Quad-rotor P&S

Exercise 2: Control Architecture and Tuning

Learning objectives relevant for this exercise sheet:

iv) Be able to explain why a quad-rotor design allows the control architecture to be de-coupled into a collection of separate simple controllers,

v) Experienced the challenges of tuning a PID and LQR controller for achieving stable hover of a quad-rotor in simulation,

The following steps are a guideline for converting the template Simulink model provided from an Linear Quadratic Regulator (LQR) controller with a hard-coded state-feedback matrix to a combined PID and LQR controller that is specific to your N-rotor design.

Pre-work Task: Linearisation about hover

Derive the linearisation of the continuous time non-linear equations of motion about the hover equilibrium. Section 4.2 of the script describes the steps required and provides a brief review of linearisation. Specifically the pre-work exercise is to derive the linear system matrices given in equation (4.8) of the script, and come to class prepared to discuss points (i)–(iii) below.

The derivation for linearising the equations of motion can be split up into 20 separate blocks, of which 13 blocks are identically zero due to there being no dependency of the expression on the variable of derivation, and 2 blocks are 3-by-3 identity matrices. Thus the derivation reduces to the following blocks:

$$\begin{bmatrix} \Delta\dot{\vec{p}} \\ \Delta\ddot{\vec{p}} \\ \Delta\dot{\vec{\psi}} \\ \Delta\ddot{\vec{\psi}} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & I_{3\times3} & 0 & 0 \\ 0 & 0 & \frac{d}{d\vec{\psi}}\ddot{\vec{p}} & 0 \\ 0 & 0 & 0 & I_{3\times3} \\ 0 & 0 & \frac{d}{d\vec{\psi}}\ddot{\vec{\psi}} & \frac{d}{d\vec{\psi}}\ddot{\vec{\psi}} \end{bmatrix}}_{A} \begin{bmatrix} \Delta\vec{p} \\ \Delta\dot{\vec{p}} \\ \Delta\vec{\psi} \\ \Delta\dot{\vec{\psi}} \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ \frac{d}{d\vec{f}}\ddot{\vec{p}} \\ 0 \\ \frac{d}{d\vec{f}}\ddot{\vec{\psi}} \end{bmatrix}}_{B} \Delta\vec{f}.$$

where we have introduced the same notation as from the script, i.e.,

- $\Delta\vec{p}$ indicates a deviation of $\vec{p}$ from its equilibrium value,
- $\vec{f} = [f_1, \ldots, f_N]^\top$ is the column vector of the individual rotor thrust forces f_i stacked,
- The size of each block in the A matrix is 3-by-3, and size 3-by-N in the B matrix.

Based on your derivation, answer the following:

(i) What happens to the $\frac{d}{d\vec{\psi}}\ddot{\vec{p}}$ block if we linearise about a non-zero yaw for the equilibrium point, denoted α_{hover}?

(ii) Provide an intuitive explanation for why the blocks $\frac{d}{d\vec{\psi}}\ddot{\vec{\psi}}$ and $\frac{d}{d\vec{\psi}}\ddot{\vec{\psi}}$ are identically zero despite the non-trivial dependencies.

(iii) The sparse structure of the continuous-time linear system matrices, i.e., A and B, can be separated into smaller independent systems. What is the state vector for each of these independent systems?

In-class Tasks

A) Rotation of error (approximately) into body frame

Open the file "`exercise02_get_reference_specification.m`" and provide a 20 degree step change for yaw by changing the variable named "`reference_step_xyz_yaw`". Next, provide a 90 degree step change for yaw. How does the (x, y) tracking performance differ?

The goal of this task is to derive and implement a simple rotation of the controller errors into the body frame and remedy the (x, y) tracking performance.

- The nested controller architecture will be explained and discussed in class.
- Assuming the roll and pitch angles are zero, derive the transformation of the (p_x, p_y) and $(\dot{p}_x, \dot{p}_y)$ errors into the body frame.
- Implement this transformation in the "`outer control loop`" sub-system of your Simulink model in the MATLAB Function block named "`rotate_xy_inertial_to_body`".
- Does this transformation sufficiently remedy the (x, y) tracking performance? For all types of references? For any controller frequency?
- Is the assumption of zero roll and pitch angle reasonable? Is there a need for, and/or benefit from, a transformation to the p_z and $\dot{p}_z$ error also?

B) PID Control for altitude and yaw

So far the outer loop controller has been stabilising the system using a design that is based on the Linear Quadratic Regulator (LQR) method.

The goal of this task is to replace the parts of the LQR contoller in the outer loop that control the altitude and yaw.

- An example Simulink PID controller implementation is shown in Figure 1.
- Start with the altitude controller, and once you having completed tuning, then move onto the yaw controller.
- Material will be handed out in class with "rules-of-thumb" for tuning the gains of a PID controller using the Ziegler Nichols method.
- If actuator saturation occurs, this could result in misleadingly high values of the ultimate gain K_u needed for the Ziegler Nichols method. In determining K_u, try to find a gain that leads to sustained oscillations without saturating the actuators.
- Can you use the gains of the Linear Quadratic Regulator (LQR) controller included in the template Simulink model to provide reasonable starting gains for your PID controller, from which you can then perform some fine-tuning?

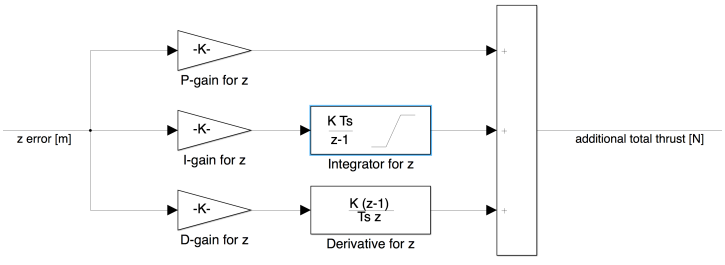

Figure 1: Showing a example of how a discrete time PID controller can be implemented in Simulink. The blocks used are a separate `Gain` block for each of the proportional, integral, and derivative gains, a `Discrete-Time Integrator` block for integrating the error signal, and a `Discrete Derivative` block for computing the finite-difference derivative of the error signal.

C) LQR control for (x, y) position

Use the linear system matrices to design an infinte-horizon LQR controller state-feedback matrix and implement this in the template Simulink model provided.

- You will need to choose the Q and R **cost function** matrices to achieve the desired flight performance of your N-rotor vehicle. The infinite-horizon LQR cost function is:

$$J_\infty = \int_0^\infty \Big(x(t)^\top Q\, x(t) + u(t)^\top R\, u(t) \Big)\, dt$$

- For choosing the Q and R cost function matrices you should think about the units of the different elements of the state vector. For example, perhaps you wish to penalise a 10 centimeter deviation in x position the same amount as a 5 degree deviation in yaw.
- Use the MATLAB functions `care` and `dare` to compute solutions for the continuous and discrete time algebraic Riccati equation respectively. Read the help documentation to be sure you understand what these functions compute.
- The infinite-horizon LQR design equations for a **continuous-time** linear-time-invariant (LTI) system are (taken directly from the Control Systems I lecture notes):

$$0 = -A^\top P_\infty - P_\infty A + P_\infty B R^{-1} B^\top P_\infty - Q \tag{1a}$$
$$u(t) = -K_\infty\, x(t) \tag{1b}$$
$$K_\infty = R^{-1} B^\top P_\infty \tag{1c}$$

  where (1a) is the Riccati equation to solve for P_∞, (1b) is the control law to implement, and (1c) is the state-feedback gain matrix.
- The infinite-horizon LQR design equations for a **discrete-time** LTI system are:

$$0 = -P_{\infty,\mathrm{D}} + A_\mathrm{D}^\top P_{\mathrm{D},\infty} B_\mathrm{D} \left(B_\mathrm{D}^\top P_{\mathrm{D},\infty} B_\mathrm{D} + R \right) B_\mathrm{D}^\top P_{\mathrm{D},\infty} A_\mathrm{D} + Q \tag{2a}$$
$$u_\mathrm{D}(k) = -K_{\mathrm{D},\infty}\, x_\mathrm{D}(k) \tag{2b}$$
$$K_{\mathrm{D},\infty} = \left(B_\mathrm{D}^\top P_{\mathrm{D},\infty} B_\mathrm{D} + R \right) B_\mathrm{D}^\top P_{\mathrm{D},\infty} A_\mathrm{D} \tag{2c}$$

  where the subscript $(\cdot)_\mathrm{D}$ indicates that the quantities are for a discrete-time system.
- Both the `care` and `dare` function return a state-feedback gain matrix, be careful of the sign convention if you use this matrix.
- To convert the continuous-time system matrices to discrete-time system matrices use the MATLAB function `c2d`, specifying *zero order hold* as the discretisation method.

Additional Tasks

D) Extension: measurement noise

All the tasks above were completed without measurement noise, now include the measurement noise and re-tune the PID and LQR gains as necessary.

Additional info - Feed forward and cascaded control architecture:

It is common to design a controller for a linearisation of a system around a chosen equilibrium point. It is important to remember that the equilibrium point chosen is a combination of:

(i) an equilibrium state of the system, and

(ii) the equilibrium input that keep the system at that equilibrium state.

For an N-rotor vehicle, the inputs to the vehicle are the thrust for each propeller, and therefore a controller designed for a linearised N-rotor vehicle system is a function that:

- should receive the **error** between the current state of the system and the **reference** equilibrium state, and
- returns the **thrust adjustments** that should be made to the **equilibrium thrust** of each propeller.

This equilibrium input is commonly referred to as the feed-forward input, and the feed-forward architecture for an N-rotor vehicle is shown in Figure 2. The control function can therefore be described in words as:

$$\begin{matrix}\text{propeller}\\ \text{thrusts}\end{matrix} = \begin{matrix}\text{equilibrium}\\ \text{thrusts}\end{matrix} + \begin{matrix}\text{thrust}\\ \text{adjustments}\end{matrix} . \tag{3}$$

To control an N-rotor vehicle, it is common practice to use a cascaded control architecture where the control is separated into two nested feedback control loops. The motivation for this cascaded architecture is that the IMU (Inertial Measurement Unit) on-board the vehicle has a gyroscope that provides high-frequency (up to 1000Hz) measurements of the angular rates about the body frame axes of the vehicle, i.e., measurements of $\vec{\omega}$. This is in contrast to the position and attitude measurements that are generally available at a much lower frequency. When using am external motion capture system, the frequency of position and attitude measurements could be as high as 200Hz, however, other types of sensors may be limited to frequencies of less than 50Hz.

Based on these two separate measurement streams, the cascaded control architecture is shown in Figures 3, 4, 5, and can be described in words as:

- the **"inner controller"** uses the body rate measurements, and is designed to regulate deviations from the body rates reference provided by the "outer controller".
- the **"outer controller"** uses the measurements of (x, y, z) position in the inertial frame, and the yaw angle, and is designed to regulate deviations from the $(x^{(\mathrm{I})}, y^{(\mathrm{I})}, z^{(\mathrm{I})}, \alpha)$ reference that is provided by some external oracle.

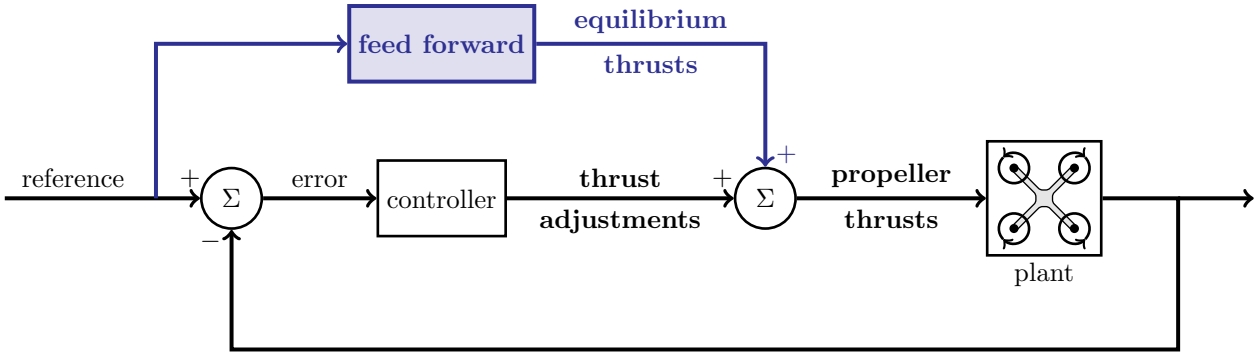

Figure 2: Schematic of the feed-forward architecture for an N-rotor vehicle.

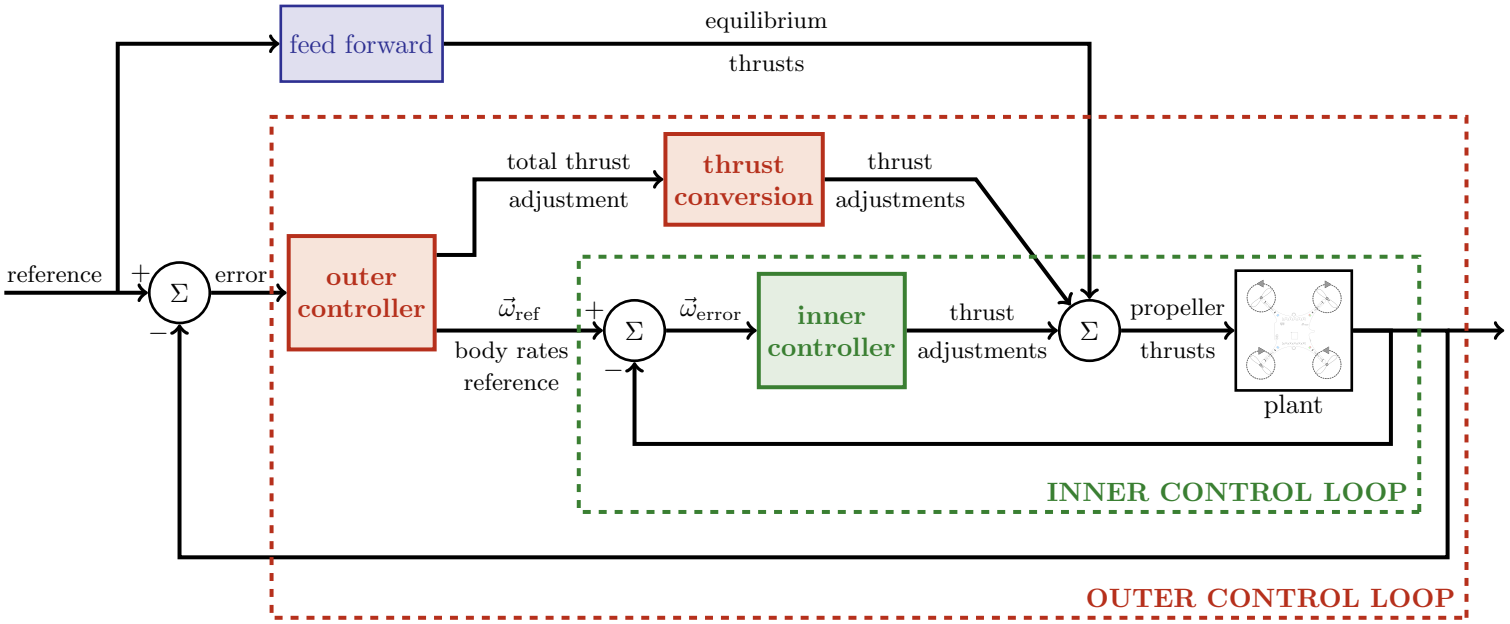

Figure 3: Schematic of the cascaded architecture for an N-rotor vehicle. The "thrust conversion" block is needed because the "total thrust adjustment" from the "outer controller" is a scalar quantity, and it must be converted to a vector of N "thrust adjustments" that together provide the "total thrust adjustment" requested and apply zero net torque about the body frame axes.

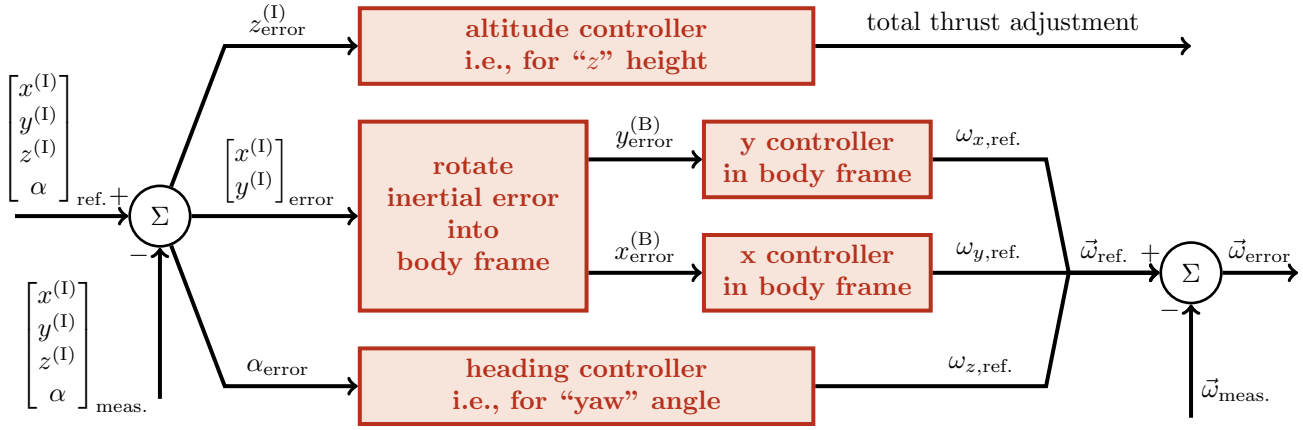

Figure 4: Schematic of the outer controller architecture for an N-rotor vehicle. The four controllers are essentially independent. The altitude controller regulates deviations in the height on the vehicle, while the heading controller regulates deviates in the heading direction. The (x, y) horizontal position controllers are slightly coupled through the "rotate interial error into body frame" block. This block is required because the (x, y) controllers use body frame actuation but the measurements are inertial frame positions.

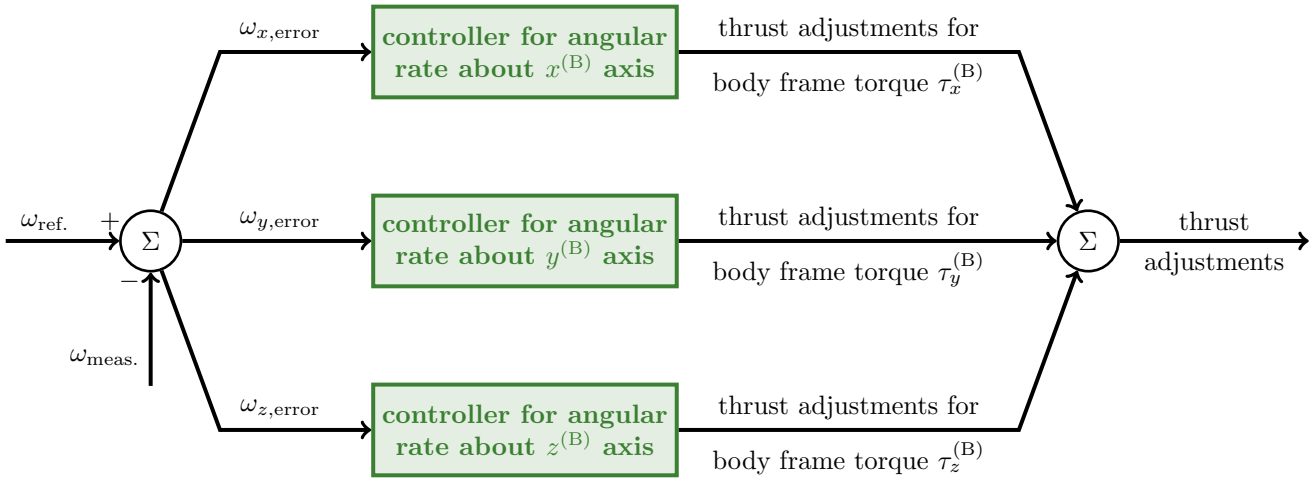

Figure 5: Schematic of the inner controller architecture for an N-rotor vehicle. The three controllers independently regulate angular rates deviations about each of the body frame axes.

Additional info - linearised equations of motion:

The following are the state-space representation of the linearised equations of motion for the "inner" and "outer" loop. The inner loop has the body rates $\vec{\omega}$ as the state and actuator torques $(\tau_x^{(B)}, \tau_y^{(B)}, \tau_z^{(B)})$ as the inputs. The outer loop has the position $\vec{p}$, velocity $\dot{\vec{p}}$, and intrinsic Euler angles $\vec{\psi}$ as the state and the total thrust f_{total} and body rates $\vec{\omega}$ as the inputs. The assumption of splitting the equations of motion into these two loops for controller design is that we have a sufficient time scale separation between the dynamics of the two loops. This assumption means that the difference between when the outer loop requests a desired body rate $\vec{\omega}$ from the inner loop, and the time when the inner loop tracks this request is negligible relative to the time scale of the outer loop dynamics. In more colloquial terms, the outer loop assumes that the inner loop tracks its requests infinitely fast. One method of checking whether this separation exists is to check that the slowest eigenvalue of the inner loop is 5-10 times faster than the fastest eigenvalue of the outer loop.

The linearised equations of motion, linearised about the hover equilibrium state, for the outer loop are:

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \\ \\ \ddot{p}_x \\ \ddot{p}_y \\ \ddot{p}_z \\ \\ \dot{\gamma} \\ \dot{\beta} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & g & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ \\ \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \\ \\ \gamma \\ \beta \\ \alpha \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f_{\text{total}} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix},$$

where the notation is as defined in the course script. We see in this linearised form that the equations of motion relating to the inertial $z^{(\text{I})}$ position are completely decoupled from the other states and the input f_{total} influences only $\ddot{p}_z$. Similarly, those equations relating to the yaw α angle are completely decoupled from the other states and the input ω_z influences only $\dot{\alpha}$. Finally, the equations relating to the inertial $x^{(\text{I})}$ are coupled to the pitch angle β and ω_y influences only $\dot{\beta}$. Similarly, $y^{(\text{I})}$ is coupled to the roll angle γ and ω_x influences only $\dot{\gamma}$. This decoupled structure of the linearised equations of motion matches the intuition used for establishing the controller architecture.

The linearised equations of motion, linearised about the hover equilibrium state, for the inner loop are:

$$\begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} + J^{-1} \begin{bmatrix} \tau_x^{(\text{B})} \\ \tau_y^{(\text{B})} \\ \tau_z^{(\text{B})} \end{bmatrix},$$

where the mass moment of inertia matrix will be approximately diagonal for a symmetric quadrotor and hence J^{-1} is approximately equal to

$$J^{-1} \approx \begin{bmatrix} \frac{1}{J_x} & 0 & 0 \\ 0 & \frac{1}{J_y} & 0 \\ 0 & 0 & \frac{1}{J_y}. \end{bmatrix}$$

This highlights that equations of motions for rotations about the body frame axes are only weakly coupled, and so we can expect to achieve good control performance using a separate, decoupled, controller for each axis.