Quadcopter Robust Controller via μ-Synthesis

Michael Wang , mw732

Advisor: Professor Silvia Ferrari

Cornell University

May 13, 2018

Abstract

Quadcopters are highly nonlinear and coupled dynamical systems that prove to be ubiquitous in the modern society. From hobbyists to military drones, quadcopters need to operate in a variety of environments which are often times unpredictable. Therefore, in this project, a multivariable variable quadcopter controller using structured singular value (μ) synthesis techniques is considered. μ-synthesis is a type robust controller that stabilizes the plant under a bounded set of disturbances. While the exact disturbances are unknown, the bounds on the disturbances are known. Using mathematical formalisms such as structured singular value and linear fractional transformation, which will be discussed in detail later, robust stability and robust performance can be proven. Using Monte Carlo simulations, the efficacy of μ-synthesis is demonstrated for a quadcopter set-point regulation problem.

Introduction and Theory

In this section, concepts in modeling uncertainty and robustness will be introduced. We define uncertainty as the difference between our model and reality. For instance, uncertainty in a control system can involve sensor noise, external disturbances, mass properties, and plant model. We can then define robustness as a characteristic of a controller that maintains the stability of the closed loop system under a pre-determined set of unknown disturbances. While the bounds and distribution of the disturbances must be known *a priori* for the development of the controller, the actual realizations of the disturbances are not known until runtime.

Robustness can be said to be dual to optimality. While optimal controllers, such as the Linear Quadratic Regulator (LQR), seek to minimize a performance cost function regardless of disturbances, robust controllers seek to maintain stability and robust performance over a set of bounded uncertainty. As shown in the seminal paper *Guaranteed Margins for LQG Regulators*, Doyle used a simple counterexample to prove that LQG Regulators have no guaranteed margins [1]. To quantify robustness, we thus have the following definitions [2]:

Robust Stability (RS): When a controller K internally stabilizes every plant sampled from a set of uncertain plants

Robust Performance (RP): When a controller K satisfies performance objectives for every plant sampled from a set of uncertain plants.

The criteria for RS and RP will be explored in detail later. First, Linear Fractional Transformations (LFT) will be explained.

Linear Fractional Transforms

For a given complex matrix partitioned as:

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \tag{1}$$

and given the following block diagram:

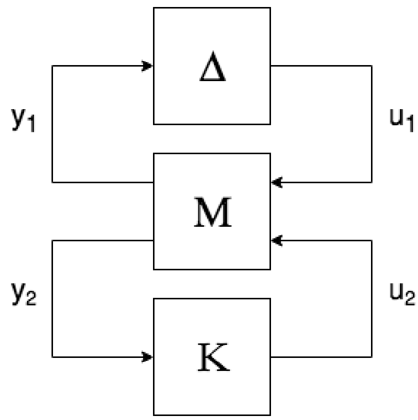


Figure 1: LFT

The following set of equations can be written:

$$\begin{aligned} u_1 &= \Delta y_1 \\ u_2 &= K y_2 \\ \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} &= \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \end{aligned} \tag{2}$$

The upper and lower linear fractional transforms are defined as:

$$\begin{aligned} F_u(M, \Delta) &\triangleq M_{22} + M_{21}\Delta(I - M_{11}\Delta)^{-1}M_{12} \\ F_l(M, K) &\triangleq M_{11} + M_{12}K(I - M_{22}K)^{-1}M_{21} \end{aligned} \tag{3}$$

given that $(I - M_{11}\Delta)$ and $(I - M_{22}K)$ are invertible. One can verify easily by manipulating Equation 2 that $F_u(M, \Delta)$ is the transfer matrix from u_2 to y_2 and $F_l(M, K)$ is the transfer matrix from u_1 to y_1. If K is a controller and Δ is a structured disturbance, $F_u(M, \Delta)$ can be used to synthesize a controller via μ-synthesis while $F_l(M, K)$ can be used to analyze robust stability and performance of the open loop plant M. The inverse of the LFTs are given by:

$$
\begin{aligned}
F_l(M,K)^{-1} &= F_l(M_{li},K) \\
F_u(M,\Delta)^{-1} &= F_u(M_{ui},\Delta)
\end{aligned} \tag{4}
$$

where:

$$
\begin{aligned}
M_{li} &= \begin{bmatrix} M_{11}^{-1} & -M_{11}^{-1}M_{12} \\ M_{21}M_{11}^{-1} & M_{22}-M_{21}M_{11}^{-1}M_{12} \end{bmatrix} \\
M_{ui} &= \begin{bmatrix} M_{11}-M_{12}M_{22}^{-1}M_{21} & M_{12}M_{22}^{-1} \\ -M_{22}^{-1}M_{21} & M_{22}^{-1} \end{bmatrix}
\end{aligned} \tag{5}
$$

Uncertainty Representations

LFTs can be used to represent a variety of uncertainties. The first kind is parametric uncertainty. For instance, if a parameter c is modeled as:

$$c = 2.4 + 0.4\delta_c \tag{6}$$

where δ_c is drawn from a uniform distribution $[-1,1]$. Then c can be replaced by a LFT:

$$c = F_l\left(\begin{bmatrix} 2.4 & 0.4 \\ 1 & 0 \end{bmatrix}, \delta_c\right) \tag{7}$$

Using Equation 5, the equation for $1/c$ can also be found. This technique can be used to represent uncertain parameters in state space models or transfer functions. Corresponding MATLAB functions from the Robust Control Toolbox, `ureal()` and `ucomplex()`, are used to create uncertain parameters [3].

The other type of uncertainty representation is unmodeled dynamics. One subtype of unmodeled dynamics is multiplicative uncertainty. Whereas parametric uncertainty is frequency-independent, multiplicative uncertainty allows the user to specify frequency-dependent uncertainty in the plant. Given the nominal plant transfer function $G(s)$, uncertainty weighting function $W(s)$, and the uncertain dynamics element $\Delta(s)$ where $\max\limits_{\omega\in\mathcal{R}}|\Delta(j\omega)| \leq 1$, the multiplicative uncertainty is:

$$G(1+W\Delta) = F_u(H,\Delta) \tag{8}$$

which can also be represented as a LFT with:

$$H = \begin{bmatrix} 0 & W \\ G & G \end{bmatrix}$$

Furthermore, additive uncertainty can be used:

$$(G + W\Delta) = F_u(J, \Delta) \tag{9}$$

where

$$J = \begin{bmatrix} 0 & W \\ 1 & G \end{bmatrix}$$

The three types of uncertainty representation mentioned above can be mixed and matched to realistically model the uncertainty in a given system. Due to the properties of LFT, a model with a mix of these three uncertainty representations can be expressed in another LFT using the following rules [4]. Given the following systems:

$$G_1(\Delta_1) = \begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} \qquad G_2(\Delta_2) = \begin{bmatrix} A_2 & B_2 \\ C_2 & D_2 \end{bmatrix}$$

with

$$\Delta = \begin{bmatrix} \Delta_1 & 0 \\ 0 & \Delta_2 \end{bmatrix}$$

The LFT addition and multiplication rules are:

$$\begin{aligned} (G_1 G_2)(\Delta) &= \begin{bmatrix} A_1 & B_1 C_2 & B_1 D_2 \\ 0 & A_2 & B_2 \\ C_1 & D_1 C_2 & D_1 D_2 \end{bmatrix} \\ (G_1 + G_2)(\Delta) &= \begin{bmatrix} A_1 & 0 & B_1 \\ 0 & A_2 & B_2 \\ C_1 & C_2 & D_1 + D_2 \end{bmatrix} \end{aligned} \tag{10}$$

Using these simple rules, the whole system can be remodeled as shown in 1, where K is the controller, M is the open loop plan, and Δ is a *structured* block diagonal set of uncertainty. As the system becomes more and more complex, calculating the M matrix by hand becomes harder. MATLAB functions such as `sysic` and `connect` can be very useful in these situations.

Structured Singular Value

We can now introduce the theory of the structured singular value, which is defined as:

$$\mu_\Delta(M) \triangleq \frac{1}{\min\{\bar{\sigma}(\Delta) : \Delta \in \mathbf{\Delta}, det(I - M\Delta) = 0\}} \tag{11}$$

If there is no Δ that makes $(I - M\Delta)$ singular, $\mu_\Delta(M) = 0$. We can use the upper loop in Figure 1 to interpret this (in this case M is $F_l(M, K)$). If $(I - M\Delta)$ is nonsnigular, the only solutions are $u_1 = 0, y_1 = 0$. If $(I - M\Delta)$ is singular, there are infinite many solutions and $\|u_1\|, \|y_1\|$ can be arbitrarily large. We then call these systems robustly stable and robustly unstable, respectively. $\mu_\Delta(M)$ is thus a measure of smallest structured Δ that causes robust instability [4]. Since the structured singular value is hard to compute, we instead compute the bounds:

$$\rho(M) \leq \mu_\Delta(M) \leq \bar{\sigma}(M) \tag{12}$$

We thus arrive at the important result for robust stability. Let $\beta > 0$. M is well-posed and internally stable or all $\Delta \in \mathbf{\Delta}$ with $\|\Delta\|_\infty < \frac{1}{\beta}$ *if and only if* [2]:

$$\sup_{\omega \in \mathcal{R}} \mu_\Delta\left(G(j\omega)\right) \leq \beta \tag{13}$$

This criteria can be verified computationally by using MATLAB commands `mu()` and `dksyn()`. It turns out, robust performance can be posed as a robust stability proof on an augmented Δ. Given a new fictitious Δ_F and the augmented uncertainty block:

$$\Delta_A = \begin{bmatrix} \Delta & 0 \\ 0 & \Delta_F \end{bmatrix}$$

A system G has robust performance if [2]

$$\sup_{\omega \in \mathcal{R}} \mu_{\Delta_A}\left(G(j\omega)\right) \leq \beta \tag{14}$$

The goal of μ-synthesis is then to minimize over all stabilizing controllers K the peak value of the structured singular value of the closed loop transfer function:

$$K^* = \max_{K_{stab}} \mu_\Delta(F_l(P, K)(j, \omega)) \tag{15}$$

Due to the difficulty of computing μ exactly, we instead resort to computing the upper bound.

$$\mu_\Delta(M) \leq \inf_{D \in \boldsymbol{D}} \bar{\sigma}(DMD^{-1}) \tag{16}$$

where the set $\boldsymbol{D}$ represents matrices with the property $D\Delta = \Delta D$. The D-K iteration is reformulated into [3]:

$$K^* = \min_{K_{stab}} \min_{D \in \boldsymbol{D}} \left\| D F_l(P, K) D^{-1} \right\|_\infty \tag{17}$$

where D is stable and min-phase. This optimization problem is implemented by the MATLAB function `dksyn()`. Due to the two minimization processes, the D-K iteration holds K fixed and optimize D, and vice versa, until convergence. We thus have powerful tools to analyze the robust stability and performance of uncertain systems and implement a robust controller via μ-synthesis.

Procedure

In this section, the procedure for synthesizing a μ-synthesis controller will be described. First, we need to linearize the nonlinear dynamics of the quadcopter, since the theory developed earlier is based on Linear Time Invariant (LTI) systems.

Linearize Dynamics

The full quadcopter nonlinear dynamics are given by [5]:

$$\begin{aligned}
\dot{u} &= (vr - wq) + g \sin\theta \\
\dot{v} &= (wp - ur) - g \sin\phi \cos\theta \\
\dot{w} &= (uq - vp) - g \cos\phi \cos\theta + \frac{u_1}{m} \\
\dot{p} &= \frac{(I_y - I_z)}{I_x} qr - \frac{J_R}{I_x} q\Omega + \frac{u_2}{I_x} \\
\dot{q} &= \frac{(I_z - I_x)}{I_y} pr + \frac{J_R}{I_y} p\Omega + \frac{u_3}{I_y} \\
\dot{r} &= \frac{(I_x - I_y)}{I_z} pq + \frac{u_4}{I_z}
\end{aligned} \tag{18}$$

$$
\begin{aligned}
\ddot{x} &= (\sin\psi\sin\phi + \cos\psi\sin\theta\cos\phi)\frac{u_1}{m} \\
\ddot{y} &= (-\cos\psi\sin\phi + \sin\psi\sin\theta\cos\phi)\frac{u_1}{m} \\
\ddot{z} &= -g + (\cos\theta\cos\phi)\frac{u_1}{m} \\
\dot{\phi} &= p + (q\sin\phi + r\cos\phi)\tan\theta \\
\dot{\theta} &= q\cos\phi - r\sin\phi \\
\dot{\psi} &= \frac{q\sin\phi + r\cos\phi}{\cos\theta}
\end{aligned}
\tag{19}
$$

where $\Omega \triangleq -\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4$. The control inputs are mapped directly from external forces and moments:

$$
\begin{bmatrix} thrust \\ \boldsymbol{\tau} \end{bmatrix} = \begin{bmatrix} C_T(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ C_T l(\Omega_4^2 - \Omega_2^2) \\ C_T l(\Omega_3^2 - \Omega_1^2) \\ C_D(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \end{bmatrix} \triangleq \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}
\tag{20}
$$

The full equations of motion will be linearized about a quasi-steady operating condition characterized by $\begin{bmatrix}\phi & \theta & \psi\end{bmatrix} = \begin{bmatrix}0 & 0 & \psi^*\end{bmatrix}$. We then assume small p, q, r, which simplifies our state to $\boldsymbol{x} = \begin{bmatrix}x & y & z & \dot{x} & \dot{y} & \dot{z} & \phi & \theta & \psi & \dot{\phi} & \dot{\theta} & \dot{\psi}\end{bmatrix}^T$. Assuming small perturbations, $\dot{\phi} \approx p$, $\dot{\theta} \approx q$, and $\dot{\psi} \approx r$. Using small angle approximations, we can also say that $\cos\phi \approx \cos\theta \approx 1$, $\sin\phi \approx \phi$, and $\sin\theta \approx \theta$. Assuming small gyroscopic effects Ω, the simplified equations of motion are then given by [5]:

$$
\begin{aligned}
\ddot{x} &= (\phi\sin\psi^* + \theta\cos\psi^*)\frac{u_1}{m} \\
\ddot{y} &= (-\phi\cos\psi^* + \theta\sin\psi^*)\frac{u_1}{m} \\
\ddot{z} &= -g + \frac{u_1}{m} \\
\\
\ddot{\phi} &= \frac{u_2}{I_x} \\
\ddot{\theta} &= \frac{u_3}{I_y} \\
\ddot{\psi} &= \frac{u_4}{I_z}
\end{aligned}
\tag{21}
$$

By finding the Jacobians of Equation 21, we can put equations in linear form:

$$A = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \sin\psi^* \frac{u_1^*}{m} & \cos\psi^* \frac{u_1^*}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\cos\psi^* \frac{u_1^*}{m} & \sin\psi^* \frac{u_1^*}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{22}$$

$$B = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ (\phi^* \sin\psi^* + \theta^* \cos\psi^*)\frac{1}{m} & 0 & 0 & 0 \\ (-\phi^* \cos\psi^* + \theta^* \sin\psi^*)\frac{1}{m} & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{l_x}{I_x} & 0 & 0 \\ 0 & 0 & \frac{l_y}{I_y} & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} \end{bmatrix} \tag{23}$$

We assume the system is fully observable $C = I_{12x12}$. The linearized equations of motion is finally:

$$\begin{aligned} \delta\dot{\boldsymbol{x}} &= A\delta\boldsymbol{x} + B\delta\boldsymbol{u} \\ y &= C\delta\boldsymbol{x} \end{aligned} \tag{24}$$

with $\delta\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x}^*$, $\delta\boldsymbol{u} = \boldsymbol{u} - \boldsymbol{u}^*$, and $\begin{bmatrix}\phi^* & \theta^* & \psi^* & u_1^*\end{bmatrix} = \begin{bmatrix}0 & 0 & \psi^* & mg\end{bmatrix}$

Uncertainty Modeling

Using the techniques developed earlier, we model the quadcopter system as:

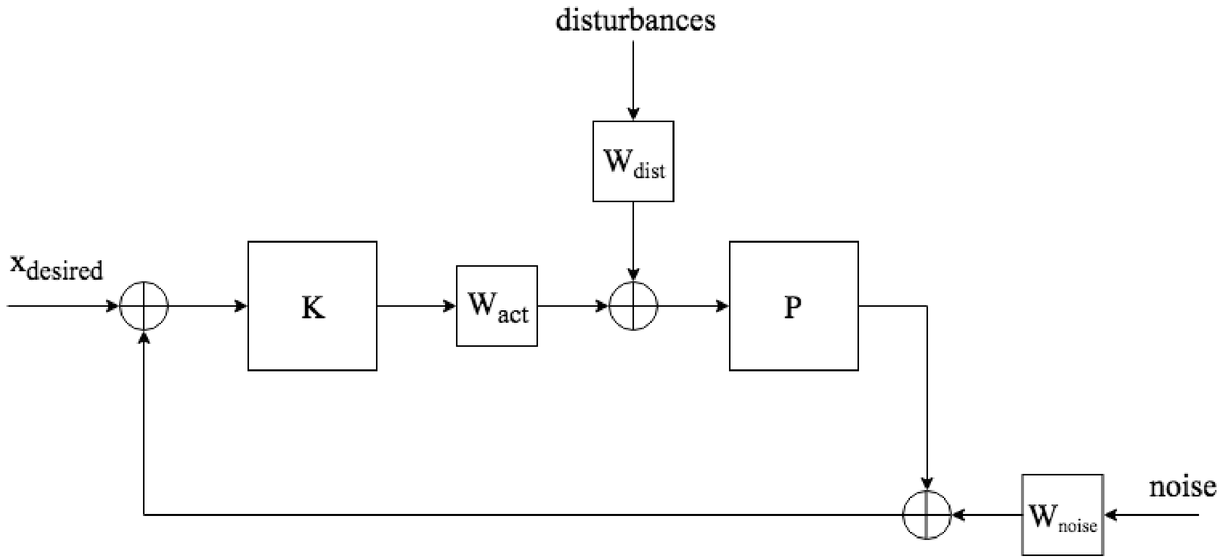


Figure 2: Uncertain Quadcopter Model

We use parametric uncertainty to model potentially uncertain parameters in the plant P:

Parameter	Nominal	± % Uncertainty
l_x	0.422 m	30
l_y	0.422 m	30
m	2.1 kg	30
I_x	2.1385 kg/m^2	30
I_y	2.1385 kg/m^2	30
I_z	3.7479 kg/m^2	30
τ	0.1 s	30
γ	0.01 s	30

Table 1: Parametric Uncertainty

Other model uncertainties include actuator delay and lag:

$$W_{act} = \frac{1}{\tau s + 1} \frac{-\gamma s + 1}{\gamma s + 1} \tag{25}$$

external disturbances such as random wind gusts:

$$W_{dist} = diag([2,\ .4,\ .4,\ .01]) \tag{26}$$

and sensor noise:

$$W_{noise} = diag([.1,\ .1,\ .1,\ .01,\ .01,\ .01,\ .0873,\ .0873,\ .0873,\ .00873,\ .00873,\ .00873]) \quad (27)$$

The *disturbance* and *noise* inputs are modeled as zero-mean unit-variance random numbers $N(0,1)$ and their dimensions match the dimensions of control inputs and state vector respectively

MATLAB Robust Control Toolbox

We will be using MATLAB's Robust Control Toolbox to implement the μ-synthesis controller [3], [6]. Please refer to the MATLAB code section at the end for details. One can easily build the necessary state space representations and transfer functions given in the above section. To connect these models, one can utilize the function `connect()` to obtain a system object M by specifying inputs and outputs of each function block. Arrange the order of the inputs and outputs such that the controller inputs (state vector) and outputs (controls) are arranged towards the end in system M. We can then use `dksyn()` to synthesize a controller K using μ-synthesis through D-K iteration. The output controller K is given as a state space object with matrices A, B, C, and D. The closed loop system can be easily simulated using `lsim()`. If Simulink is desired, one would need to use a state-space block to implement this controller:

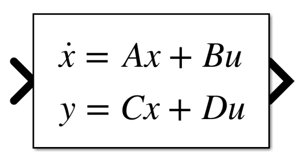


Figure 3: Simulink State Space Block

Simulation Results

First, we will use to `lsim()` to simulate our μ-synthesis controller with the linear model. The noise and disturbances are provided in Table 1. The desired set-point is given as $\boldsymbol{x}_{desired} = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & \frac{\pi}{12} & 0 & 0 & 0 \end{bmatrix}^T$. The initial conditions are all zero. A Monte Carlo simulation of 30 samples is conducted and plotted against the nominal model with no disturbance, noise, or uncertainty.

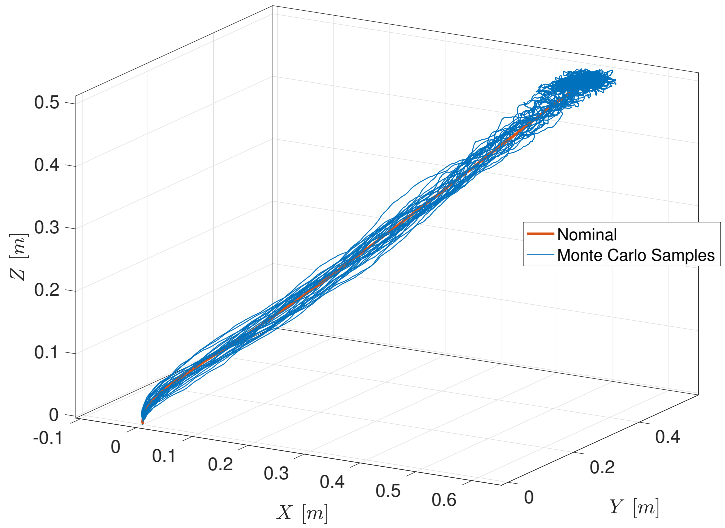


Figure 4: μ-Synthesis on Linear Dynamics 3D plot

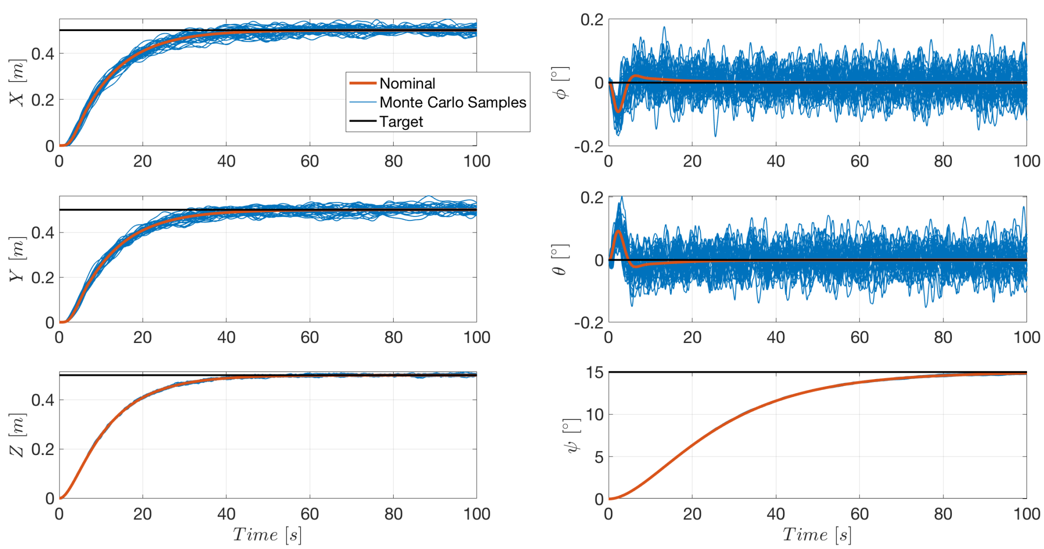


Figure 5: μ-Synthesis on Linear Dynamics State

The orange line represents the nominal model while the blue lines represent samples from the Monte Carlo simulation. As shown, even with disturbance, noise, and uncertainty, the μ-synthesis stabilized and still converged to the set-point. In addition, the controller also has decent disturbance rejection when tasked to maintain at the set-point after arrival. The structured singular value $\mu_\Delta(M)$ is computed as:

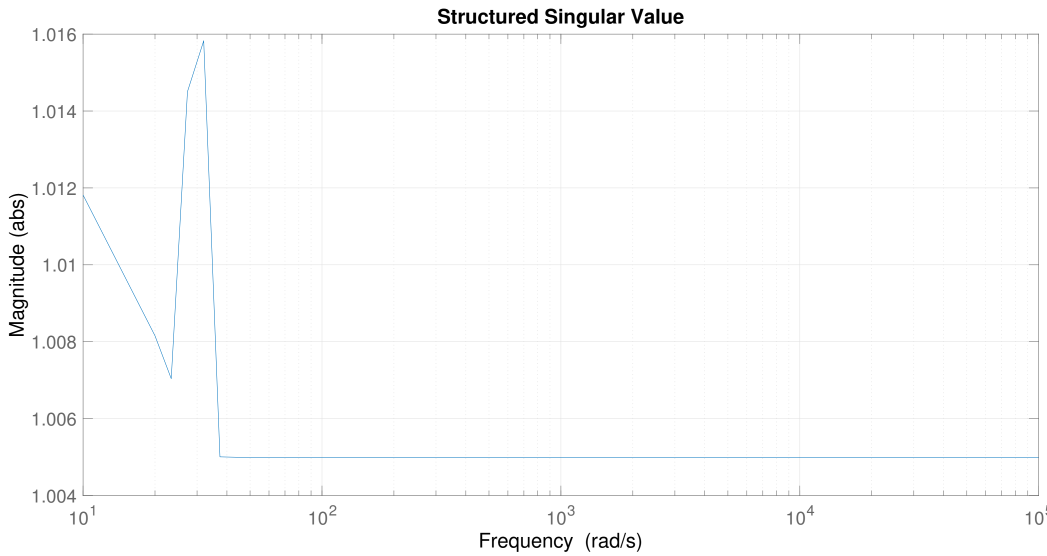


Figure 6: Structured Singular Value

As shown in Figure 6, the maximum structured singular value for this closed loop system is approximately 1.016. While it does not explicitly satisfy robust stability (less than 1), it is close enough such that the controller is fairly robust. Note that this is an upper bound on the structured singular value; as a result, it is possible for the actual SSVs to be below 1. However, without computing the actual SSVs (which is difficult), we cannot guarantee robust stability in this case. Note that it is possible to dial down the parametric uncertainties, disturbance, and noise to drive the upper bound of SSV explicitly below 1; however, in this case, I decided to accurately model the uncertainties that match real world conditions.

Now, the μ-synthesis controller will be tested on the full nonlinear model in Simulink. Please look at the Simulink section for details. For comparison, a test Linear Quadratic Regulator (LQR) is synthesized using MATLAB `lqr()` command:

$$K = lqr(A, B, Q, R) \qquad Q = I_{12x12}; \qquad R = 100 I_{4x4} \tag{28}$$

The matrices A and B are given by Equations 22 and 23 respectively. The same disturbances, sensor noise, and parameter uncertainties are implemented in the Simulink Model. Monte Carlo simulation of 20 samples is conducted. The desired state vector is $\boldsymbol{x}_{desired} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \frac{\pi}{12} & 0 & 0 & 0 \end{bmatrix}^T$

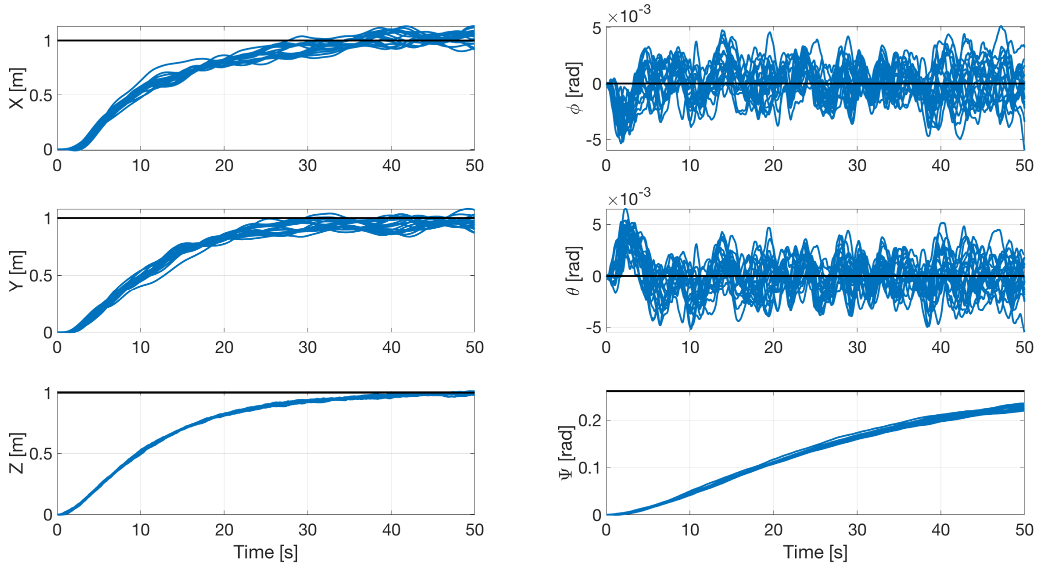


Figure 7: μ-Synthesis on Nonlinear Dynamics

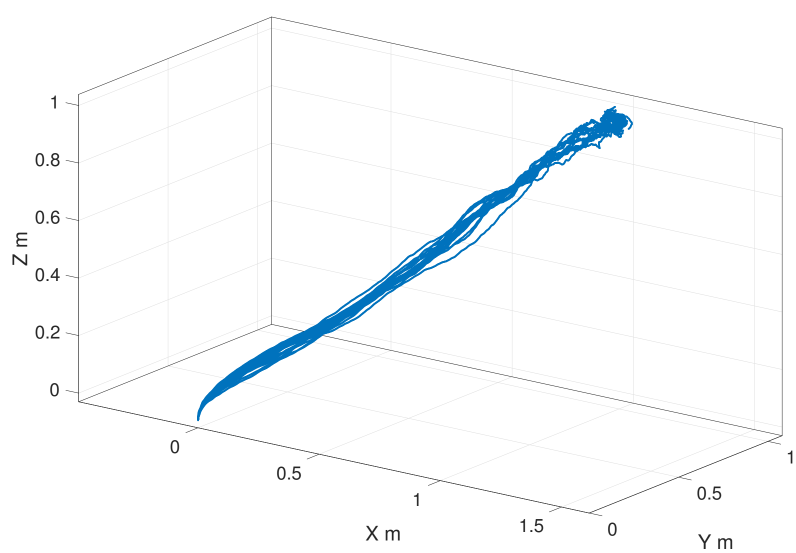


Figure 8: μ-Synthesis on Nonlinear Dynamics 3D Plot

As shown in Figures 7 and 8, the μ-synthesis controller is more prone to noise and disturbances when compared to the linear model simulation. This is expected, since the μ-synthesis controller is based on linear dynamics that neglected higher order effects such as the gyroscopic term. Under the exact conditions, the LQR performed:

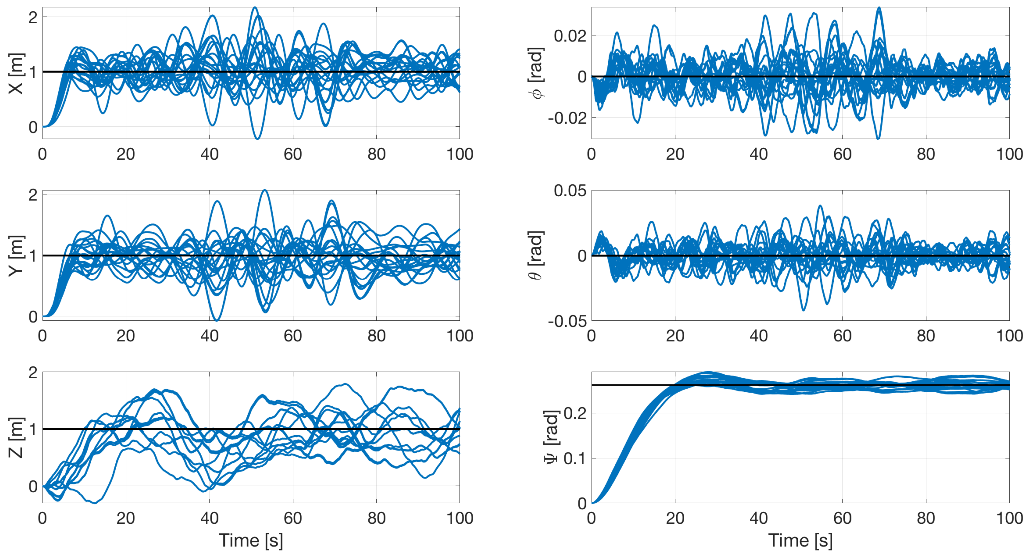


Figure 9: LQR on Nonlinear Dynamics

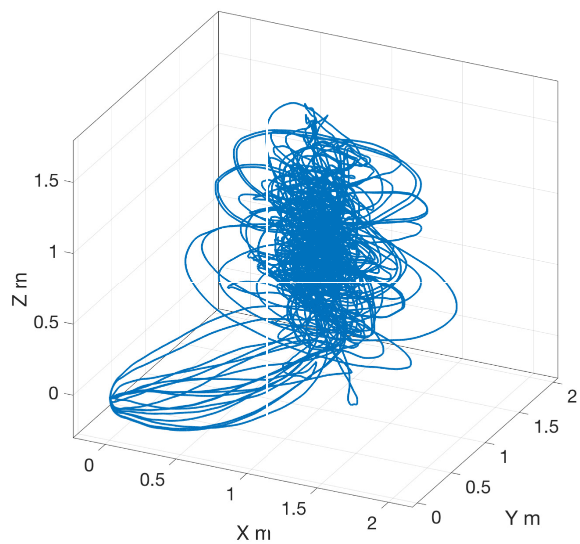


Figure 10: LQR on Nonlinear Dynamics 3D Plot

As shown in Figures 9 and 10, the LQR performed very poorly with the same conditions. The controller is not stable for any of the cases and lacks robustness. This demonstrates importance of incorporating robustness in a controller.

Conclusion and Future Work

A robust controller for a quadcopter using μ-synthesis is presented and implemented in this paper. As shown in the simulation results, the controller is able to track virtually any set-point in inertial position and heading. Additionally, Monte Carlo simulations were conducted by varying the values of the uncertain parameters as well as adding external disturbances and sensor noise, which demonstrated the robustness of the μ-synthesis controller. Furthermore, μ-synthesis is compared with LQR, simulated on full nonlinear dynamics, which demonstrated that μ-synthesis is much more robust than LQR. One disadvantage of μ-synthesis controller is that the dynamics have to be linearized. In addition, μ-synthesis does not optimize towards any performance cost function. Furthermore, the D-K iteration used to synthesize the controller is not guaranteed to converge. As a result, possible future work include higher fidelity modeling of uncertainty of actuators as well as external disturbances and noise, synthesis with feedback linearization, and implementation in an actual quadcopter.

References

[1] J. Doyle, "Guaranteed margins for lqg regulators," *Automatic Control, IEEE Transactions*, vol. 23, pp. 756 – 757, 09 1978.

[2] K. Zhou, J. C. Doyle, and K. Glover, *Robust and Optimal Control.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

[3] G. Balas, J. Doyle, K. Glover, A. Packard, and R. Smith, "Mu-analysis and synthesis toolbox," 1998.

[4] J. Doyle, A. Packard, and K. Zhou, "Review of lfts, lmis, and mu," *Proceedings of the 30th Conference on Decision and Control*, pp. 1227– 1232, 12 1991.

[5] S. Ferrari and T. Wettergren, *Information-driven Planning an Control.* CRC, Prentice Hall, 2018.

[6] G. Balas, R. Chiang, A. Packard, and M. Safonov, "Robust control toolbox," 1998.

[7] S. Bouabdallah, P. Murrieri, and R. Siegwart, "Design and control of an indoor micro quadrotor," in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 5, pp. 4393–4398 Vol.5, April 2004.

[8] G. M. Hoffmann, H. Huang, S. L. Wasl, and E. C. J. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," in *In Proc. of the AIAA Guidance, Navigation, and Control Conference*, 2007.

[9] K. Zhou and J. Doyle, *Essentials of Robust Control.* Prentice Hall Modular Series for Eng, Prentice Hall, 1998.

[10] A. Packard and J. Doyle, "The complex structured singular value," *Automatica*, vol. 29, pp. 71–109, jan 1993.

MATLAB Code

```matlab
%% Quadcopter Robust Control
clear; close all; clc;
% state = [x y z xd yd zd phi th psi phid thd psid]
% control = [u1 u2 u3 u4];


% uncertain parameters
g = 9.807; % m/s^2
m_nom = 2.1; % kg
l_nom = 0.422; % m
tau_nom = 0.001; % s 0.1
gamma_nom = 0.001;
lx = ureal('lx', l_nom, 'Percentage', 30); % m
ly = ureal('ly', l_nom, 'Percentage', 30); % m
m = ureal('m', m_nom, 'Percentage', 30); % kg
Ix = ureal('Ix', 2.1385, 'Percentage', 30); % kg m^2
Iy = ureal('Iy', 2.1385, 'Percentage', 30); % kg m^2
Iz = ureal('Iz', 3.7479, 'Percentage', 30); % kg m^2
tau = ureal('tau', tau_nom, 'Percentage', 30);
gamma = ureal('gamma', gamma_nom, 'Percentage', 30);
% operating point
U1_op = m_nom*g;
psi_op = 0;
phi_op = 0;
th_op = 0;
x_op = [0; 0; 0; 0; 0; 0; phi_op; th_op; psi_op; 0; 0; 0];
u_op = [U1_op; 0; 0; 0];

% linearized dynamics
A = umat(zeros(12, 12));
A(1:3, 4:6) = eye(3);
A(4:5, 7:8) = [sin(psi_op)*U1_op/m, cos(psi_op)*U1_op/m;...
               -cos(psi_op)*U1_op/m, sin(psi_op)*U1_op/m];
A(7:9, 10:12) = eye(3);
B = umat(zeros(12, 4));
B(4, 1) = (phi_op*sin(psi_op) + th_op*cos(psi_op))/m;
B(5, 1) = (-phi_op*cos(psi_op) + th_op*sin(psi_op))/m;
B(6, 1) = 1/m;
B(10:12, 2:4) = diag([lx/Ix; ly/Iy; 1/Iz]);
C = eye(12);
states = {'dx', 'dy', 'dz', 'dxd', 'dyd', 'dzd', 'dphi', 'dth', 'dpsi', 'dphid', 'dthd', 'dpsid'};
```

```
inputs = {'du1', 'du2', 'du3', 'du4'};
outputs = {'dxo', 'dyo', 'dzo', 'dxdo', 'dydo', 'dzdo', 'dphio', 'dtho', 'dpsio', 'dphido', 'dthdo', 'dpsido'};
P = ss(A, B, C, 0, 'statename', states, 'inputname', inputs, 'outputname', outputs);

% sensor noise
sensor_noise_weights = [0.1, 0.1, 0.1, 0.01, 0.01, 0.01, 0.0873, 0.0873, 0.0873, 0.00873, 0.00873, 0.00873];
Wn = ss(diag(sensor_noise_weights));

% actuator perf
Wc = ss(diag([0.4, 1, 1, 10]));

% actuator lag and time delay
Wact_del = append(tf(1, [tau, 1])*tf([-gamma, 1],[gamma, 1]),...
    tf(1, [tau, 1])*tf([-gamma, 1],[gamma, 1]),...
    tf(1, [tau, 1])*tf([-gamma, 1],[gamma, 1]),...
    tf(1, [tau, 1])*tf([-gamma, 1],[gamma, 1]));

% external input disturbance
input_disturbance = [2, 0.4, 0.4, 0.05];
Wdist = ss(diag(input_disturbance));

% label block I/O's
Wn.u = 'noise'; Wn.y = 'Wn';
Wc.u = {'u1', 'u2', 'u3', 'u4'}; Wc.y = 'z_act';
Wdist.u = 'dist'; Wdist.y = 'Wdist';
Wact_del.u = {'u1', 'u2', 'u3', 'u4'}; Wact_del.y = {'u11', 'u22', 'u33', 'u44'};

% specify summing junctions
Sum1 = sumblk('%u_dist = %u + Wdist', {'du1', 'du2', 'du3', 'du4'}, {'u11', 'u22', 'u33', 'u44'});
Sum2 = sumblk('ymeas = %y + Wn', {'dxo', 'dyo', 'dzo', 'dxdo', 'dydo', 'dzdo', 'dphio', 'dtho', 'dpsio', 'dphido', 'dthdo', 'dpsido'});
Sum3 = sumblk('err = ymeas - dx_des', 12);

% connect everything
% M = connect(P, Wn, Wc, Wdist, Sum1, Sum2, Sum3,...
%     {'dist', 'noise', 'dx_des', 'u1', 'u2', 'u3', 'u4'}, ...
%     {'z_act', 'err1', 'err2', 'err3', 'err4', 'err5', 'err6', 'err7', 'err8', 'err9', 'err10', 'err11', 'err12'});
```

```matlab
M = connect(P, Wn, Wc, Wdist, Wact_del, Sum1, Sum2, Sum3,...
    {'dist', 'noise', 'dx_des', 'u1', 'u2', 'u3', 'u4'}, ...
    {'z_act', 'err', 'ymeas'});
% M = connect(P, Wn, Wc, Wdist, Sum1, Sum2,...
%     {'dist', 'noise', 'u1', 'u2', 'u3', 'u4'}, ...
%     {'z_act', 'dxm', 'dym', 'dzm', 'dxdm', 'dydm', 'dzdm', 'dphim', 'dthm', 'dpsim', 'dphidm', 'dthdm', 'dpsidm'});


% D-K iteration
disp('Start D-K');
opts = dksynOptions('MixedMU','on');
[k,clp,bnd,dkinfo] = dksyn(M,12,4,opts);
disp('End D-K');

save('QuadrotorModelCLXPro/K.mat', 'k', 'clp', 'bnd', 'Sum1', 'Sum2', 'Sum3', 'P', 'Wn', 'Wc', 'Wdist', 'x_op', 'u_op', 'A', 'B', 'C', 'm', 'Ix', 'Iy', 'Iz', 'g', 'm_nom', 'sensor_noise_weights', 'input_disturbance', 'l_nom', 'lx', 'ly', 'tau', 'gamma', 'Wact_del', 'dkinfo');

%% Plot

% state = [x y z xd yd zd phi th psi phid thd psid]
% control = [u1 u2 u3 u4];
clear; close all; clc;

load('QuadrotorModelCLXPro/K.mat');


k.InputName = {'err'};
k.OutputName = {'u1', 'u2', 'u3', 'u4'};
% lsim simulate
t = 0:0.1:100;
num = 30;

if num ~= 0
    Parray = usample(P, num);
end
Pnom = P.nom;
yks = zeros(length(t), 16, num+1);
x_des = [0.5; 0.5; 0.5; 0; 0; 0; 0; 0; pi/12; 0; 0; 0]';
for i = 1:num
    cl = connect(Parray(:, :, i), Wn, Wdist, Wact_del, k, Sum1, Sum2, Sum3, {'dist', 'noise', 'dx_des'}, {'dxo', 'dyo', '
```

```
dzo', 'dxdo', 'dydo', 'dzdo', 'dphio', 'dtho', 'dpsio', 'dphido', 'dthdo', 'dpsido', 'u1', 'u2', 'u3', 'u4'});
    U = normrnd(0, 1, length(t), 16)./10;
    U = [U, repmat(x_des, length(t), 1)];
    % shift by operating point
    yks(:, :, i) = lsim(cl, U, t) + [x_op', u_op'];
end
% nominal model and no noise
cl = connect(Pnom, Wn, Wdist, Wact_del, k, Sum1, Sum2, Sum3, {'dist', 'noise', 'dx_des'}, {'dxo', 'dyo', 'dzo', 'dxdo', 'dydo', 'dzdo', 'dphio', 'dtho', 'dpsio', 'dphido', 'dthdo', 'dpsido', 'u1', 'u2', 'u3', 'u4'});
U = normrnd(0, 1, length(t), 16).*0;
U = [U, repmat(x_des, length(t), 1)];
yks(:, :, end) = lsim(cl, U, t) + [x_op', u_op'];


% plot 3D
figure; hold on;
for i = 1:num
    samp = plot3(yks(:, 1, i), yks(:, 2, i), yks(:, 3, i), '-', 'Color', [0    0.4470    0.7410], 'LineWidth', 1);
end
nom = plot3(yks(:, 1, end), yks(:, 2, end), yks(:, 3, end), '-', 'Color', [0.8500    0.3250    0.0980], 'LineWidth', 3);
grid on; box on; axis equal;
xlabel('$$X\ [m]$$', 'interpreter', 'latex');
ylabel('$$Y\ [m]$$', 'interpreter', 'latex');
zlabel('$$Z\ [m]$$', 'interpreter', 'latex');
if num ~= 0
    legend([nom, samp], 'Nominal', 'Monte Carlo Samples');
else
    legend(nom, 'Nominal');
end

% plot control inputs
choose = 1;
figure;
subplot(4,1,1);
plot(t, yks(:, 13, choose));
ylabel('$$U_1$$', 'interpreter', 'latex');
grid on; box on;
subplot(4,1,2);
plot(t, yks(:, 14, choose));
grid on; box on;
```

```matlab
ylabel('$$U_2$$', 'interpreter', 'latex');
subplot(4,1,3);
plot(t, yks(:, 15, choose));
grid on; box on;
ylabel('$$U_3$$', 'interpreter', 'latex');
subplot(4,1,4);
plot(t, yks(:, 16, choose));
grid on; box on;
ylabel('$$U_4$$', 'interpreter', 'latex');

% plot states
figure;
for choose = 1:num
    subplot(3,2,1); hold on;
    plot(t, yks(:, 1, choose), '-', 'Color', [0    0.4470    0.7410], 'LineWidth', 1);
    plot([t(1), t(end)], [x_des(1), x_des(1)], 'k-');
    grid on; box on;
    ylabel('$$X\ [m]$$', 'interpreter', 'latex');
    subplot(3,2,3); hold on;
    plot(t, yks(:, 2, choose), '-', 'Color', [0    0.4470    0.7410], 'LineWidth', 1);
    plot([t(1), t(end)], [x_des(2), x_des(2)], 'k-');
    grid on; box on;
    ylabel('$$Y\ [m]$$', 'interpreter', 'latex');
    subplot(3,2,5); hold on;
    plot(t, yks(:, 3, choose), '-', 'Color', [0    0.4470    0.7410], 'LineWidth', 1);
    plot([t(1), t(end)], [x_des(3), x_des(3)], 'k-');
    grid on; box on;
    ylabel('$$Z\ [m]$$', 'interpreter', 'latex');
    xlabel('$$Time\ [s]$$', 'interpreter', 'latex');

    subplot(3,2,2); hold on;
    sample = plot(t, rad2deg(yks(:, 7, choose)), '-', 'Color', [0    0.4470    0.7410], 'LineWidth', 1);
    target = plot([t(1), t(end)], rad2deg([x_des(7), x_des(7)]), 'k-');
    grid on; box on;
    ylabel('$$\phi\ [^\circ]$$', 'interpreter', 'latex');
    subplot(3,2,4); hold on;
    plot(t, rad2deg(yks(:, 8, choose)), '-', 'Color', [0    0.4470    0.7410], 'LineWidth', 1);
    plot([t(1), t(end)], rad2deg([x_des(8), x_des(8)]), 'k-');
    grid on; box on;
```

```
    ylabel('$$\theta\ [^\circ]$$', 'interpreter', 'latex');
    subplot(3,2,6); hold on;
    plot(t, rad2deg(yks(:, 9, choose)), '-', 'Color', [0 0.4470 0.7410], 'LineWidth', 1);
    plot([t(1), t(end)], rad2deg([x_des(9), x_des(9)]), 'k-');
    grid on; box on;
    ylabel('$$\psi\ [^\circ]$$', 'interpreter', 'latex');
    xlabel('$$Time\ [s]$$', 'interpreter', 'latex');
end
% plot nominal
subplot(3,2,1); hold on;
plot(t, yks(:, 1, end), '-', 'Color', [0.8500 0.3250 0.0980], 'LineWidth', 3);
plot([t(1), t(end)], [x_des(1), x_des(1)], 'k-');
grid on; box on;
    ylabel('$$X\ [m]$$', 'interpreter', 'latex');

subplot(3,2,3); hold on;
plot(t, yks(:, 2, end), '-', 'Color', [0.8500 0.3250 0.0980], 'LineWidth', 3);
plot([t(1), t(end)], [x_des(2), x_des(2)], 'k-');
grid on; box on;
    ylabel('$$Y\ [m]$$', 'interpreter', 'latex');

subplot(3,2,5); hold on;
plot(t, yks(:, 3, end), '-', 'Color', [0.8500 0.3250 0.0980], 'LineWidth', 3);
plot([t(1), t(end)], [x_des(3), x_des(3)], 'k-');
grid on; box on;
    ylabel('$$Z\ [m]$$', 'interpreter', 'latex');
    xlabel('$$Time\ [s]$$', 'interpreter', 'latex');

subplot(3,2,2); hold on;
nom = plot(t, rad2deg(yks(:, 7, end)), '-', 'Color', [0.8500 0.3250 0.0980], 'LineWidth', 3);
target = plot([t(1), t(end)], rad2deg([x_des(7), x_des(7)]), 'k-');
if num ~= 0
    legend([nom, sample, target], 'Nominal', 'Monte Carlo Samples', 'Target');
else
    legend([nom, target], 'Nominal', 'Target');
end
grid on; box on;
    ylabel('$$\phi\ [^\circ]$$', 'interpreter', 'latex');
```

```
subplot(3,2,4); hold on;
plot(t, rad2deg(yks(:, 8, end)), '-', 'Color', [0.8500 0.3250 0.0980], 'LineWidth', 3);
target = plot([t(1), t(end)], rad2deg([x_des(8), x_des(8)]), 'k-');
grid on; box on;
    ylabel('$$\theta\ [^\circ]$$', 'interpreter', 'latex');

subplot(3,2,6); hold on;
plot(t, rad2deg(yks(:, 9, end)), '-', 'Color', [0.8500 0.3250 0.0980], 'LineWidth', 3);
target = plot([t(1), t(end)], rad2deg([x_des(9), x_des(9)]), 'k-');
grid on; box on;
    ylabel('$$\psi\ [^\circ]$$', 'interpreter', 'latex');
    xlabel('$$Time\ [s]$$', 'interpreter', 'latex');

% plot bounds
figure;
opts = bodeoptions('cstprefs');
opts.Grid = 'on';
opts.MagUnits = 'abs';
bodemag(dkinfo{1}.MussvBnds(1,1),opts);
grid on; box on;
title('Structured Singular Value');
```

```
clear; close all; clc;

choice = 'sample'; % 'nom' for nominal; 'sample' for samplied
controller = 0; % 0 for mu-synthesis; 1 for LQR
num = 20;
factor = 0.35;

% desired state
% state = [x y z xd yd zd phi th psi phid thd psid]
x_des = [1; 1; 1; 0; 0; 0; 0; 0; pi/12; 0; 0; 0];

for i = 1:num
    i
    % load program parameters
    initial_conditions
    LoadQuadrotorConst_XPro1a

    % simulink model
    sim('CL_Xpro_model');

    % plot data
    figure(1);
    subplot(3,2,1); hold on;
    plot(x.Time, x.Data,'-', 'Color', [0    0.4470    0.7410]);
    plot([x.Time(1), x.Time(end)], [x_des(1), x_des(1)], 'k-');
    grid on; box on;
    ylabel('X [m]');
    subplot(3,2,3); hold on;
    plot(y.Time, y.Data,'-', 'Color', [0    0.4470    0.7410]);
    plot([y.Time(1), y.Time(end)], [x_des(2), x_des(2)], 'k-');
    grid on; box on;
    ylabel('Y [m]');
    subplot(3,2,5); hold on;
    plot(z.Time, z.Data,'-', 'Color', [0    0.4470    0.7410]);
    plot([z.Time(1), z.Time(end)], [x_des(3), x_des(3)], 'k-');
    grid on; box on;
    ylabel('Z [m]');
    xlabel('Time [s]');

    subplot(3,2,2); hold on;
    plot(Phi.Time, Phi.Data,'-', 'Color', [0    0.4470
       0.7410]);
    plot([Phi.Time(1), Phi.Time(end)], [x_des(7), x_des(7)], 'k-')
       ;
    grid on; box on;
```

```matlab
    ylabel('\phi [rad]');
    subplot(3,2,4); hold on;
    plot(Theta.Time, Theta.Data,'-', 'Color', [0    0.4470    0.7410]);
    plot([Theta.Time(1), Theta.Time(end)], [x_des(8), x_des(8)], 'k-');
    grid on; box on;
    ylabel('\theta [rad]');
    subplot(3,2,6); hold on;
    plot(Psi.Time, Psi.Data,'-', 'Color', [0    0.4470    0.7410]);
    plot([Psi.Time(1), Psi.Time(end)], [x_des(9), x_des(9)], 'k-');
    grid on; box on;
    ylabel('\Psi [rad]');
    xlabel('Time [s]');

    figure(2); hold on;
    plot3(x.Data, y.Data, z.Data,'-', 'Color', [0    0.4470    0.7410]);
    grid on; box on; axis equal;
    xlabel('X m');
    ylabel('Y m');
    zlabel('Z m');

    figure(3);
    subplot(4,1,1);
    plot(U.Time, U.Data(:, 1))
    grid on; box on;
    ylabel('U_1');

    subplot(4,1,2);
    plot(U.Time, U.Data(:, 2))
    grid on; box on;
    ylabel('U_2');

    subplot(4,1,3);
    plot(U.Time, U.Data(:, 3))
    grid on; box on;
    ylabel('U_3');

    subplot(4,1,4);
    plot(U.Time, U.Data(:, 4))
    grid on; box on;
    ylabel('U_4');
```

```
end
```

```
% ------------------------------------------------------------------------------------------
% ------------------------------ Initial conditions ------------------------------
% ------------------------------------------------------------------------------------------

% -------------------- WP: [0 0 0] & [0 0 30] --------------
% ------------------------------------------------------------

load('K.mat');


V1_WP=9.1275;                    % WP - hovering
V2_WP=8.955;
V3_WP=8.61;
V4_WP=8.543;

d_omega_R10=9.1275;               % motor 1. rotor
d_omega_R20=8.955;                % motor 2. rotor
d_omega_R30=8.61;                 % motor 3. rotor
d_omega_R40=8.543;                % motor 4. rotor


dfi_0=-0;                         % roll rate
fi_0=0;                          % roll

dtheta_0=0;                      % pitch rate
theta_0=0;                       % pitch

dpsi_0=0;                        % yaw rate
psi_0=0;                         % yaw

dx_0=0;                          % linear velocity in x - axis
x_0=0;                           % position in x - axis

dy_0=0;                          % linear velocity in y - axis
y_0=0;                           % position in y - axis

dz_0=0;                          % linear velocity in z - axis
z_0=0;                          % position in z - axis

% state = [x y z xd yd zd phi th psi phid thd psid]
```

```
% dx0 = [x_0; y_0; z_0; dx_0; dy_0; dz_0; fi_0; theta_0; psi_0;
    dfi_0; dtheta_0; dpsi_0] - x_op;
state0 = [x_0; y_0; z_0; dx_0; dy_0; dz_0; fi_0; theta_0; psi_0;
    dfi_0; dtheta_0; dpsi_0];

total_time = 50;            % total time of simulation
```

```
% LoadQuadrotorConsts_XPro1a.m
%
% Load electro-mechanical constants for X-Pro specific parameters
% into general quadrotor model:  EPA_Quadrotor_v2b.mdl
%

load('K.mat');

ga = g;                     % grav. accel. (m/s^2)

if strcmp(choice, 'nom')
    m = m_nom;
    L = l_nom;
    Ixx = Ix.nom;
    Iyy = Iy.nom;
    Izz = Iz.nom;
    Tau = tau.nom;                  % Motor/Rotor time const (s)
    Gamma = gamma.nom;
elseif strcmp(choice, 'sample')
    m = m_nom;                  % total mass (kg)
    L = usample(lx);                 % moment arm CG to rotor (m)
    Ixx = usample(Ix);        % Phi (x-axis) Inertia (kgm^2)
    Iyy = usample(Iy);               % Theta (y-axis) Inertia (kgm
        ^2)
    Izz = usample(Iz);        % Psi (z-axis) Inertia (kgm^2)
    Tau = usample(tau);              % Motor/Rotor time const (s)
    Gamma = usample(gamma);
end
Jr=4.86851*0.1;             % rotor inertia (kgm^2)
% Rotor Speed -to- Lift & Drag Functions:
% Lift (N) = c*(r/s)^2+d*(r/s)+e
c = 0.0002;                 % Rotor Speed (r/s)-to-Lift (N), 1st const
d = 0.0071;                 % Rotor Speed (r/s)-to-Lift (N), 2nd const
e = -0.2625;                % Rotor Speed (r/s)-to-Lift (N), 3rd const
% Drag (Nm) = f*(r/s)^2+g*(r/s)+h
f = 5e-6;                   % Rotor Speed (r/s)-to-Drag (Nm), 1st
    const
g = 0.0008;                 % Rotor Speed (r/s)-to-Drag (Nm), 2nd
    const
h = -0.0282;                % Rotor Speed (r/s)-to-Drag (Nm), 3rd
    const

% Specific Motor/Rotor Speed Constants: (r/s) = a*(Volts)+b
a1 = 16.235;                % Motor 1 Volts-to-Rotor Speed (r/s), 1st
    const
```

```
b1 = -0.5036;            % Motor 1 Volts-to-Rotor Speed (r/s), 2nd const
a2 = 17.912;             % Motor 2 Volts-to-Rotor Speed (r/s), 1st const
b2 = -12.728;            % Motor 2 Volts-to-Rotor Speed (r/s), 2nd const
a3 = 17.914;             % Motor 3 Volts-to-Rotor Speed (r/s), 1st const
b3 = -6.5646;            % Motor 3 Volts-to-Rotor Speed (r/s), 2nd const
a4 = 18.161;             % Motor 4 Volts-to-Rotor Speed (r/s), 1st const
b4 = -7.4637;            % Motor 4 Volts-to-Rotor Speed (r/s), 2nd const

co_x=1.7*0;
co_y=1.7*0;
co_z=90.7*0;

c_mi_x=170*0;
c_mi_y=170*0;
c_mi_z=40*0;

% Toy LQR
[K_lqr,S,E] = lqr(A_nom,B_nom,eye(12),eye(4).*100);

% get angular velocity
b = 1;% thrust coefficient
d = 0.5;% drag coefficient
Jac = [b, b, b, b;...
      0, -b, 0, b;...
     -b, 0, b, 0;...
     -d, d, -d, d];
invJac = inv(Jac);
```

Simulink Models

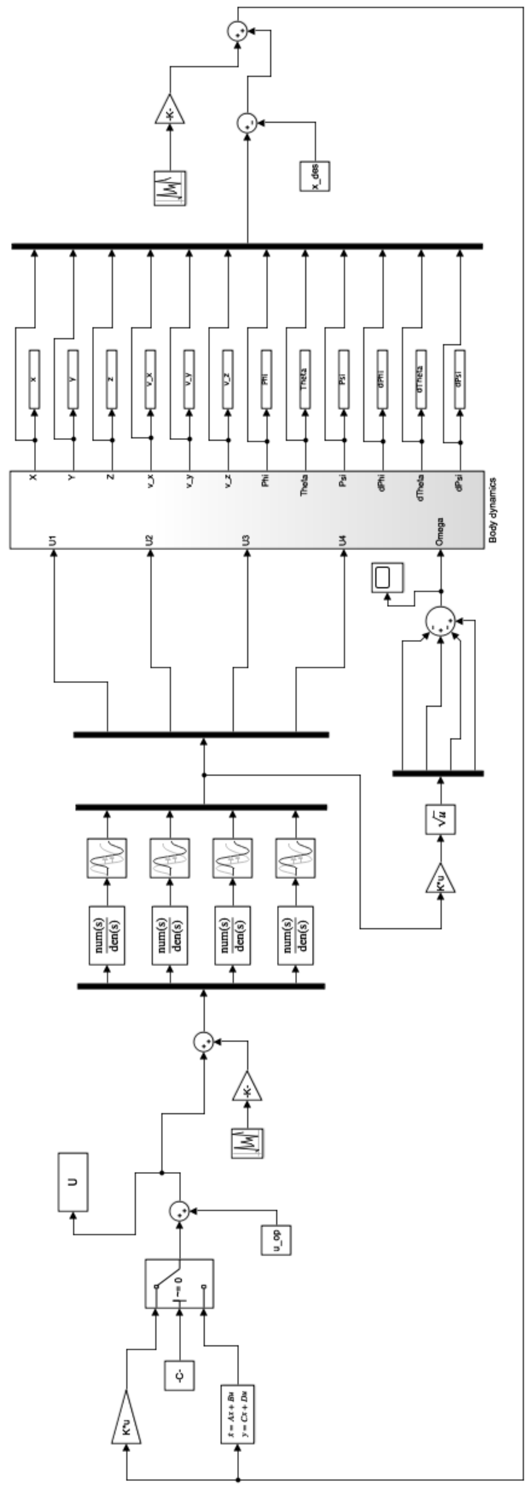


Figure 11: Full Simulink

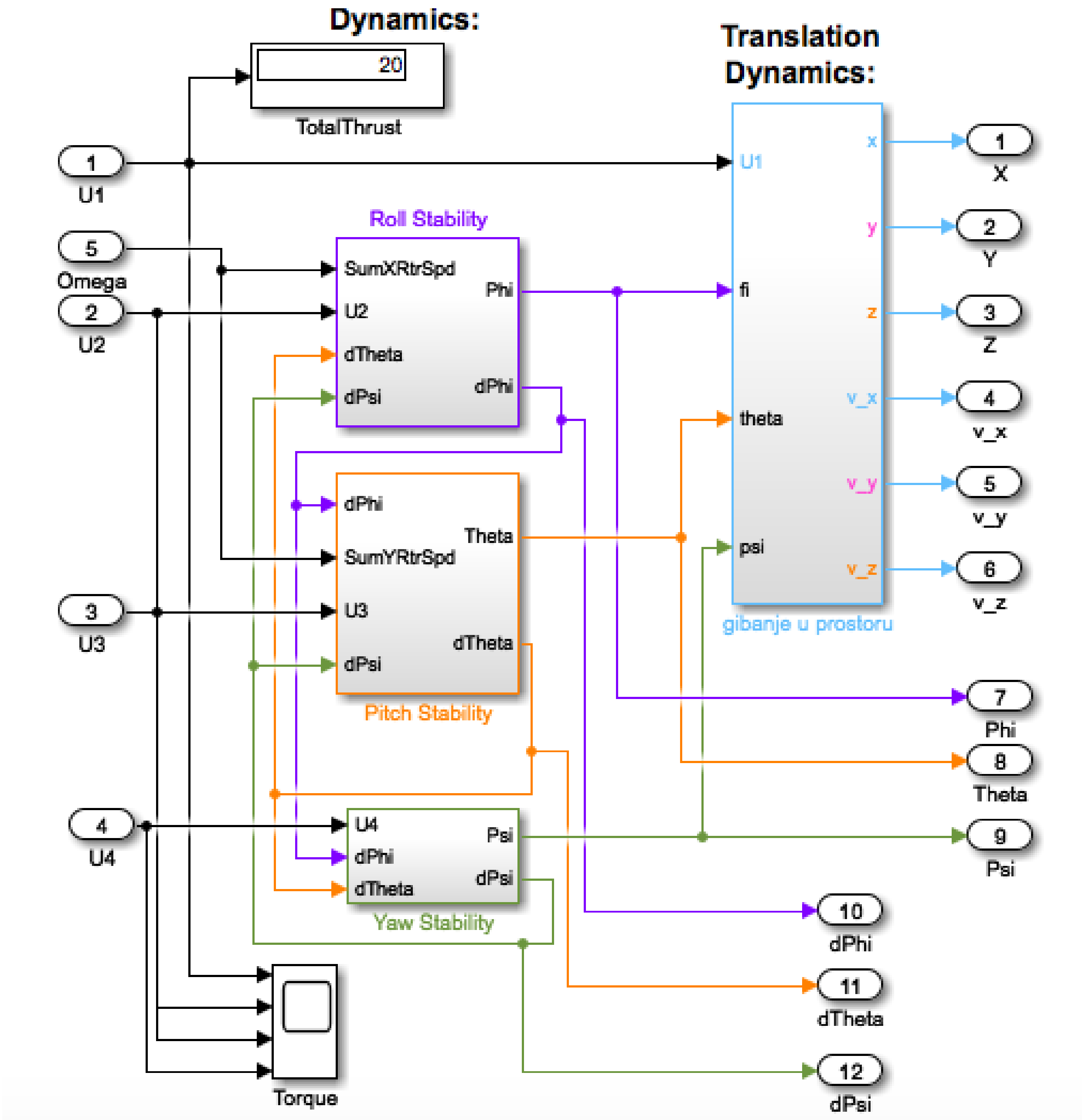


Figure 12: Simulink Dynamics

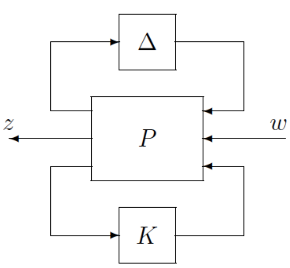
Simplified, Linearized Relative Dynamics

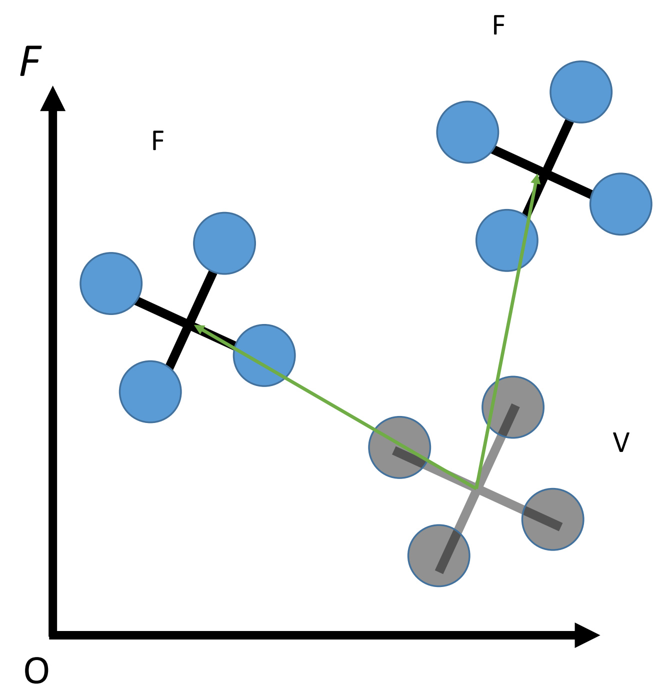
$$\delta\dot{x} = A\delta x + Bu$$

$$\delta x = [\delta\rho_1 \quad \cdots \quad \delta\rho_n]$$
$$= position\ vectors\ from\ follower\ to\ virtual\ leader$$

$$u = [F_1 \quad \cdots \quad F_n]$$
$$= generalized\ forces\ to\ maintain\ formation$$

Lump follower nonlinear dynamics, sensor noises, and disturbances into structured uncertainty matrix. Transform state space form using Linear Fractional Transformation





Modeling Uncertainties

- Sensors:
  - 3-axis Gyroscope:
    - $\widetilde{\omega} = \omega + \beta + \eta_v,\ \eta_v \sim (0, \sigma_v^2)$
    - $\dot{\beta} = \eta_u, \eta_u \sim (0, \sigma_u^2)$
  - GPS:
    - $\begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \eta_x \\ \eta_y \end{bmatrix}, \eta_x \sim (0, \sigma_x^2), \eta_y \sim \left(0, \sigma_y^2\right)$
  - Barometer:
    - $\tilde{h} = h + \eta_h, \eta_h \sim \left(0, \sigma_h^2\right)$
  - 3-axis Accelerometer:
    - Sense direction of local gravity vector

- Extended Kalman Filter for state estimation

- Disturbances
  - Wind -> random forces and torques on quadcopter

- Plant uncertainties
  - Inertias, motor constants

Control Strategy

- **Control Objective:** Maintain robust performance of quadcopter fleet formation controller under environmental disturbances, sensor noise, and plant uncertainties while following reference trajectories.

- Linear Quadratic Gaussian (LQG), backstepping controller for virtual leader quadcopter reference tracking. Open-loop control for follower.

- μ-synthesis controller calculated from D-K iteration for formation control to compensate system uncertainties (use MATLAB Robust Control Toolbox).
- "Generalized forces" will be translated into actual motor inputs