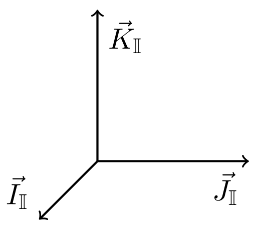
Lab 3: Drone Controllers

The purpose of this lab is familiarize with the controller the drone currently uses. Then you will design and simulate an LQR controller in Simulink and implement it on the drone.

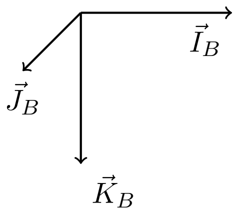
Drone Dynamics

Let us define some notation first:

- $\mathbb{I}$ is the inertial frame and B is the body-fixed frame with origin at the center of mass of the drone.



(a) Inertial frame $\mathbb{I}$



(b) Body-fixed frame B

- $\vec{r}$ is the position of the drone center of mass in the inertial frame

$$\vec{r} = X\vec{I}_{\mathbb{I}} + Y\vec{J}_{\mathbb{I}} + Z\vec{K}_{\mathbb{I}}$$

- $\vec{v}$ is the velocity of the drone center of mass relative to the inertial frame. It expressed in the body-fixed frame is

$$\begin{aligned}\vec{v} =& \dot{X}\vec{I}_{\mathbb{I}} + \dot{Y}\vec{J}_{\mathbb{I}} + \dot{Z}\vec{K}_{\mathbb{I}}\\ \vec{v} =& v_x\vec{I}_B + v_y\vec{J}_B + v_z\vec{K}_B\\ =& [\vec{I}_B\ \vec{J}_B\ \vec{K}_B]\begin{bmatrix} v_x\\ v_y\\ v_z\end{bmatrix}\\ =& [\vec{I}_{\mathbb{I}}\ \vec{J}_{\mathbb{I}}\ \vec{K}_{\mathbb{I}}]Q_{B/\mathbb{I}}\begin{bmatrix} v_x\\ v_y\\ v_z\end{bmatrix}\end{aligned}$$

- E is the Euler angles (yaw, pitch, and roll)

$$E = \begin{bmatrix}\psi\\ \theta\\ \phi\end{bmatrix}$$

describing the body orientation of the drone w.r.t to the inertial frame $\mathbb{I}$. Also $\dot{E} = \begin{bmatrix} \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix}$ is the angular velocity of the drone in the Euler angles.

- $\vec{\omega}_{B/\mathbb{I}}$ is the angular velocity of the body-fixed frame w.r.t the inertial frame. It expressed in the body-fixed frame is

$$\begin{aligned}\vec{\omega}_{B/\mathbb{I}} =& \omega_x \vec{I}_B + \omega_y \vec{J}_B + \omega_z \vec{K}_B \\ =& (\dot{\phi} - \dot{\psi}\sin\theta)\vec{I}_B + (\dot{\psi}\cos\theta\sin\theta + \dot{\theta}\cos\phi)\vec{J}_B + (\dot{\psi}\cos\theta\cos\phi - \dot{\theta}\sin\phi)\vec{K}_B\end{aligned}$$

- W^{-1} transforms the angular velocities $\vec{\omega}_{B/\mathbb{I}}$ to the Euler angle rates $\dot{E}$:

$$\dot{E} = \begin{bmatrix} \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = W^{-1}[\vec{\omega}_{B/\mathbb{I}}]_B$$

where $[\vec{\omega}_{B/\mathbb{I}}]_B = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$ and

$$W^{-1} = \frac{1}{\cos(\theta)} \begin{bmatrix} 0 & \sin(\phi) & \cos(\phi) \\ 0 & \cos(\phi)\cos(\theta) & -\sin(\phi)\cos(\theta) \\ \cos(\theta) & \sin(\phi)\sin(\theta) & \cos(\phi)\sin(\theta) \end{bmatrix}$$

- T is the total vertical thrust generated by the rotors in the body frame. τ_z is the total torque about the body z-axis resulting from propeller drag and angular acceleration, τ_y and τ_x are the resulting torque about the body fixed y-axis and x-axis, respectively. I_B is the drone's inertia matrix expressed in the body-fixed frame and we assume that

$$I_B = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix}.$$

Now, let us define the state input variables:

- The state vector, x, is:

$$x = \begin{bmatrix} X \\ Y \\ Z \\ \psi \\ \theta \\ \phi \\ v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

- The input variables are the vertical force and moments generated by the propellers:

$$u_P = \begin{bmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix}. \tag{1}$$

- Furthermore, we define the vector of external forces acting on the drone expressed in the body-fixed frame as

$$[F_{ext}]_B = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = Q_{\mathbb{I}/B} \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix}$$

  where $Q_{\mathbb{I}/B}$ is the matrix that transforms vectors in the inertial frame to the body-fixed frame, $g = -9.81 m/s^2$ is the acceleration due to gravity, and m is the mass of the drone.

Then the drone equations of motion are:

$$\begin{aligned}
\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} &= Q_{B/\mathbb{I}} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \\
\begin{bmatrix} \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} &= W^{-1} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \\
\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} &= \frac{1}{m} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} - \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \\
I_B \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} &= \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} - \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} I_B \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}
\end{aligned}$$

For notation simplicity we write the drone dynamics as

$$\dot{x} = f_P(x, u_P)$$

The rotation matrices are given by

$$Q_{B/\mathbb{I}} = \begin{bmatrix} \cos(\theta)\cos(\psi) & \cos(\psi)\sin(\theta)\sin(\phi) - \cos(\phi)\sin(\psi) & \sin(\phi)\sin(\psi) + \cos(\phi)\cos(\psi)\sin(\theta) \\ \cos(\theta)\sin(\psi) & \cos(\phi)\cos(\psi) + \sin(\theta)\sin(\phi)\sin(\psi) & \cos(\phi)\sin(\theta)\sin(\psi) - \cos(\psi)\sin(\phi) \\ -\sin(\theta) & \cos(\theta)\sin(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix}$$

and

$$Q_{\mathbb{I}/B} = \begin{bmatrix} \cos(\psi)\cos(\theta) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ \cos(\psi)\sin(\phi)\sin(\theta) - \cos(\phi)\sin(\psi) & \cos(\psi)\cos(\phi) + \sin(\theta)\sin(\phi)\sin(\psi) & \cos(\theta)\sin(\phi) \\ \sin(\psi)\sin(\phi) + \cos(\psi)\cos(\phi)\sin(\theta) & \cos(\phi)\sin(\psi)\sin(\theta) - \cos(\psi)\sin(\phi) & \cos(\phi)\cos(\theta) \end{bmatrix}$$

Note that the inputs into the drone dynamics are u_P (the vertical thrust and torques around the body-frame axis). However, we actually command the desired thrust for each of the four motors:

$$u_M = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix}.$$

The conversion from u_P to the four motor commands u_M is, with our simplified model, a linear transformation described by an invertible matrix M. So we can compute u_M from u_P, or vice versa, from the relation:

$$u_P = M u_M$$

The matrix M can be found in the `parameters_estimationcontrol.m` file in the folder: `MIT_MatlabToolbox/trunk/matlab/Simulation`. In this file it is called `controlHelperParams.Ts2Q`. The inverse M^{-1} is called `controlHelperParams.Q2Ts`

Therefore, depending on the situation we can work with either u_M or u_P as the input and can compute the other. A block diagram of the system is shown in Figure 2. We can also easily calculate the drone dynamics with u_M as the input by substituting $M^{-1}u_P$ for u_M in the dynamics:

$$\dot{x} = f_P(x, u_P) = f_M(x, M^{-1}u_p) = f_M(x, u_M)$$

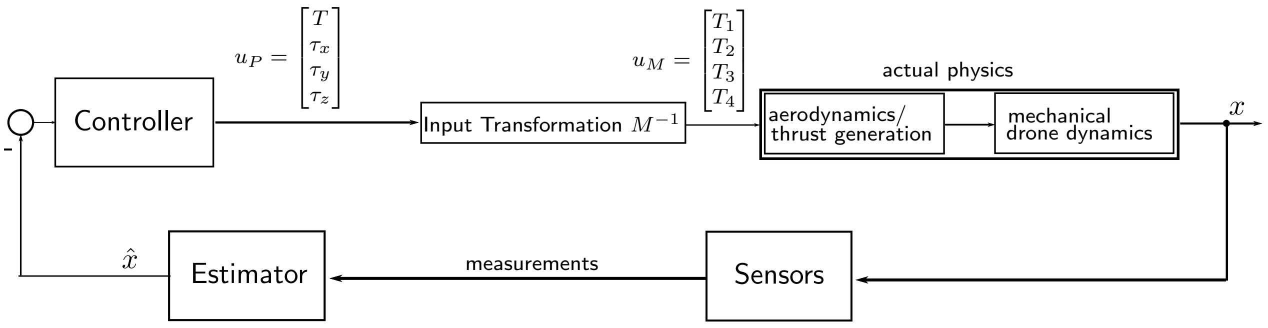

$u_P = \begin{bmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix}$
$u_M = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix}$
actual physics
Controller
Input Transformation M^{-1}
aerodynamics/ thrust generation
mechanical drone dynamics
x
-
$\hat{x}$
Estimator
measurements
Sensors

Figure 2: Practical implementation scheme for drone control

1 Linearization

Important: Navigate to `MIT_MatlabToolbox/trunk/matlab` and run the `startup.m` script before starting this lab to make sure all the necessary files are on the Matlab path.

For the LQR controller we will consider the drone dynamics with u_M as the input. In order to do control design we first want to linearize the drone dynamics about a certain operating point. From the nonlinear dynamics

$$\dot{x} = f_M(x, u_M)$$

we can compute a linearized plant of the form

$$\triangle\dot{x} = A\triangle x + B\triangle u_M$$

about a specified operating point. This will allow you to design a full-state feedback controller using the LQR framework.

1. The file `LinearDroneAndPolePlaceControl.m` in `/trunk/matlab/Simulation/controllers/controller_fullstate/Poleplacement` performs a symbolic linearization with u_P as the input to the drone dynamics. Modify the `LinearDroneAndPolePlaceControl.m` file so that the dynamics take u_M as the input (you need to add the input transformation). Then you can run the first section of this file to symbolically compute the A and B matrices of the linearized model about the operating point specified by the variable `equil`. **Compute and report the A and B matrices you obtain from the linearization.**

2. The drone dynamics in the Simulink model `sim_quadrotor.slx` include a more detailed and complicated model of the drone and propeller aerodynamics. This model is located in the `Drone Dynamics` block and is called `quadrotor_dynamics`.

   We would like to get a linearization of this system in hover and compare it to the linearization found in the previous question. To do this you can use the `linearizeDrone_motorcmdTostate.slx` Simulink model in the `Simulation/controllers/controller_fullstate/LQR` folder. This is just a copy of the Drone Dynamics model with the input fixed so the drone is hovering. Simulink's `Analysis/ControlDesign/Linear Analysis` tool can then be used to find the (A, B, C, D) matrices around hover condition for this complex model. **Report the A and B matrices you obtain.**

   (In Simulink help search for 'Linearizing Nonlinear Models" for more information and to understand how Simulink linearization works. You can also consult `https://www.mathworks.com/help/slcontrol/ug/linearize-simulink-model.html`).

3. Compare the A and B matrices and the eigenvalues of A you obtain from the two previous questions and comment on your results. **Note that the B matrix matrices for the two problems will be significantly different because there is another transform in the Simulink model that maps the desired torques to the motor commands.**

4. What are the advantage/differences of having a linearized model obtained symbolically versus the one obtained by linearizing the Simulink schema?

2 LQR Design

Please make sure to follow all safety procedures before you fly the drone.

For the LQR the cost functional is defined as

$$J = \int_0^\infty x^\top Q x + {u_M}^\top R u_M \, dt$$

where $Q \in \mathbb{R}^{12\times 12}$ and $R \in \mathbb{R}^{4\times 4}$ are the state and input weighting matrices. The goal of LQR is to minimize the cost functional subject to the linearized dynamics $\triangle\dot{x} = A\triangle x + B\triangle u_M$.

The solution to this problem is a linear static controller of the form $u_M = -Kx$ where $K \in \mathbb{R}^{4\times 12}$.

1. Open the file `MIT_MatlabToolbox/trunk/matlab/Simulation/controllers/controller_fullstate/LQR/controller_lqr.slx`. It contains a template simulink block `ControllerLQR` for the LQR full-state feedback controller that you will implement. This block will replace the `ControllerPID` block in `sim_quadrotor.slx`.
2. Please look at the following Matlab file in the toolbox:
   /trunk/matlab/Simulation/controllers/controller_fullstate/LQRControl.m.
   This file provides a template for designing the LQR hover controller, with preset values for the bounds using Bryson's rule and cost weights for state error and control effort.
   Comment on the choice of the preset bounds and weights.
3. Using the linearized dynamics determined from Problem 2, run the original version of the m-file `LQRcontrol.m` (i.e. with the original values for bounds and cost weights). It generates an LQR full state feedback controller matrix `K_lqr_toMotorcmd`.
4. Complete the `ControllerLQR` Simulink block such that it implements the LQR full state feedback controller that uses the matrix `K_lqr_toMotorcmd`. In the `ControllerLQR` block, please navigate to `FullstateController` to implement the controller.
   (Side note: The other blocks are for interface/transformation purposes.)
5. Simulate the drone's behavior with this new controller. You can play with the ReferenceValue-Server block to change the reference.

   In your report, show the following plots and add them with comments to your report:

   - One plot that describes the position of the drone. This plot should show three variables, namely the X, Y, and Z coordinates of the drone, with respect to time, in one figure.
   - One plot that describes the orientation of the drone. This plot should show the three Euler angles, namely roll, pitch, and yaw angles of the drone, with respect to time, in one figure.
   - One plot that shows the four motor commands sent to the drone with respect to time, in one figure.

   In your report, for each of the three plots, briefly discuss whether the data is what you expected and why.
6. Generate embedded C code from the `Drone_Compensator` block with the LQR feedback control system, and compile/upload the embedded C code to your drone by running

   ```
   $ DroneUploadEmbeddedCode.sh
   ```

   Fly your drone with the LQR feedback control system. Once the drone is in stable hover, hit the 'y' key to provide an altitude reference that is 0.6m higher (you can use the 'e' key, if you would like to exit the experiment).
7. Download the data of this experiment from your drone into the Matlab environment.

   Use the FlightAnalyzer.m file to generate the following plots and add them with comments to your report:
   - One plot that shows the time evolution of IMU, altitude, and motor commands. This plot is the first figure generated by FlightAnalyzer.m.
   - One plot that shows the time evolution of altitude. This plot is the second figure generated by FlightAnalyzer.m.

In your report, for each of the two plots, briefly discuss whether the data is what you expected and why. Tell us if you are not satisfied with the control performance and list reasons that could explain it.

8. **Repeat steps 3-6 with a new LQR controller** that has faster altitude dynamics. Try simulating controllers generated with different weights to find a good balance between speed and small overshoot. It is very possible that the new control will decrease your control system's robustness significantly and, as a result, the real drone becomes unstable. In that case, you need to iterate your choice of LQR parameters.

   NOTE: Keep in mind that the control inputs into the plant are the four motor commands, which are proportional to the thrust provided by the motors. The LQR-optimization tries to minimize costs caused by these commands. Gaining altitude (hence changing the z-state) requires all four motors to provide considerably more thrust while changing e.g. the roll angle can be achieved by small differences in thrusts. To considerably change the altitude response you should therefore mainly tweak the overall control effort costs ρ and the weight on the state z.

   Include relevant plots of the simulated controller and the controller implemented on the drone and comment on the performance of the controller in simulation and on the drone.

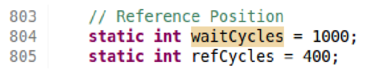
9. **Repeat steps 3-6 with a new LQR controller** that tracks a square in the $X - Y$ plane at constant altitude. The length of the square edge is 1 meter. To do this you will modify the `rsedu_control.c` file in the `MIT_MatlabToolbox/trunk/embcode` folder. The main control funtion in this file is also called `RSEDU_control` and begins at approximately line 806. The primary flight control occurs at approximately line 1280 in this file (the code before this initializes the optical flow and reference value server, calibrates the sensors, and commands the drone to take-off).

   You can see in this section of the code that the orientation reference (`Drone_Compensator_U_orient_refin`) and altitude reference (`Drone_Compensator_U_pos_refin[2]`) gets set if they are changed by the reference value server. Modify the code so that the X reference (`Drone_Compensator_U_pos_refin[0]`) and Y reference (`Drone_Compensator_U_pos_refin[1]`) change so that the drone tracks the 1-meter square. For smoother control you should have these values change gradually, not just change from 0 to 1 instantaneously.

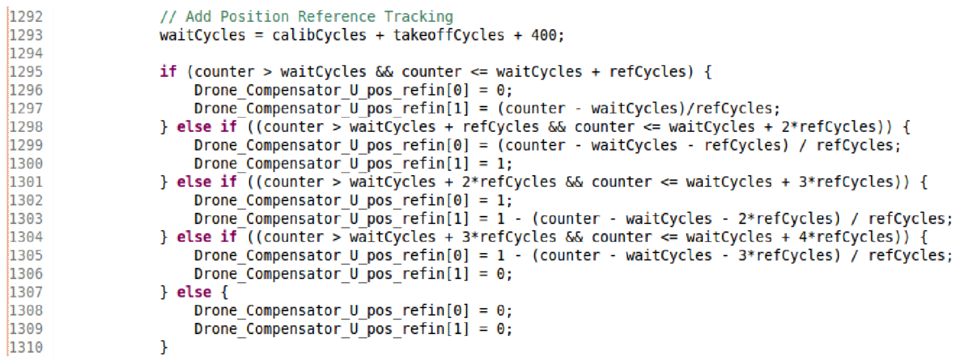
   Note that this code runs at 200Hz and the variable `counter` is the number of times the code has executed. Therefore, if `counter = 1000` then the code has been running for 5 seconds.

   Include relevant plots of the simulated controller (you can modify the simulation reference commands in the `ReferenceValueServer` block in the `sim_quadrotor` Simulink model) and the controller implemented on the drone and comment on the performance of the controller in simulation and on the drone. Include the X, Y, and Z reference signals in your plots.

An example of the code:



```c
// Reference Position
static int waitCycles = 1000;
static int refCycles = 400;
```



```c
        // Add Position Reference Tracking
        waitCycles = calibCycles + takeoffCycles + 400;

        if (counter > waitCycles && counter <= waitCycles + refCycles) {
            Drone_Compensator_U_pos_refin[0] = 0;
            Drone_Compensator_U_pos_refin[1] = (counter - waitCycles)/refCycles;
        } else if ((counter > waitCycles + refCycles && counter <= waitCycles + 2*refCycles)) {
            Drone_Compensator_U_pos_refin[0] = (counter - waitCycles - refCycles) / refCycles;
            Drone_Compensator_U_pos_refin[1] = 1;
        } else if ((counter > waitCycles + 2*refCycles && counter <= waitCycles + 3*refCycles)) {
            Drone_Compensator_U_pos_refin[0] = 1;
            Drone_Compensator_U_pos_refin[1] = 1 - (counter - waitCycles - 2*refCycles) / refCycles;
        } else if ((counter > waitCycles + 3*refCycles && counter <= waitCycles + 4*refCycles)) {
            Drone_Compensator_U_pos_refin[0] = 1 - (counter - waitCycles - 3*refCycles) / refCycles;
            Drone_Compensator_U_pos_refin[1] = 0;
        } else {
            Drone_Compensator_U_pos_refin[0] = 0;
            Drone_Compensator_U_pos_refin[1] = 0;
        }
```