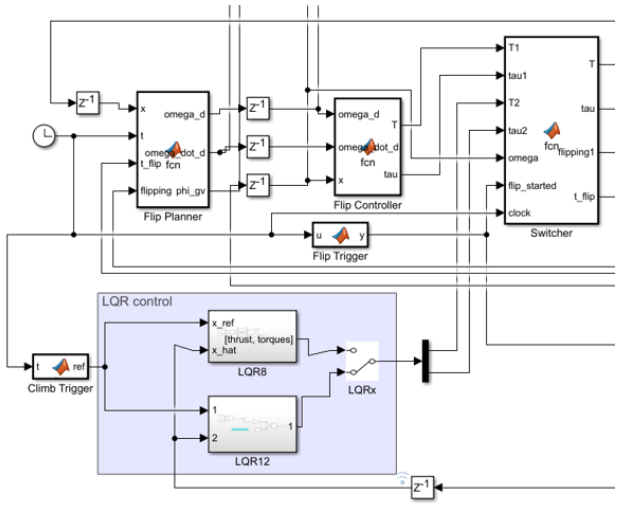


Fig. 9: Simulink diagram of the entire system.

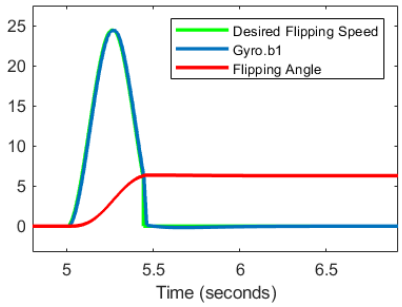


Fig. 10: Single flip simulation result.

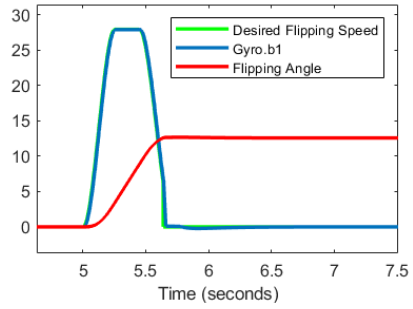


Fig. 11: Double flip simulation result.

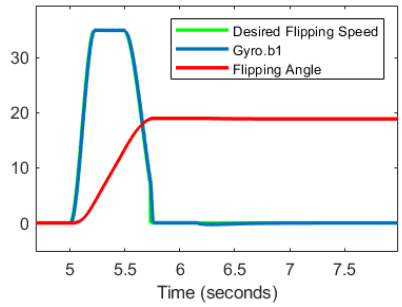


Fig. 12: Triple flip simulation result.

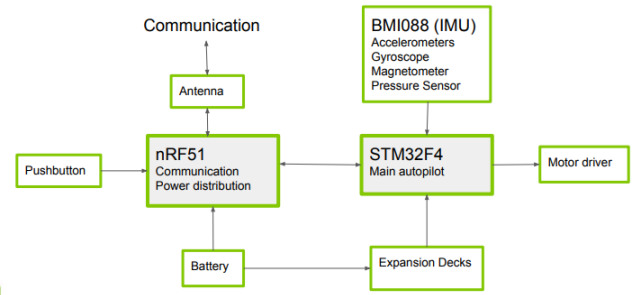


Fig. 13: Crazyflie hardware architecture (from Bitcraze)

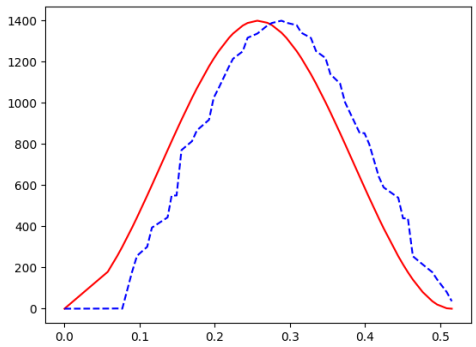


Fig. 14: Logging of sent and received trajectory. Red and blue lines represent the sent and received trajectory signals respectively.

VI. Hardware Implementation & Testing

Our hardware includes the stock Crazyflie drone which is a quadrotor drone in X-configuration with a size of 92x92x29mm and a weight of 27 grams. The drone has four DC coreless motors and can generate a max thrust of 60 gms. The drone has two microprocessors; the STM32F4 microprocessor supports the drone's Autopilot features, and the nRF51 supports the drone's communication. The drone has an inbuilt BMI088 IMU that continuously streams the sensor measurements to the Autopilot microprocessor. The drone was augmented with the flow deck from Bitcraze's expansion deck, which has a PMW3901 optical flow sensor and a VL53L1x Range sensor and provides the necessary information for the relative positioning of the drone for navigation. The overall hardware architecture of the drone is as shown in Fig. 13.

The flipping planner is implemented within Crazyflie's Python-lib environment and communicated with Crazyflie via the Crazyradio at 75-155Hz. We have two variants of the flipping planner; the open-loop planner uses the time to determine trajectory stages, while the closed-loop planner takes in the drone's accumulated roll angle as feedback, calculates the next waypoint, and transmits it continuously to the drone while flipping. We tested the open-loop planner first because the duration of each planning stage can be precomputed. However, the risk is that only small timing errors could make the flip maneuver unsuccessful. Fig. 14 shows the trajectory from the base station and from the quadrotor. Although the signal

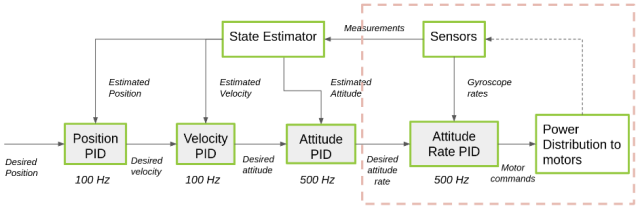


Fig. 15: Cascaded PID architecture (credits Bitcraze).

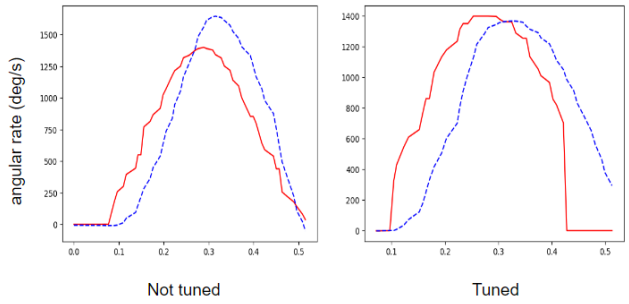


Fig. 16: The performance of the flipping controllers. The blue dotted line is tracking the red one. When the attitude rate PID is tuned, there is no overshoot anymore.

is well received, there is an amount of delay due to radio communication. Therefore, we have to forward the trajectory to compensate for that delay in software.

For the flipping controller to track the trajectory, we leverage the 500Hz attitude rate portion of the inbuilt cascaded PID (see Fig. 15) as its gains could be easily tuned to behave as the controller in [1], which basically have the same property as a PD controller (see Fig. 16). We need to change the PID from angle mode to rate mode to enable this function. Now, the input for that block is the flipping rate command sent from our planner at the base station. We also switch off the supervisory check for tumble detection during the flip period to ensure the motors keep working throughout the execution of the flip. This is done by modifying the firmware to receive commands through the Python API to enable and disable the supervisory check as needed. Similar modifications are also made to enable sending controller gains to the Crazyflie through Python API.

As for the LQR controller, it takes in all the 12 drone states from the inbuilt Extended Kalman Filter as input and produces the motor commands as output. The resulting thrust and body torques are mapped to motor commands as follows:

$$\begin{bmatrix} f \\ \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \begin{bmatrix} k_f & k_f & k_f & k_f \\ -k_f d & -k_f d & k_f d & k_f d \\ k_f d & -k_f d & -k_f d & k_f d \\ -k_m & k_m & -k_m & k_m \end{bmatrix} \begin{bmatrix} \tilde{\omega}_i^1 \\ \tilde{\omega}_i^2 \\ \tilde{\omega}_i^3 \\ \tilde{\omega}_i^4 \end{bmatrix} \tag{19}$$

Based on [7], we obtain the thrust coefficient $k_f = 1.8221e^{-6}$ N per PWM unit, and drag coefficient $k_m = 4.4733e^{-8}$ Nm per PWM unit, and distance from each motor to CoM $d = 0.046$ m with $\tilde{\omega}_i^2$ is the percentage of maximum angular velocity represented as a 16-bit integer and act as the motor commands. This LQR controller is implemented directly in firmware at a sampling rate of 500Hz.

We conduct both our flipping and stabilizing tests in indoor environments with abundant light to avoid wind disturbances and to ensure state estimates from our flow deck are as accurate as possible we ensured that the drone flew above a feature-rich surface. Hover tests under controlled disturbances qualitatively assess the performance of stabilizing controller. We log the drone states continuously during flipping and use it to assess accuracy in tracking the flip trajectory of our flipping controller.

A simplified flipping procedure is depicted as follows

1) Initialize Kalman filter and stabilizing controller
2) Hover at h (m)
3) Ascend in t_{asc} seconds
4) Turn off tumble check, switch to the attitude rate controller
5) Send and track reference until reaching time/angle
6) Turn on tumble check, switch to stabilizing controller
7) Stabilize then land

VII. Demo Results

This section describes our demo results for hovering the LQR controller and flipping maneuver. While several plots are shown and analyzed, the links to videos are provided.

A. LQR Controller Results

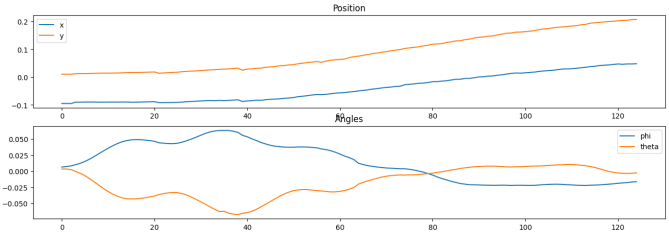


Fig. 17: Hovering LQR controller results. x-axis presents sample counts, the y-axis represents meters and radians. The x-y position is drifting slightly due to lower weights and state estimation. Roll and pitch converge to equilibrium values quickly under no disturbances.

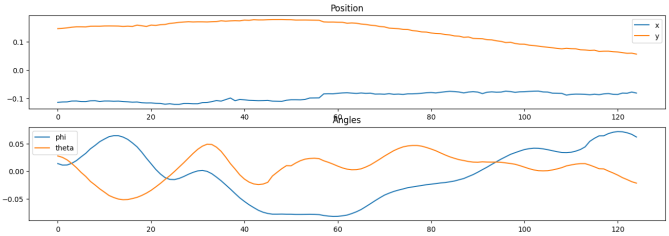


Fig. 18: Hovering LQR controller results under disturbances. x-axis presents sample counts, the y-axis represents meters and radians. Under disturbances, the controller still keeps the drone's position and orientation near the equilibrium point. Check our demo video.

Firstly, the LQR controller is tested for hovering at a constant height of 0.5 m. Later, some disturbances are created by waving a notebook toward the drone to verify its robustness. The position x-y and orientation roll-pitch of the drone while hovering are logged and shown as in Fig. 17 and 18. As can be seen from Fig. 17, our LQR controller can stabilize both position and attitude of the drone. However, the position drifted slightly due to its lower weight compared to attitude and sensitivity to state errors. In Fig. 18, with the same hovering setup, some disturbances that mimic natural winds are introduced. Due to this effect, there are some deviations in the states, but the drone's stability is still maintained by the LQR controller. This shows that our LQR design is robust and ready to be integrated into the flipping pipeline as a stabilizing controller.

B. Flipping Maneuver Results

There are several testing cases for flipping maneuvers depending on open-loop vs. closed-loop planner, not tuned vs. tuned attitude rate PID, and PID vs. LQR stabilizer. Single flip is tested first with detailed analysis before moving to the more challenging cases. The simplified procedure in VI is performed for each case, with the main program being executed using Python API.

Fig. 19, 20 and 21 show the demo results for a successful single flip maneuver in which open-loop planner and PID stabilizer are used, and attitude rate PID is not tuned. In Fig. 19 (see the video for a better experience), eight samples of the flipping procedure are shown subsequently, strictly following the simplified procedure in VI.

Fig. 20 looks at the details when the flipping controller is active. There are definitely significant delays from base station to drone and from command to response. Because the attitude rate PID is the default one, it has some overshoot and reaches approximately 500 deg/s instead of zero at the end of the stage. Because the roll angle is directly logged and plotted, it has a jump from 180 to -180 degree and then go to near zero, which intuitively represents a continuous increase from zero to 180 and to almost 360 degrees at the end. This figure gives us the intuition that for successful stabilization after flipping, the terminal angular rate should be less than 500 deg/s and the angle should be less than 20 degrees. it is noteworthy that this only applies to the default cascaded PID which has quite robust performance.

In Fig. 21, the entire flipping process is logged and presented for 3.5 seconds in which the flipping controller is active for about 0.5 seconds. It takes approximately 0.6 seconds to stabilize after the flip using the default PID controller, much faster compared to 1.6 seconds from [1] but definitely not as good as our simulation results. Unfortunately, the majority of the open-loop tests witness failures in which the quadrotor cannot maintain stability after flipping.

Many modifications and improvements are made during the testing process including tuning the attitude rate PID and cascaded PID, adjusting the hovering height and thrust setpoint, modifying the switching time and trajectory phases, etc.

Instead of the stock PID, the LQR controller is integrated into the flipping pipeline for the re-stabilization phase. It is worth noting that the open-loop planner is still used till this experiment. However, the LQR controller cannot re-stabilize the quadrotor after the flip. We also tested pitch-axis and multiple flipping maneuvers but they are not successful.

Last but not least, the closed-loop planner is implemented and tweaked from Python API. By using a feedback signal from accumulated roll angle, the switching time and trajectory stages are now decided in a better fashion. The conditions for re-stabilization are all good and promising but more time is needed to provide further results.

These failures will be analyzed in the next section.

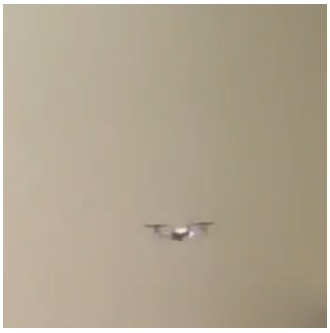
C. Failure Analysis

Fig. 22 shows that the open-loop planner often fails due to incorrect timings. Switching to stabilizer is planned at 0.5 seconds but the actual roll angle and roll rate have not been close to zeros. This makes it challenging for the stabilizing controller to recover at that large initial error. With the tuned flipping controller, the angle response is slower and the switch happens at just 260 degrees. These results highlight the importance of a closed-loop planner, which uses roll angle feedback to decide trajectory stages and switching time.

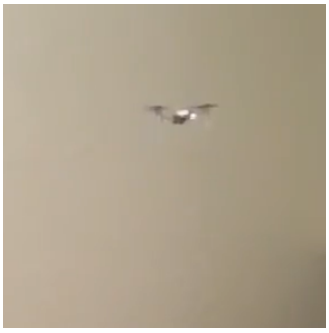
As for the LQR controller, Fig. 21 shows that even when the conditions at the switching instance (0.5 seconds) are good, it still cannot re-stabilize the quadrotor after flipping. It is obvious that our LQR design is not good enough for this application. Our LQR controller, which takes full 12-state feedback and outputs the control signals directly at 500 Hz, is sensitive to measurement noises. Another working approach is to run LQR at only 50 Hz and use a motion capture system for accurate state measurements. Moreover, LQRs for lower-level control with 8 or 6 states are derived as well and proved to have robust performance.

Pitch-axis flipping behavior turns out different from the roll-axis one due to the asymmetric configuration of the quadrotor. The angle response is faster, which means the quadrotor rotates more than 360 degrees during 0.5 seconds. Therefore, more modifications in the planner are required to make it work. As the single flip maneuver is not consistent, multiple flips are simply not possible to this end. But it is certain that the quadrotor can still track the reference to some extent.

As the close-loop planner is introduced, the flipping phase is well executed. The roll angle and rate are consistently close to zeros at the switching time. However, the results are still not as expected but very promising. Notice that the fragility and inconsistency of this 19-gram quadrotor is also a huge challenge for our application. A simulation in Simulink reproduces the effect of time delay communication between the off-board planner, and the on-board microprocessor is made. As a time delay of 0.05 seconds is injected, the re-stabilization phase is negatively affected. It would be even

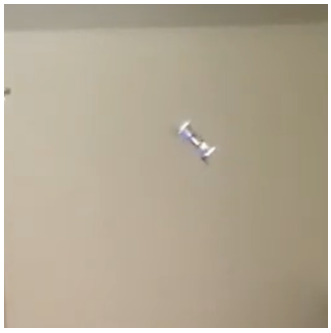


(a) Hovering at h m

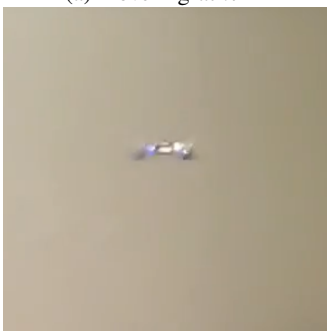


(b) Ascending and trigger flip

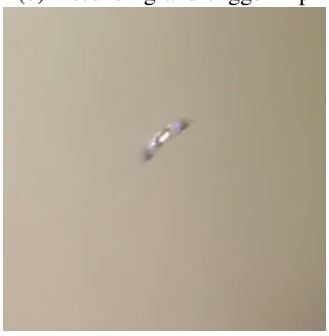
(c) At 45 degrees



(d) At 135 degrees



(e) At 180 degrees



(f) At 225 degrees

(g) At 340 degrees, switch

(h) Stabilize and land

Fig. 19: Eight samples of our flipping maneuver from this demo video.

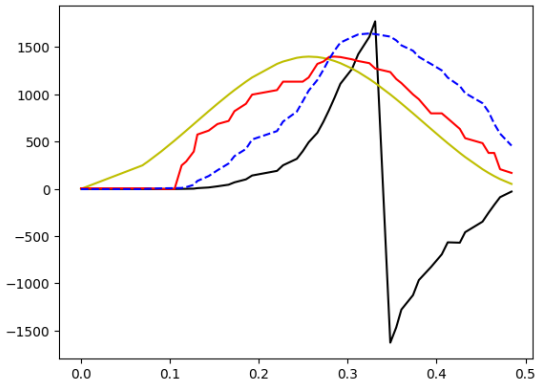


Fig. 20: Experimental results while the flipping controller is active. The x-axis presents time (approximately 0.5 seconds for a single flip), and the y-axis represents degrees and degrees/second. Yellow and red lines are calculated/sent and received reference trajectory. The Blue dotted line is the measured angular velocity trajectory which is tracking the red one. The black line represents 10 times roll angle with a jump being accumulated angle from 180 to 360 degrees.

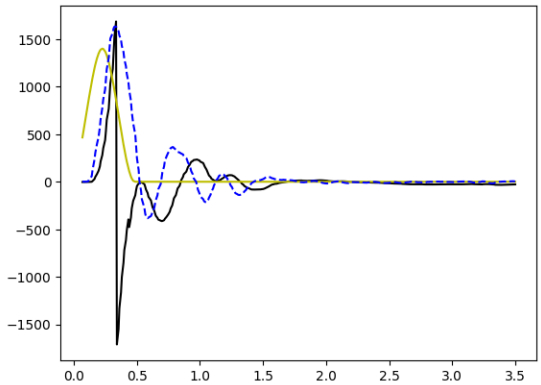
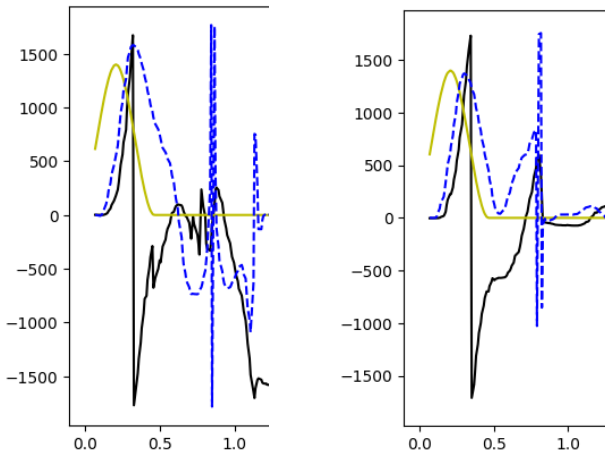


Fig. 21: Experimental results of the entire flipping maneuver. The x-axis presents time, the y-axis represents degrees and degrees/second. Yellow is calculated/sent reference trajectory. The Blue dotted line is the measured angular velocity trajectory. The black line represents 10 times roll angle which is completely stable after 1.6 seconds.

worse with real hardware, which explains why all modules should be embedded in the on-board microprocessor.

VIII. Conclusions

This work has provided a complete flipping maneuver pipeline which is very challenging for a micro quadrotor's acrobatic application. Our approach divides the process into three steps: ascent, flipping and re-stabilization. Following that, we developed a flipping planner, a flipping controller, and a stabilizing controller. An open-loop flipping planner will generate a three-stage attitude rate reference based on time, compared to the generalized flipping angle feedback of a closed-loop one. This planner must consider the limited sensing and actuating capacity of the quadrotor. To track that reference, we use a flipping controller, which has a PD-



(a) Normal test (b) Tuned attitude rate PID

Fig. 22: Experimental failure due to incorrect timings of open-loop planner. Labels are as in Fig 21. With the planned switch at 0.5 seconds, the actual angle and rate are far from zeros, making it impossible for the default PID stabilizer. Even when the attitude rate PID is tuned, the angle response is slower.

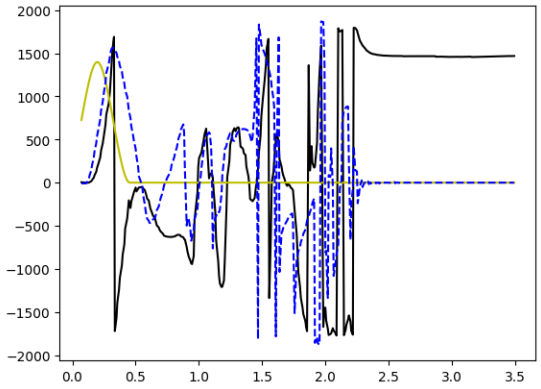


Fig. 23: Experimental failure due to LQR stabilizer. Labels are as in Fig 21. Although the condition at the switch (0.5 seconds) is good, the LQR stabilizer is not successful.

like architecture, then switch to a regular controller after completing a flip. For that re-stabilization phase, both cascaded PID and LQR controllers are considered. These modules are implemented via Python API as well as directly Crazyflie firmware. Finally, this paper has detailed experimental testing and demonstrated successful flip maneuvers. However, the majority of the tests witness failures due to many different factors, including communication delay, open-loop drawbacks, and not-robust-enough LQR controllers.

Many modifications to the current pipeline can be made to improve the performance in the future. Firstly, the closed-loop planner, which needs more integration tests, has provided the stabilizing controller with good initial conditions. Secondly, all modules should be directly implemented in the on-board microprocessor, which will significantly help with the delay

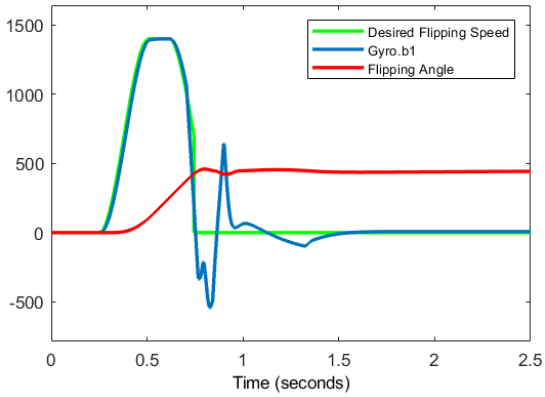


Fig. 24: Delay effect simulation. A delay of 0.05 seconds is added to the communication between the planner and controller. This significantly worsens the re-stabilization phase.

problem. Last but not least, more accurate state measurements and different architectures can boost the LQR stabilizer's robustness and overall performance.

ACKNOWLEDGMENT

We want to thank Prof. Bedillion for his continued support and guidance for our project. We would also like to thank our course assistant Yufeng, who was prompt in answering our concerns throughout the course. Lastly, we would like to thank all my classmates who made this Advanced Control Systems Integration course interesting and enjoyable.

REFERENCES

[1] Y. Chen and N. O. Pérez-Arancibia, "Generation and real-time implementation of high-speed controlled maneuvers using an autonomous 19-gram quadrotor," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 3204–3211.

[2] J. H. Gillula, H. Huang, M. P. Vitus, and C. J. Tomlin, "Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 1649–1654.

[3] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012.

[4] E. Kaufmann, A. Loquercio, R. Ranftl, M. Mueller, V. Koltun, and D. Scaramuzza, "Deep drone acrobatics," 07 2020.

[5] R. W. Beard, "Quadrotor dynamics and control," *Brigham Young University*, vol. 19, no. 3, pp. 46–56, 2008.

[6] J. Förster, "System identification of the crazyflie 2.0 nano quadrocopter," B.S. thesis, ETH Zurich, 2015.

[7] A. Majumdar, "Introduction to robotics."