ROB 599 HW 1

Prof. Johnson-Roberson and Prof. Vasudevan

15 Sept, 2017

Submission Details

Use this PDF only as a reference for the questions. You will find a code template for each problem on Cody. You can copy the template from Cody into MATLAB on your personal computer in order to write and test your own code, but *final code submission must be through Cody*.

Problem 1: Pole Placement [25 points]

Consider a simplified model of an automobile traveling up an inclined road, given by the following differential equation:

$$m\frac{dv}{dt} = \frac{n}{r}u - mgC_r\text{sgn}(v) - \frac{1}{2}\rho C_d a v^2 - mg\sin\theta \tag{1}$$

where v is the longitudinal velocity, u is the engine torque, θ is the road grade, m is the vehicle mass, r is the wheel radius, n is the gear ratio (gearbox + final drive), g is the gravity constant, C_r is the rolling resistance, ρ is the density of air, C_d is the vehicle's drag coefficient, and a is the vehicle's frontal area. These constants will be used for the remainder of the problem:

$m = 1500$ [kg]	$r = 0.4$ [m]	$n = 4.8$	$g = 9.81$ [m/s^2]
$C_r = 0.01$	$\rho = 1.3$ [kg/m^3]	$C_d = 0.32$	$a = 2.4$ [m^2]

1.1 [3 points] Assume you are traveling on a road with constant inclination $\theta_e = 2°$ and at a constant longitudinal velocity $v_e = 20$ [m/s]. Determine the engine torque required to maintain this equilibrium point, i.e u_e= ?

1.2 Linearized System

1.2.1 [5 points] Linearize system (1) about the equilibrium point in part 1.1 and place in the form:

$$\frac{d\tilde{v}}{dt} = A\,\tilde{v} + B\,\tilde{u} + F\,\tilde{\theta} \tag{2}$$

where $\tilde{v} = v - v_e$, is the state we are interested in controlling, $\tilde{u} = u - u_e$, is the input we are applying, and $\tilde{\theta} = \theta - \theta_e$ is treated as a disturbance. This is a linear system where velocity is the state, engine torque is the input, and the road grade is treated as a disturbance.
What are the matrices A, B, and F?

1.2.2 [4 points] Ackerman's formula is used to determine the feedback gains necessary to place the poles of a n-dimensional linear system at desired locations:

$$k^T = [0\ 0 \cdots 1][B\ AB \cdots A^{n-1}B]^{-1}\Phi(A) \tag{3}$$

where k is the vector of feedback gains for each input, A and B are the state and input matrices of the system, and $\Phi(A)$ is the desired characteristic polynomial.

Use Ackerman's formula to determine the feedback gain required to place the pole of system (2) at -1, i.e. k = ?

1.2.3 [3 points] With feedback the input to system (2) is $\tilde{u} = -k\tilde{v}$
Use k from part 1.2.2. Use the initial condition $\tilde{v}_0 = -1$ [m/s]. Simulate the system response with $\tilde{\theta} = 0°$. Notice how the system converges to its equilibrium velocity ($\tilde{v} \to 0$). Now, with the same initial condition, simulate the system response with $\tilde{\theta} = 3°$. Find the steady-state velocity error after 10 seconds, i.e. $\tilde{v}(10)$= ?

1.3 Integral Action

Now add a second state, z, where $\frac{dz}{dt} = \tilde{v}$. This state tracks the accumulated error between the actual velocity and equilibrium velocity. This is called integral action, and is used to get rid of steady state error when there is an unforseen disturbance (e.g. terrain variation) when we do feedback. The new equation of motion for the linearized system is given by:

$$\frac{d}{dt}\begin{bmatrix}\tilde{v}\\ z\end{bmatrix} = A_I\begin{bmatrix}\tilde{v}\\ z\end{bmatrix} + B_I\begin{bmatrix}\tilde{u}\end{bmatrix} + F_I\begin{bmatrix}\tilde{\theta}\end{bmatrix} \tag{4}$$

Note: the subscript I is just used to indicate matrices and vectors that are for the system with integral action. The subscript has not mathematical meaning, we're doing this to help the auto-grader distinguish between variable names.

1.3.1 [4 points] What are the matrices A_I, B_I, and F_I in system (4)? Use the parameters and equilibrium point from 1.1.

1.3.2 [3 points] Use Ackerman's formula to find the feedback gains, k_I, needed to place the system's poles at -1 and -2.

1.3.3 [3 points] With feedback, the input to system (4) is $\tilde{u} = -k_I \begin{bmatrix} \tilde{v} \\ z \end{bmatrix}$

Using the conditions from 1.2.3 and k_I from 1.3.2, simulate the system response for 10 seconds with $\tilde{\theta} = 3°$. Use the initial condition $\tilde{v}_0 = -1$ [m/s], $z(0) = 0$ [m]. Find the steady-state velocity error after 10 seconds, i.e. $\tilde{v}(10)$= ?

1.4 Extra Practice (not graded)

1.4.1 Simulate different feedback gains and check how they affect the linear system (4) response. Pay attention to the overshoot (how far past the equilibrium point the trajectory goes) and damping (how quickly it settles to the equilibrium and if it oscillates at all).

1.4.2 Using the feedback gains in part 1.3.2, simulate the response of the nonlinear system (1) for equilibrium velocities $v_e = \{12,\ 14,\ 16\}$ [m/s] with the initial condition $v = 10$ [m/s]. Now place a lower and upper limits of 0 and 350 [Nm] on the engine torque, u, meaning if at anytime during your feedback simulation, the required engine torque is >350 [Nm], set the engine torque to 350 Nm. This is called actuator saturation. Similarly do not allow the engine to apply negative torque (as this would be the job of the brakes). Simulate the system response. Plot and compare the results of your simulation with and without actuator saturation.

Problem 2: System Modes [15 points]

Consider a linear system with three states: $x = [x_1, x_2, x_3]^\top \in \mathbb{R}^3$. The evolution of the system is described by the following differential equation:

$$\dot{x} = Ax = \begin{bmatrix} -3 & 0 & 0 \\ 0 & -1 & -3 \\ 0 & 1 & -2 \end{bmatrix} x \tag{5}$$

2.1 [3 points] Find the eigenvalues and eigenvectors of the system.

2.2 [3 points] Notice that there is one real eigenvector. With symbolic MATLAB, write an expression for the solution $x(t)$ to (5) when

$$x(0) = \begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix}$$

2.3 [4 points] Numerically solve (using `ode45`) for the trajectory $x(t)$ when the system starts along the real part of one of the imaginary eigenvectors. Let the time span be $t \in [0, 10]$ [s] with $\Delta t = 0.1$ [s].

$$x(0) = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

It may be useful to plot your results as a sanity check.

2.4 [5 points] Compute the following integral:

$$\int_0^T x(t)^T v dt \quad \text{where} \quad v = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{6}$$

where $x(t)$ is the solution that was found and plotted in 2.3.

Problem 3: 2D Affine Transformations in Image Processing

Perform the following 2D homogeneous affine transformations on the image of an apple:

1. Translate the apple by 250 pixels along the $x-$axis and 200 along the $y-$axis in the current frame. After this transformation, the origin of the current frame should be at $(250, 200)$ in the world frame.
2. Rotate the apple about the current frame $z-$axis -90 degrees.
3. Translate the apple by 100 pixels along the current frame $y-$axis. After this transformation, the origin of the current frame should be at $(350, 200)$ in the world frame.
4. Rotate the apple about the world frame $z-$axis 45 degrees. After this transformation, the origin of the current frame should be at around $(106, 389)$ in the world frame.
5. Scale the apple by a factor of 2 in the current frame.
6. Translate the apple by -150 pixels along the world frame $y-$axis. After this transformation, the origin of the current frame should be at around $(106, 239)$ in the world frame.

The current/body frame of the apple image is axis-aligned with the fixed Cartesian world frame. This means that the first pixel of the apple image, $(0, 0)$, is located in the world frame at $(0, 0)$, etc. The world frame is 500×500 pixels.

All the rotations are assumed to follow the right-hand rule.

Write a MATLAB function that takes in the apple image file path and outputs the image matrix `new_apple`, which is the transformed apple image placed in the larger world space. Your output image, `new_apple`, should be a 500×500 gray scale image in 8-bit unsigned integers.

IMPORTANT NOTES:

1. Your .m file submission should be of the form:

```
function new_apple = apple_transformations(file_path)
    apple = rgb2gray(imread(file_path))

    % some code for visualization (optional)
    %figure(1); clf()
    %imagesc(apple)
    %axis equal; xlabel('x'); ylabel('y');
```

```
    % your transformation code here
    % ...

end
```

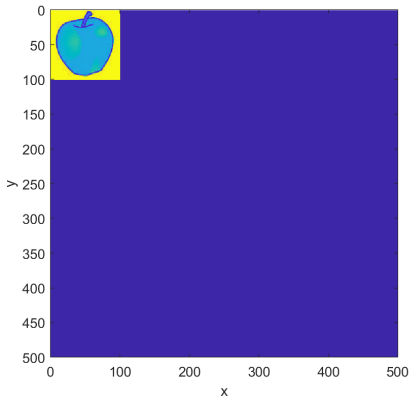
2. You can use the provided `load_apple` function to load the `cartoonapple.jpeg` into your workspace (the input to `load_apple` is image file path, i.e, `'cartoonapple.jpg'`).

3. When modifying an image to create a new image using affinities, you need to perform the transformations in such a way as to prevent 'holes' from appearing in the final output image. This often happens with rotation and scaling transformations, which do not necessarily transform integer (x,y) points to integer (x,y) points. Therefore, you must find the correspondence of each pixel in the 500×500 output image, `new_apple`, to a pixel in the original space (see 3 for how the apple image is originally oriented in world space). To 'find the correspondences' between the two image spaces, consider the output image space of `new_apple` to be a 500×500 grid of pixels, where each pixel is represented by a point (x,y). You must calculate the transformation of all the (x,y) points from the `new_apple` image space into the original image space, and assign the `new_apple` (x,y) pixel the color value of its corresponding original image pixel. When the transformed `new_apple` (x,y) point lies between the pixels in the original image (i.e, it is a non-integer value), use the nearest neighbors algorithm to calculate the correct color for the `new_apple` (x,y) pixel.

4. Note that, when creating the final image `new_apple`, you need to index `new_apple(y,x)` because MATLAB indexing for images begins at `(1,1)` and is of the form `im(row,col)`.

5. Useful functions: `meshgrid`, `uint8`, `contourf`, `imagesc`

Below is a visualization of the grayscale apple image in the axis-aligned starting location in the fixed Cartesian world image frame.



Problem 4: 3D Homogeneous Transformations in Image Processing

For this problem, we are going to perform the following homogeneous transformations on a 3D hourglass object, and then generate a 2D image of the 3D transformed 3D object. Assume all positive rotations are counter-clockwise. Assume that the current frame of the hourglass is axis-aligned with the world frame.

1. Translate the hourglass by 250 mm along the world $x-$axis and 250 mm along the world $y-$axis. After this transformation, the origin of the current frame should be at $(250, 250, 0)$ in the world frame.

2. Rotate the hourglass 105 degrees about the world $y-$axis and scale by a factor of 50 in the world x direction and 0.25 in the world z direction. After this transformation, the origin of the current frame should be at around $(-3235, 250, -60)$ in the world frame.

3. Translate the hourglass by 350 mm along the current $y-$axis. After this transformation, the origin of the current frame should be at around $(-3235, 600, -60)$ in the world frame.

4. Rotate the hourglass by 45 degrees about the current $x-$axis. After this transformation, the origin of the current frame should be at around $(-3235, 600, -60)$ in the world frame.

5. Project from the fixed 3D world frame to a 2D image plane defined by the perspective matrix P and intrinsic matrix I.

For the perspective matrix, use the origin as the center of projection and $z = 1$ as the image plane. For the intrinsic matrix, assume a focal length of 1. Our final image space is 1000×1000 pixels. Note that you will need to calculate the α, β, C_x, and C_y parameters from the camera mm coordinate space to the 1000×1000 pixel coordinate space.

Hint: Projection all the points in 3D to the image plane first, then translate and scale the points on the image plane such that

$$\min_i x_i = \min_i y_i = 0$$

$$\max_i x_i = \max_i y_i = 999$$

Note that transformations (1-4) are 3D to 3D homogeneous transformations. The final transformation is a 3D to 2D projective transformation.

For the hourglass, use the attached function `gen_hourglass` to get the x,y,z Cartesian locations. The output of `gen_hourglass` is a `num_points x 4` dimensional matrix, where each row is the $(x, y, z, 1)$ location of that given point. Your inputs to `gen_hourglass` are `scale = 100`, and `grid_grain = 10`. The function that you submit should use the following function template:

```
function pnt_2D = transform3Dto2D(ws)
    % inputs: ws is the dimension of the world space, ws x ws
```

```
    % your code here
    % ...
end
```

The output of your function, `pnt_2D`, is a `3 x num_point` dimensional matrix, where each column is an $\begin{bmatrix} x_i & y_i & 1 \end{bmatrix}^\top$ hourglass point transformed into the 2D image plane (pixel space).

$$\texttt{pnt\_2D} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

Note that these final points are not integers (you do not need to perform nearest neighbor, since you are not creating an image!).

Useful functions: `scatter3` for visualization.

Below is a visualization of the hourglass point cloud in the axis-aligned starting location in the world frame.