Handed out: October 10th, 2015 **_Due: October 16th, Friday_**

16.30/16.31 Problem Set 2

1 Eigenvalue Assignment for Two-state One-input Systems

Consider the state-space model

$$\dot{\mathbf{x}} = \begin{bmatrix} -5 & 1 \\ 14 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u,$$
$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{x}.$$

Suppose we want to apply a full-state feedback controller of the form $u = -K\mathbf{x}$.

- Find the open-loop eigenvalues (without computing the transfer function).
- Select K such that the closed-loop poles are placed at the roots of
$$\lambda^2 + 2\zeta\omega_n\lambda + \omega_n^2 = 0.$$
- Use your analytic expression, in terms of K, from part (b) to support the following claim: "Moving the eigenvalues far away from the imaginary axis, thus making the closed-loop system dynamics fast, requires large gains. Hence, for fast closed loop dynamics, we need to invest a large amount of control effort."
- Select K to place the closed-loop poles at $\lambda = -9$ and $\lambda = -4$.

2 Eigenvalue Placement for the Drone by Recognizing Separable Structure

In this problem, we will represent the drone dynamics in a way that the dynamics become separable. This will allow us to design a separate controller for each independent loop. We will test this controller on a more complex (but still linear) model of the drone.

2.1 Drone Dynamics

Let us define some notation first:

- $\mathbf{P}$ are inertial XYZ-world coordinates

$$\mathbf{P} = (X, Y, Z)'$$

  of the drone's center of mass.

- $\dot{\mathbf{p}}$ is the quadroter's velocity wrt. the world coordinate frame, expressed in the body frame

$$\dot{\mathbf{p}} = (\dot{x}, \dot{y}, \dot{z})'$$

- $\mathbf{O}$ is the euler-angles $(yaw, pitch, roll)$

$$\mathbf{O} = (\psi, \theta, \phi)'$$

  relating to a rotation R_{B2W} to translate a vector expressed in the drone's bodyframe to world coordinates by subsequent rotations about euler-roll about x, pitch about y, yaw about z.

- W^{-1} transforms the body-angular rates

$$\dot{\mathbf{o}} = (p, q, r)'$$

  about local x-y-z-axes to euler-rates

$$(\dot{O}) = (\dot{\psi}, \dot{\theta}, \dot{\phi})'$$

- $\mathbf{T}$ is the total thrust generated by the rotors expressed in the body frame, τ_{yaw} the total torque about the (bodyframe) z-axis resulting from propeller drag and angular acceleration, τ_{pitch} resp. τ_{roll} the resulting torque about (bodyframe) y-axis resp. x-axis, $\mathbf{G}$ the gravitational vector expressed in world coordinates and $\mathbf{J}$ the quadrotor's inertia expressed in the body frame.

Now, let us define the state variables and the input variables:

- The state variables vector, $\mathbf{x}$, is defined as follows:

$$\mathbf{x} = \begin{bmatrix} \mathbf{P} \\ \mathbf{O} \\ \dot{\mathbf{p}} \\ \dot{\mathbf{o}} \end{bmatrix}$$

- The input variables are:

$$T_x, T_y, T_z, \tau_{roll}, \tau_{pitch}, \tau_{yaw}$$

The quadrotor dynamics result in:

$$\begin{aligned}
\dot{\mathbf{P}} &= R_{B2W} \cdot \dot{\mathbf{p}} \\
\dot{\mathbf{O}} &= W^{-1} \cdot \ \dot{\mathbf{o}} \\
\ddot{\mathbf{p}} &= R_{W2B} \cdot \mathbf{G} + \frac{\mathbf{T}}{m} - \dot{\mathbf{o}} \times \dot{\mathbf{p}} \\
\mathbf{J}\ddot{\mathbf{o}} &= \begin{bmatrix} \tau_{roll} \\ \tau_{pitch} \\ \tau_{yaw} \end{bmatrix} - \dot{\mathbf{o}} \times \mathbf{J} \cdot \dot{\mathbf{o}}
\end{aligned}$$

$$R_{B2W} = \begin{bmatrix} c(\theta)c(\psi) & c(\psi)s(\theta)s(\phi) - c(\phi)s(\psi) & s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta) \\ \\ c(\theta)s(\psi) & c(\phi)c(\psi) + s(\theta)s(\phi)s(\psi) & c(\phi)s(\theta)s(\psi) - c(\psi)s(\phi) \\ \\ -s(\theta) & c(\theta)s(\phi) & c(\theta)c(\phi) \end{bmatrix}$$

where

$$R_{W2B} = \begin{bmatrix} c(\psi)c(\theta) & c(\theta)s(\psi) & -s(\theta) \\ c(\psi)s(\phi)s(\theta) - c(\phi)s(\psi) & c(\psi)c(\phi) + s(\theta)s(\phi)s(\psi) & c(\theta)s(\phi) \\ s(\psi)s(\phi) + c(\psi)c(\phi)s(\theta) & c(\phi)s(\psi)s(\theta) - c(\psi)s(\phi) & c(\phi)c(\theta) \end{bmatrix}$$

$$W^{-1} = \begin{bmatrix} 0 & \frac{s(\phi)}{c(\theta)} & \frac{c(\phi)}{c(\theta)} \\ 0 & c(\phi) & -s(\phi) \\ 1 & s(\phi)tan(\theta) & c(\phi)tan(\theta) \end{bmatrix}$$

If you are curious, the drone dynamics takes the following final form, when we execute the matrix operations in the equations above:

$$\begin{aligned}
\dot{X} =& \dot{z}(s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta)) - \dot{y}(c(\phi)s(\psi) - c(\psi)s(\theta)s(\phi)) + \dot{x}c(\theta)c(\psi) \\
\dot{Y} =& \dot{y}(c(\phi)c(\psi) + s(\theta)s(\phi)s(\psi)) - \dot{z}(c(\psi)s(\phi) - c(\phi)s(\theta)s(\psi)) + \dot{x}c(\theta)s(\psi) \\
\dot{Z} =& \dot{z}c(\theta)c(\phi) - \dot{x}s(\theta) + \dot{y}c(\theta)s(\phi) \\
\\
\dot{\psi} =& (rc(\phi))/c(\theta) + (qs(\phi))/c(\theta) \\
\dot{\theta} =& qc(\phi) - rs(\phi) \\
\dot{\phi} =& p + (rc(\phi)s(\theta))/c(\theta) + (qs(\theta)s(\phi))/c(\theta) \\
\\
\ddot{x} =& -gs(\theta) + \frac{1}{m}T_x + \dot{y}r - \dot{z}q \\
\ddot{y} =& gc(\theta)s(\phi) + \frac{1}{m}T_y + \dot{z}p - \dot{x}r \\
\ddot{z} =& gc(\theta)c(\phi) + \frac{1}{m}T_z + \dot{x}q - \dot{y}p \\
\\
\dot{p} =& (\tau_{roll} + J_y qr - J_z qr)/J_x \\
\dot{q} =& (\tau_{pitch} - J_x pr + J_z pr)/J_y \\
\dot{r} =& (\tau_{yaw} + J_x pq - J_y pq)/J_z
\end{aligned}$$

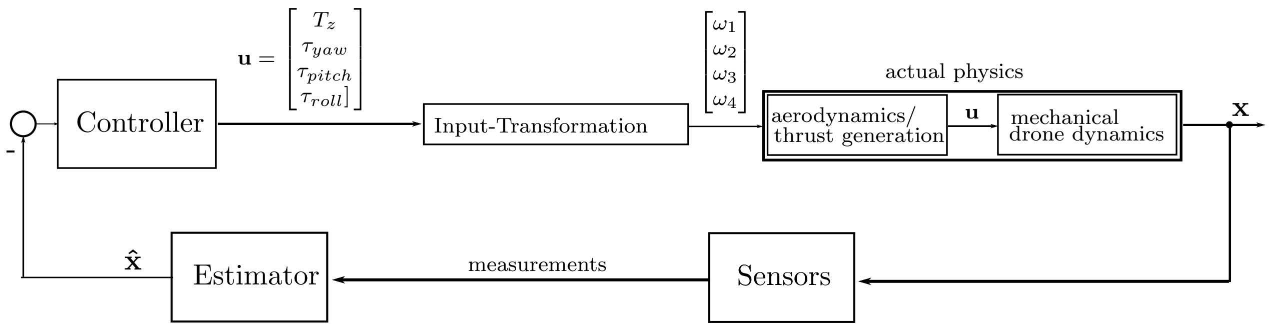
We further assume that the generated total thrust is fully aligned with the quadrotor's z-axis, so T_x and T_y are 0. For the Matlab implementation we therefore define the plant input as

$$\mathbf{u} = \begin{bmatrix} T = T_z \\ \tau_{yaw} \\ \tau_{pitch} \\ \tau_{roll} \end{bmatrix}$$

One note: With this setup, the inputs into our drone plant are the four "mechanical" inputs **u** (total thrust and torques around body-frame axis). However, what we have to provide to the plant's actuators (the motors) are, of course, motor speeds $\omega_{\mathbf{i}}$. The conversion from those four mechanical inputs to four motor commands is, with our simplified model, a bijective $\Re^4 \rightarrow \Re^4$-mapping (no dynamic system!). We can therefore move this transformation into the drone plant - and design the controller for the more intuitive mechanical inputs **u** instead of designing the controller for the motor speeds ω_i. Please have a look at Figure 1.

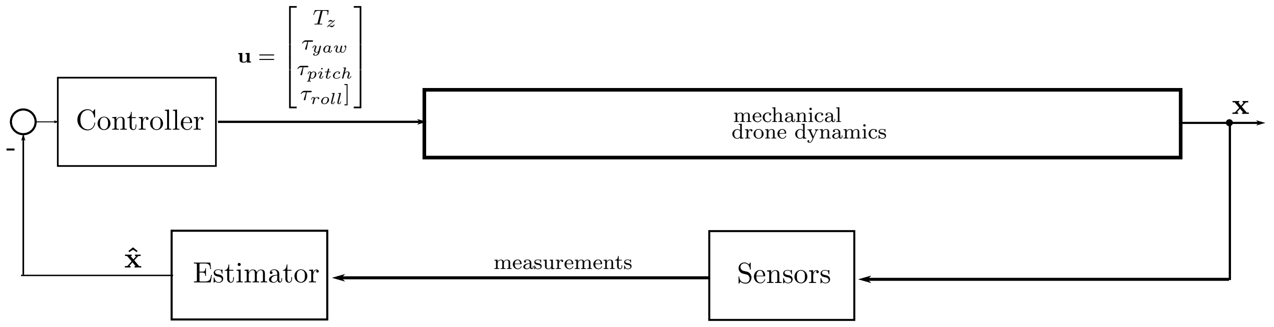
2.2 Linearization, Dynamic Analysis, and Feedback Control

1. Please update your toolbox by downloading the toolbox from https://github.com/Parrot-Developers/RollingSpiderEdu (or pulling the current version if you use git). You should rerun BuildUtils.sh.



$\mathbf{u} = \begin{bmatrix} T_z \\ \tau_{yaw} \\ \tau_{pitch} \\ \tau_{roll} \end{bmatrix}$
$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix}$
actual physics
Controller
Input-Transformation
aerodynamics/ thrust generation
u
mechanical drone dynamics
x
-
$\hat{\mathbf{x}}$
Estimator
measurements
Sensors

(a) Practical implementation scheme for drone control



$\mathbf{u} = \begin{bmatrix} T_z \\ \tau_{yaw} \\ \tau_{pitch} \\ \tau_{roll} \end{bmatrix}$
Controller
mechanical drone dynamics
x
-
$\hat{\mathbf{x}}$
Estimator
measurements
Sensors

(b) Equivalent control design scheme for drone control

Figure 1: Drone Control Schemes

2. Please look at the following Matlab file in the toolbox:
/trunk/matlab/Simulation/controllers/controller_fullstate/ LinearDroneAndPolePlaceControl.m

   In this file, we implemented these drone dynamics in Matlab using the Symbolic Math Toolbox. Section 1.1 of this file builds the dynamics of the drone model and linearizes the model around a hover condition that is 1.5m above the ground. Notice that the linearization is done using the Symbolic Math toolbox of Matlab. The Jacobian is computed in the symbolic form, and then the equilibrium conditions are substituted in to compute the A and B matrices.

   Please run the Section 1.1 of the LinearDroneAndPolePlaceControl.m file and answer the following questions:

   (a) What are the A and B matrices after the linearization procedure?

   (b) What are the inputs and states of this linear time-invariant system?

3. The dynamical system of the drone boasts a 12-dimensional state. Finding a full-state feedback controller would therefore require defining 12 eigenvalues.

   We would not intuitively know which states are affected by which modes. To overcome this problem, we check for possibilities to decouple the system. We therefore transform the system's A-matrix into its Jordan form. You will find that the transformed A-matrix of the simple dynamics reveals that decoupling is possible: States related to motion in x-direction are decoupled from states related to motion in y-direction, z-direction and yaw motion.

We can therefore place eigenvalues and find the state-feedback matrix for the four decoupled systems separately, come up with a joint fullstate-feedback controller and transform it back to work with the original system.

Please look at LinearDroneAndPolePlaceControl.m, paragraph 2.1, where all except a few steps are implemented. Please implement the missing steps to design two controllers with the following closed-loop dynamics:

- Controller 1 closed-loop dynamics:
  yaw-dynamics: $-3, -3.1$
  x-dynamics: $-9 \pm 6i; -0.18 \pm 1.8i$
  y-dynamics: $-60; -4; -0.1 \pm 2i$
  z-dynamics: $-2; -2.1$
- Controller 2 closed-loop dynamics:
  yaw-dynamics: $-3, -3.1$
  x-dynamics: $-9 \pm 6i; -0.18 \pm 1.8i$
  y-dynamics: $-60; -4; -0.1 \pm 2i$
  z-dynamics: $-5; -5.1$

When designing the controllers (that is, determining the K matrix), you can use Ackermann's formula. Please use Matlab. Calculations would be tedious by hand. You can also use the `place` function in Matlab for each of the four control loops.

Please answer the following questions:

(a) What is the difference between the two controllers? How do you expect them to behave with respect to one another? Please describe in one paragraph.

(b) What are the feedback controllers that you found for each of the controllers?

4. Now let us linearize a more complex drone model. For this purpose, we will linearize the drone model that runs within the Simulink simulation of the toolbox. The complex model takes into account complex aerodynamic effects, which we do not describe here. Those interested can investigate the model via Simulink.

   Please use the linearizeDrone_umechTostate.slx and Simulink's ControlDesign/Linear Analysis tool to find the A,B,C,D matrices around hover condition for this more complex model.

   (a) What are the A an B matrices?

5. (a) What are the eigenvalues of the A matrix you found for the simple model (2.2.2)?

   (b) What are the eigenvalues of the A matrix that you found for complex model (2.2.4)?

   (c) What can you say about the dynamics of both system looking only at the eigenvalues? Please describe briefly in one paragraph.

6. Use Matlab/Simulink to simulate the two controllers that you found in 2.2.3 with the linear plant model that you found in 2.2.2.

An easy way to implement the effect of having modeled the system around our equilibrium point is to simply set the initial condition of the system to $-\mathbf{x}_{eq}$, i.e.
$\mathbf{x_0} = 0, 0, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0$ and then have the system being controlled into the origin.

(a) Please plot positions and orientations.

Note that this simulation will not involve the complex nonlinear drone model of our Matlab toolbox, which we used for the first lab. This is a simple simulation, where the plant model is a simple linear system with A and B matrices found in 2.2.2.

7. Use Matlab/Simulink to simulate the two controllers that you found in 2.2.3 with the linear model that you found in 2.2.4.

   (a) Please plot positions and orientations.

   Note that this simulation will not involve the complex nonlinear drone model of our Matlab toolbox, which we used for the first lab. This is a simple simulation, where the plant model is a simple linear system with A and B matrices found in Part 2.2.4 of this problem.

3 Time spent

Please indicate the approximate amount of time you spent on this assignment.