White-Box Identification of Transfer Function for Dynamics of A Small-Scale Quadcopter

Rohen Agarwal, Eric Wan, Randy Chen, Eric Dobek*
University of Illinois at Urbana-Champaign, Champaign, IL, 61820, U.S.A

White-box identification, a specific type of system identification, is a numerical scheme commonly applied in control systems to estimate unknown parameters associated with an assumed dynamical model. The initial goal of this project is to identify transfer functions that govern the input-output relationship between the torques, the net thrust, and all transnational and rotational degrees of free dome of the drone. The final result accomplished by the team is the identification of a transfer function that relates the net thrust to the vertical position of the drone. A PD controller is designed and tuned in MATLAB to stabilize the open-loop step response of the identified transfer function.

I. Nomenclature

Amp	=	amplitude of oscillation
t	=	time
T_s	=	sampling time
T_d	=	delay in transfer function
F_{net}	=	net thrust acting on the drone
$U(s)$	=	open-loop transfer function
ω	=	frequency in rad/s
s	=	Laplace domain variable
z	=	complex frequency domain variable
O_z	=	z position of the drone
V_z	=	z velocity of the drone
a_z	=	z acceleration of the drone
h	=	height above ground
UAV	=	unmanned aerial vehicle
A	=	state matrix
B	=	input matrix
C	=	output matrix
D	=	feedthrough matrix
hat	=	cross product
f	=	force vector
τ	=	torque vector
m	=	mass of the drone
J	=	inertia tensor
l	=	distance from center of drone to center of propeller
σ	=	rotor spin rate
k_f	=	force coefficient related to rotor power
k_M	=	moment coefficient related to rotor power
m_1	=	motor power command on rotor 1
m_2	=	motor power command on rotor 2
m_3	=	motor power command on rotor 3
m_4	=	motor power command on rotor 4
Kp	=	proportional gain K_d

*Undergraduate in Aerospace Engineering, University of Illinois at Urbana Champaign, 2021

derivative gain N = Newton

II. Introduction

System identification is a method used to fit experimental data logged from an persistently excited dynamical system to an assumed dynamical model. The objective of applying system identification is to retrieve a plant model such that one can predict the open-loop output response given a set of input signals. This method is commonly used to analyze systems with highly complex dynamics where deriving the equations of motion is not a simple task. Our group is motivated to implement system identification because it allows us to partially bypass deriving the full equations of motion of the drone, which yields a framework rather different from deriving equations of motion based on first principles.

The group's original objective was to carry out a full system identification scheme for the dynamics of both the rotational and transnational degrees of freedom of the drone. In the end, the team was able to successfully identify a transfer function that maps the input net thrust to the output vertical position. The team follows three phases in this project. In the first phase, the group collects data from the drone that logs the net force and the vertical position; in the second phase, the group analyzes the transfer function identified in the system identification toolbox that describes the single-input single-output correlation in the vertical motion of the drone; in the third phase, the group designs and tunes a PD controller in Simulink to ensure a converging open-loop step response of the transfer function. Section III provides the the group's method of design and analysis in detail; section IV presents an analysis of the open-loop transfer function of the identified plant and proposes a PD control design; section V and VI summarizes the key findings in this project and motivations to future applications. A critical step and the most challenging task our team has resolved is the design of input signals. It is required that input signals must persistently excite the dynamical response of the drone (persistency of excitation); otherwise, the data collected will not yield an accurate open-loop transfer function. Commanding the drone to hover at a fixed altitude with fixed input frequency would not produce valid data for system identification.

Hoffer [3] outlines the critical steps in implementing system identification on a small-scale UAV: frequency range, persistent excitation, duration of excitation, types of inputs, and model verification inputs. Hoffer [3] uses the batch least squares algorithm on his system identification design and concludes that the identified dynamical model fits well to the measured output in experimental data. Compared to the Hoffer's methodology ,our group utilizes the system identification toolbox in MATLAB to identify the plant model of the unknown dynamical system rather than writing a code by ourselves. As Hoffer [3] points out, there is currently not much research into system identification methods for multi-rotor UAVs because PID controllers can be implemented without knowledge of the actual system dynamics. Therefore, our group aims to explore the unexplored: analyzing the feasibility of applying system identification on the Crazyflie drone.

III. Methods

A. Hardware

The hardware used in this project is the Crazyflie 2.0 drone designed by Bitcraze. The specifications of on-board components are summarized in Table 1 on the next page.

Table 1 Crazyflie 2.0 Specifications

Onboard microcontroller	STM32F405
Weight	27 g
Gyroscopes	3
High Precision Pressure Sensors	1
IMU	3-axis accelerometer and gyroscope
Size	$92 \times 92 \times 29$ mm

A body-fixed reference frame for the drone is shown in Fig 1

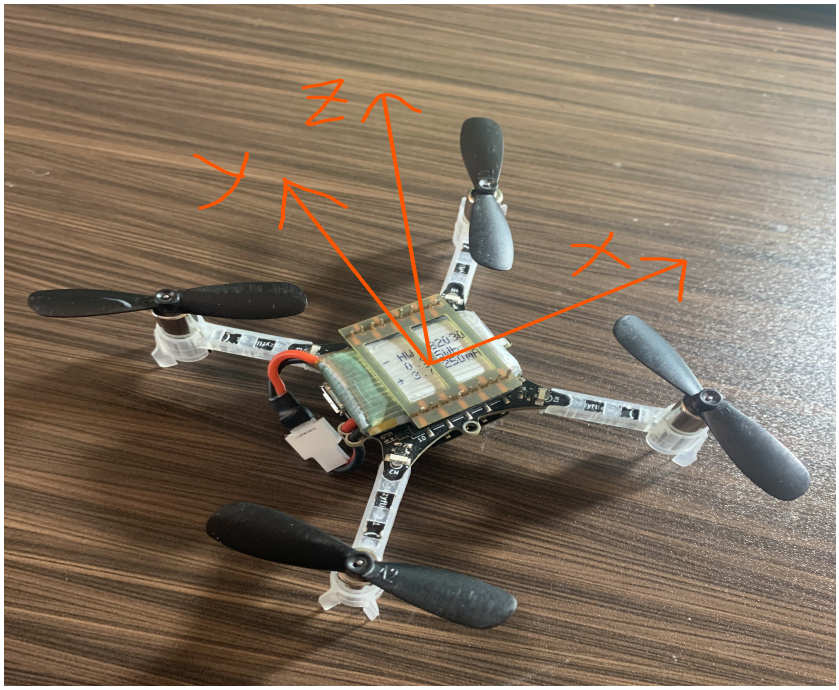


Fig. 1 Body-fixed frame

B. Theoretical Foundation

Implementing white-box identification on the vertical motion of the drone requires a dynamical model with unknown parameters. We obtain the unknown dynamical model by applying Newton's Second law:

$$F_{net} = ma_z = m\ddot{O}_z \tag{1}$$

To obtain an equivalent expression as shown in Eqn 2, the Laplace transform is applied assuming zero initial conditions, which yields the following transfer function:

$$\frac{O_z(s)}{F(s)} = U(s) = \frac{1}{ms^2} \tag{2}$$

In reality, we predict that more terms will appear in the transfer function $U(S)$ due to noise and aerodynamic drags that we have excluded in applying Newton's Second law. The unknown parameters to be identified are thus related to aerodynamic drag and moment. The system identification toolbox estimate model parameters bu minimizing the error between the predicted model output to the measured model output. In this project, the predicted model output is a dynamical model that predicts how the vertical position of the drone varies with a given input, and the measured model output is the actual vertical position of the drone given the same input. The output for a linear model is given by [4]

$$y_{\text{model}}(t) = Gu(t) \tag{3}$$

where G is the transfer function. The error to be minimized is

$$v(t) = y_{\text{meas}}(t) - y_{\text{model}}(t) \tag{4}$$

y_{model} refers to the predicted response associated with a given input and past measurements of the output [4].

C. Implementation

1. Procedures

To successfully collect data on the open-loop behavior of the drone, persistency of excitation must be satisfied [1] because any real-world dynamcial systems, including our drone, have various frequency modes. A transfer function

obtained from the frequency spectrum in which we excite the dynamics of the system dictates the input-output correlation only in that particular frequency spectrum. Persistency of excitation can be achieved through a frequency sweep in the form of sinusoids, doublet, and singlet [3]. The implementation of white-box identification is divided into the following steps. First, we modified the firmware code such that a sinusoidal input injection was activated through the net thrust. Second, we logged the input-output data from the vertical motion of the drone. Next, we assumed a second order transfer function for the vertical motion of the drone and fit the collected data to this transfer function using the system identification toolbox in MATLAB. Our final procedure is to design a controller in simulation that stabilized the vertical motion of the drone.

To persistently excite the dynamics of the drone, we modify the input net thrust of the drone in the firmware code by superposing a sinusoidal input net thrust with an input frequency sweep. The rationale is that frequency sweep is frequently used in exciting the dynamics in system identification of a system. The sinusoidal input has the following form:

$$F_{sin} = Amp * sin(\omega * t) \tag{5}$$

Our procedure is formulated as follows. The sinusoidal input will be activated when the drone stays stabilized and hovers at a fixed altitude; this is done by adding the sinusoidal thrust to the hover thrust. The unmodified input thrust required to keep the drone stabilized is controlled by a feedback loop and is deactivated once the sinusoidal input is activated. The total flight time is 90 seconds. The frequency is swept in segments of 5 seconds, where the feedback controller is deactivated for the first 4 seconds in each segments. A frequency sweep in the range of 0 rad/s to 10 rad/s is tested, and the sinusoidal input amplitude is kept the same for each flight test. The amplitudes we have experimented with are 0.03 N, 0.02 N, 0.015N, 0.0125N, 0.01N, and 0.007N. During the flight tests, the drone frequently hits the ground or the ceiling in our testing environment, effectively corrupting our data. We identify that the sinusoidal input amplitude plays a key factor here. After some trial and error, we successfully kept the drone in the air during the entire flight by using a sinusoidal input amplitude of 0.015 N. This will cause the drone to both oscillate and drift and thus the dynamics become unstable.

The major challenge our team has encountered and managed to resolve is satisfying persistency of excitation. On the theoretical side, the team did not have prior knowledge regarding what input frequency range or amplitude should be chosen to produce a sufficiently excited dynamical system; on the implementation side, our team encountered the difficulty of getting access to a time variable in the firmware code such that the sinusoidal input can be activated. These two challenges were addressed in the following way: we realize that one of the input arguments to the controller function in the firmware code is a default time variable that logs the time as soon as the drone is turned on. We then define a new time variable and initialize it as zero, but set it equal to the value of the default time variable once the observer is reset. The true time, which starts at 0 when the drone receives a input command, is then obtained by subtracting the new time variable from the default time variable. The observer will be reset every time we run the client code. The following code snippet illustrates in detail how the firmware code and the client code is modified:

```
void controllerAE483(control_t *control,
                     setpoint_t *setpoint,
                     const sensorData_t *sensors,
                     const state_t *state,
                     const uint32_t tick)
{
  if (RATE_DO_EXECUTE(ATTITUDE_RATE, tick))
  {


    // Increment time

    t = (((float)(tick - start_tick)) / 1000.0f);

    // Change Frequency Omega

    if ((t > st) && (t < st + tt))
    {
      w = ws; // (rad/s)
```

}

In the code snippet shown, $start_tick is the new time variable we have defined, and tick is the default time variable. w is the initial valu$

$if((t_omega > 5))w = 0.0f$; With this method our team successfully excited the dynamics of the drone and collected valid data, and the amplitude is chosen as 0.015 N and the frequency is swept from 3.0 rad/s to 10.0 rad/s.

2. Data Collection and Processing

Once data are collected, we verify if the frequency ω in the sinusoidal input change and examine how the vertical position O_z varies with the time. It is expected that O_z as a function of time resembles the shape of a sinusoid,and as ω increases, the sinusoid becomes more squeezed. Before the time-domain data is imported into MATLAB, we filter the data to remove the portion in which the sinusoidal input frequency ω is 0. Then we import our time-domain of our input-output data (f_z to O_z) into MATLAB and filter the data further by removing the means. This allowed us to find models without modelling the absolute equilibrium in physical units [7]. Next, we split up the time-domain data into estimation set and validation set. The estimation set consists of the first 50 seconds of the time-domain data, and the validation set consists of the remaining data as shown in 2. The red portion indicates the estimation set, and the purple portion indicates the validation set. To find a transfer function in the system identification toolbox, a polynomial model estimation is initialized to fit our data to an ARX model to determine the best-fit order of our dynamical model structure (see details in Appendix). Three coefficients associated with the ARX model N_a,N_b, and N_k are returned, and N_k is related to the amount of delay in the to-be-determined transfer function through Eqn. 6

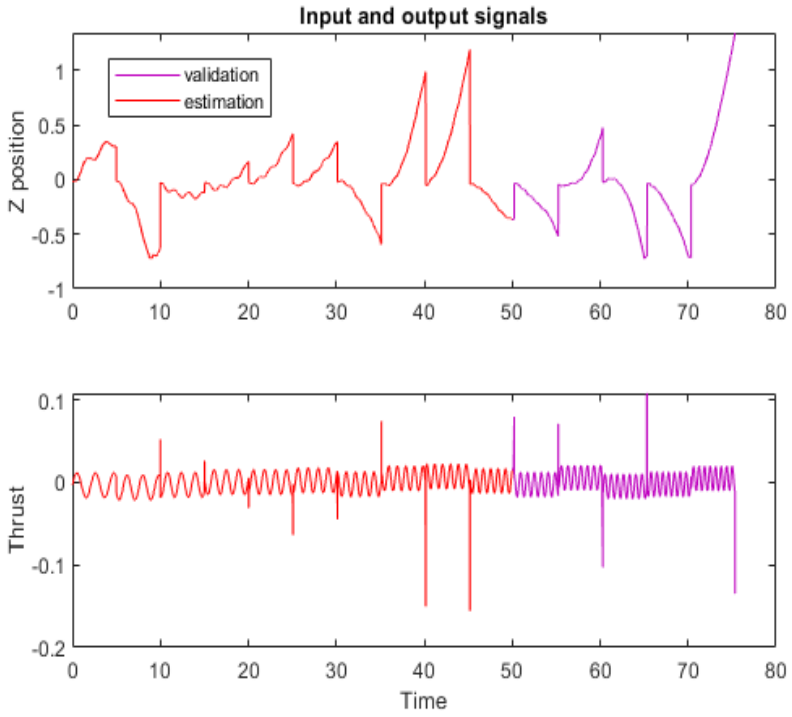


Fig. 2 Estimation and validation data partition

$$T_d = (N_k - 1) \times T_s \tag{6}$$

where T_d refers to the delay measured in seconds and T_s is the sample time interval (0.01 seconds for our drone). The ARX model returned the following values for the coefficients: $N_a = 3$, $N_b = 9$, and $N_k = 3$. This amounts to a delay of 0.02 seconds in the to-be-determined transfer function, which was fed into the transfer function approximation tab in the system identification toolbox (we designated this delay time), and a continuous-time transfer function is returned (see section IV).

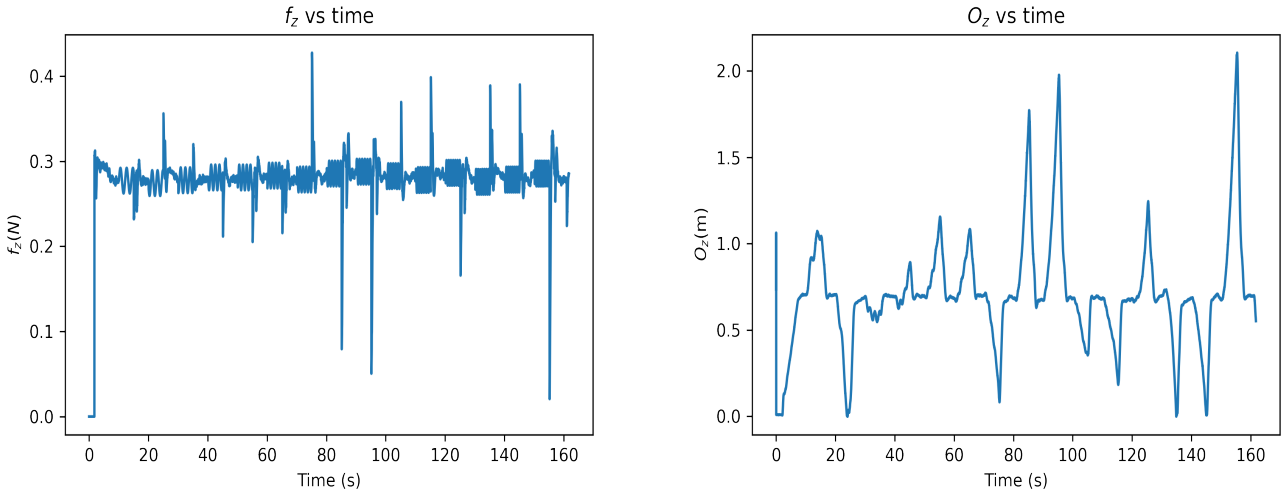
3. Controller Design

After a continuous-time transfer function is identified through the system identification toolbox, we initiate and tune a continuous-time PD controller (PD block) in Simulink such that a reasonably rapid-converging step response is achieved in the open loop. In reality our system is not continuous-time, and therefore we transform the continuous-time transfer function to the discrete domain based on the sampling time in the time-domain data,T_s = 0.01 s. The transformation is achieved by the 'c2d' command in MATLAB and a zero-order hold transformation method is chosen because it is the default setting in the 'c2d' command. The corresponding bode plots and linear step response plots generated from the continuous and discreet open-loop systems are compared (see section IV).

IV. Results

A. Excitation of Dynamics

The net thrust F_{net} and the vertical position O_z from the time-domain data logged from the drone are in Fig 3



(a) f_z vs time **(b) O_z vs time**

Fig. 3 Input and output signals

The results are in accordance with our prediction: the net thrust f_z follows the shape of a sinusoid, and we observe that as the time increases the sinusoids are look more squeezed together. This is due to the increasing sinusoidal frequency we defined in the firmware code. Note that the results shown here are unfiltered, and therefore the portion where frequency is 0 needs to be filtered out. The sinusoidal frequency spectrum is plotted as a function of time as shown in 4

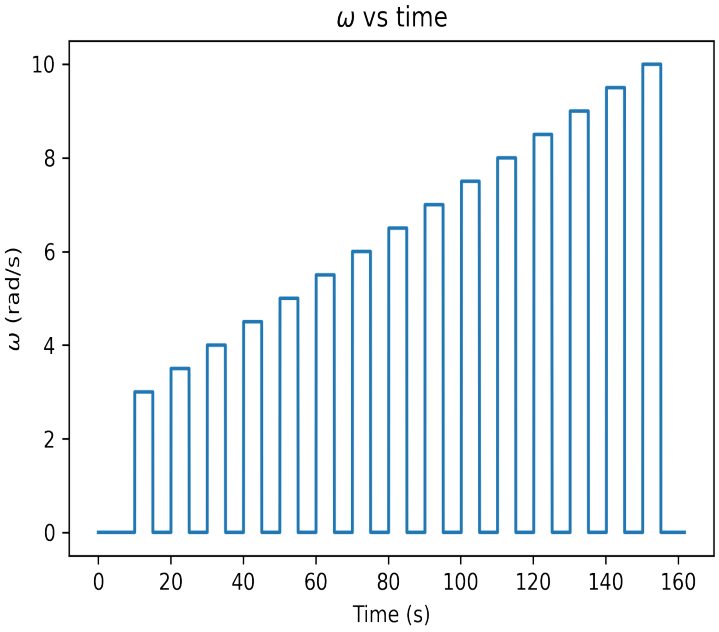


Fig. 4 Sinusoidal frequency vs time

We therefore verify that the sinusoidal input is switched on and off periodically as described in section III. Figure 5 shows the filtered F_{net} and O_z

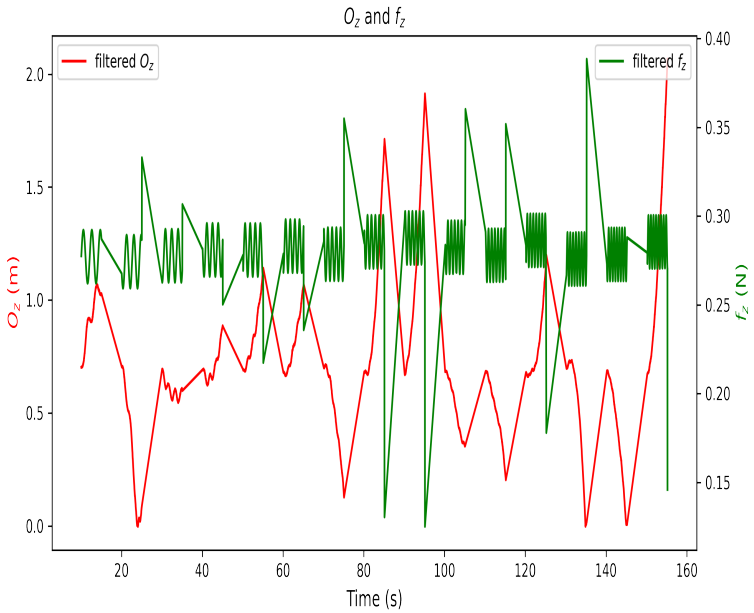


Fig. 5 Filtered data

As can be seen, instead of seeing clear oscillations in the output z position, the drone tends to continue accelerating in an initial direction regardless of sinusoidal thrust input. In order to disambiguate this, we can better visualize the response of the output z position vs input frequency by processing the data using a Fast Fourier Transform.

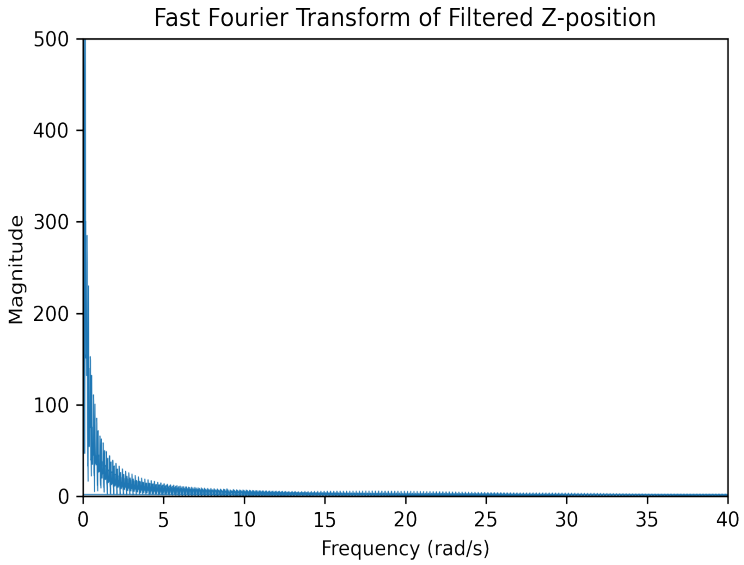


Fig. 6 Fast Fourier Transform of Filtered Z-position

As can be seen, the resulting frequency response for the output z position is still ambiguous. We should see clear peaks at certain frequencies corresponding with the tested frequencies. This tells us that our implementation of the sinusoidal thrust was not ideal. Despite this, we attempted many combinations of input frequencies, amplitudes, time shifts, and stabilization before/after tests, yet could not manage to both create a clear sinusoidal output and avoid crashing of the drone. In future testing, we would focus on tuning the input such that the output has a much clearer sinusoidal response.

The filtered data processed via FFT is then imported into the system identification toolbox to obtain a transfer function.

B. Transfer Function

The transfer function identified from the system identification toolbox is

$$U(s) = \exp\left(-0.02^{*}s\right) * \frac{22.65s + 2.516}{s^2 + 0.1779s + 0.1214} \tag{7}$$

It is noted that the transfer function contains a delay of 0.02 seconds, determined by the N_k term in the best-fit ARX model. This transfer function has 1 zero and 2 poles as graphically illustrated in the pole-zero plot in Fig. 7

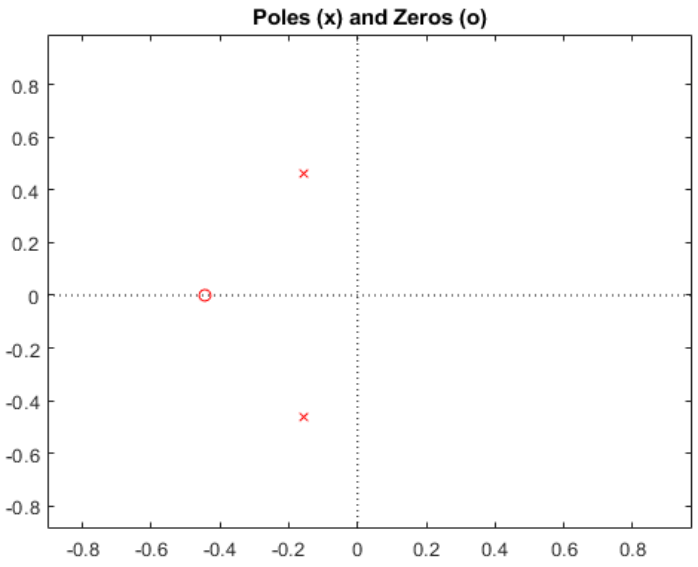


Fig. 7 Pole-zero plot of transfer function

To check how well this transfer function fits to the output validation data (see 2), we inspect the model output plot produced by the system identification toolbox as shown in 8

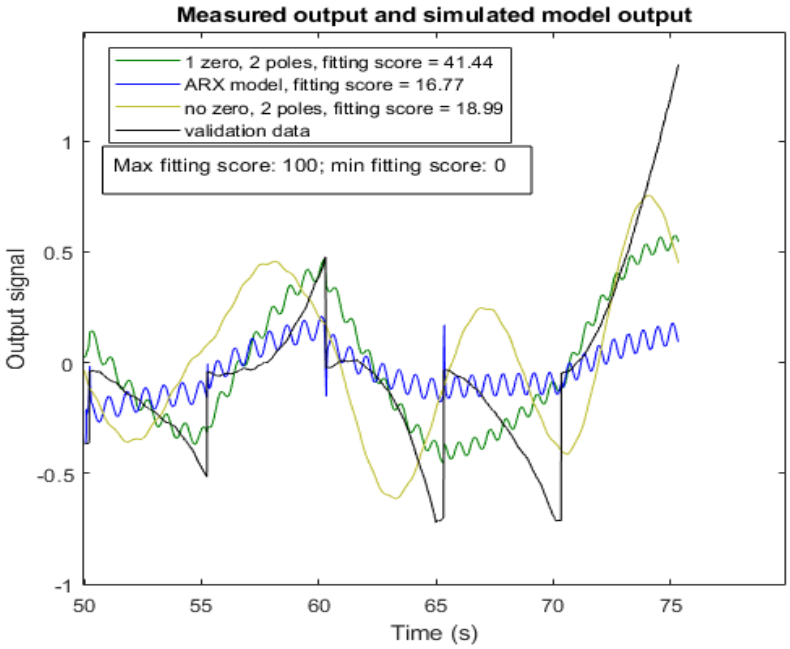


Fig. 8 Agreement between models and measured output

A fitting score is shown in the model output plot. 100 indicates a perfect agreement between the predicted output and the measured output, whereas 0 indicates a poor agreement. The identified transfer function has a fitting score of 41.44. We observe there are some issues with the transfer function : the poles are both negative, indicating it is stable. Moreover, the fitting score is rather low. The details are discussed in section V.

To visually check the stability of the open-loop transfer function in 7, we check the Bode plot in MATLAB as shown in 9

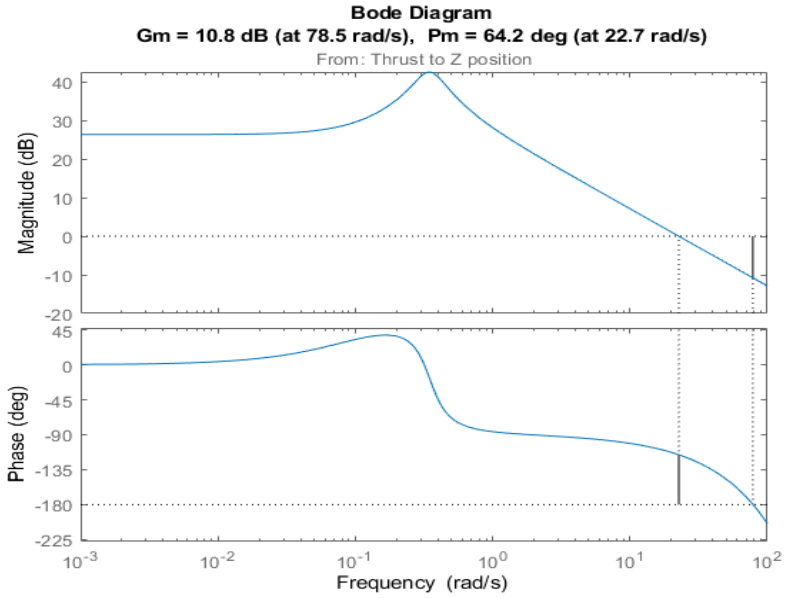


Fig. 9 Bode plot of open-loop transfer function

We observe that the bode plot indeed follows the trend of a generic 2nd order system: as the frequency increase, the magnitude reaches a peak and then descends, whereas the phase has a constantly decreasing trend. As described in section III, we then discretize our continuous-time transfer function into a discrete one. The discretized transfer function is obtained as

$$z^{-2}\frac{0.2265z^{-1}-0.2262z^{-2}}{1-1.998z^{-1}+0.9982z^{-2}} \tag{8}$$

A discrete PD controller is then designed and tuned in Simulink.

C. PD Controller

An open-loop PD controller for the discrete-time transfer function is built in Simulink as shown in 10

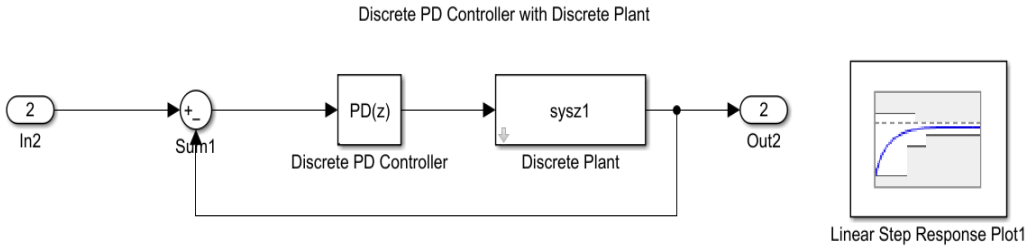


Fig. 10 Simulink model for open-loop response

The step response for the discrete open-loop systems are simulated for 10 seconds. The results are shown in 11.

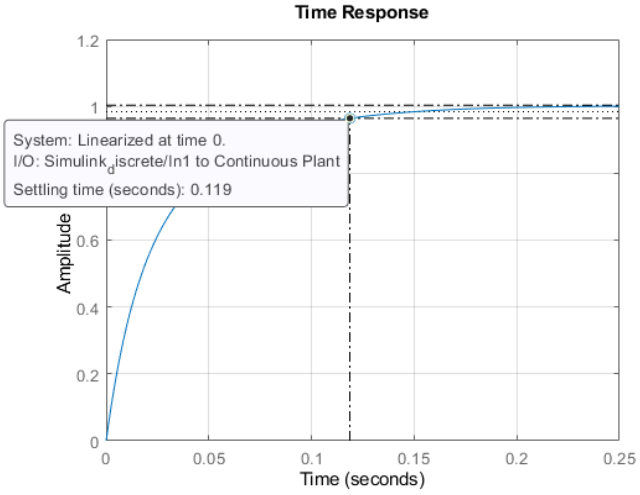


Fig. 11 Step response for the discrete transfer function

The step response for the discrete transfer function has a rather long settling time of 8.75 seconds. The following table shows a comparison the gain coefficients we obtained from Simulink (see Fig. 10) with the gains we obtained in lab No.6 (using LQR) that were confirmed to be capable of stabilizing the drone

Table 2 Gain coefficients comparison

Discrete PD controller	Kp = 1.3826, Kd =0.01050
Gains from lab 6	Kp = 0.8182, Kd = 0.00517

We observe there is a tremendous difference between the gains we obtained in Simulink and the gains we found to be working from previous labs. The reason is discussed in the following section. The proportional and derivative gains are tested on the drone, and the plot shown below illustrates how the drone behaves

V. Discussion

There are several problems with the transfer function and the gain coefficients we have obtained. By looking-at the pole-zero plot in Fig. 7, we notice that the poles are in the left half plane, which already indicates stability. Based on Fig. 8, we also notice our transfer function has relatively poor agreement with the actual measured output. This is likely due to insufficient excitation of the drone. The proportional and derivative gains are tested on the drone. Some other predictions are made regarding the inaccuracy in the transfer function obtained due to to some limitations we have encountered during data collection. First, the input frequency spectrum we applied to excite the dynamics of the Crazyflie drone is very narrow (from 3.0 rad/s to 10.0 rad/s), which cannot fully represent the frequency modes. Therefore, the transfer function we have obtained as well as the discrete PD controller tuned in MATLAB might only work when the drone is operating in our manually-designed frequency spectrum. Second, there is sensor noise included in the data collected from the drone. A general rule of thumb in exciting the dynamics of a system is that the input signal must outweigh the noise. Further study is required to conclude whether or not the input signal we have designed outweighs the noise.

VI. Conclusion

Our team obtained a transfer function that relates the net thrust and the vertical position of the Crazyflie drone designed and tested a discrete PD controller. Work presented in this study serves as a framework for system identification on small-scale quad-copters. In conclusion, our team validates the feasibility of bypassing the derivation of equations of motion to identify the dynamics of the vertical motion of a Crazyflie drone. Further study will be conducted in the future to identify the dynamics of the drone in all degrees of freedom in addition to the vertical motion.

Acknowledgments

Our team would like to thank Professor Timothy Bretl, TA Travis Zook, and Brian Douglass, a control engineer who posts high-quality videos online, for providing careful guidance on our project. We also thank Dr.Dan Block for providing a lab space for our team to collect data from the drone.

References