ROB 599 HW 3

Prof. Johnson-Roberson and Prof. Vasudevan

Due 2 Nov, 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: Tire Forces: Mild Maneuver [25 points]

The dynamics for a bicycle model, assuming constant longitudinal velocity, v_x, are given by:

$$\dot{Z} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{v_y} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} v_x \cos(\psi) - v_y \sin(\psi) \\ v_y \cos(\psi) + v_x \sin(\psi) \\ r \\ \frac{1}{m}(F_{yr} + F_{yf}\cos(\delta) - m v_x r) \\ \frac{1}{I_z}(-bF_{yr} + aF_{yf}\cos(\delta)) \end{bmatrix} \tag{1}$$

where δ is the front wheel angle, v_y is the lateral velocity, r is the yaw rate, F_{yf}, F_{yr} are the lateral tire force at the front and rear tires, a is the distance to the center of mass from the front axle, and b is the distance to the center of mass from the rear axle.

The Pacejka Tire model, sometimes referred to as the "Magic Tire Model", can be used to model the lateral forces as functions of the slip angles at the front and rear tires α_f, α_r. A simplified

version of the Pacejka Model is presented below:

$$F_{yf} = F_{z,f} D \sin\Big(C \arctan\Big(B(1-E)\alpha_f + E \arctan(B\alpha_f)\Big)\Big) \tag{2}$$

$$F_{yr} = F_{z,r} D \sin\Big(C \arctan\Big(B(1-E)\alpha_r + E \arctan(B\alpha_r)\Big)\Big) \tag{3}$$

$$F_{zf} = \frac{b}{L} mg \tag{4}$$

$$F_{zr} = \frac{a}{L} mg \tag{5}$$

$$\alpha_f = \delta - \arctan\left(\frac{v_y + ar}{v_x}\right) \tag{6}$$

$$\alpha_r = -\arctan\left(\frac{v_y - br}{v_x}\right) \tag{7}$$

Where F_{zf}, F_{zr} are the vertical tire forces at the front and rear tire, L is the wheelbase, and g is the acceleration of gravity. The remaining coefficients (B, C, D, E) are empirically found values.

In this problem use the following parameters for the vehicle and tires: $m = 1500$ $[kg]$, $I_z = 2420$ $[kg \cdot m^2]$, $a = 1.14$ $[m]$, $L = 2.54$ $[m]$, $g = 9.81$ $[m/s^2]$, $v_x = 20$ $[m/s]$, $B = 10$, $C = 1.3$, $D = 1$, $E = 0.97$. Note that these coefficients assume all angles are in **radians**. For all simulations use the timespan $T = 0 : 0.01 : 1$.

1.1 Linear Tire Dynamics [5 points] For simplicity the tire dynamics are often assumed to be linear with respect to the slip angle $F_{yi} \approx C_{\alpha_i} \alpha_i$. The cornering stiffness, C_{α_i} is defined as

$$C_{\alpha_i} = \frac{\partial F_{yi}}{\partial \alpha_i}\Big|_{\alpha_i = 0} \tag{8}$$

For the above tire model, the cornering stiffness can be calculated as $C_{\alpha_i} = F_{zi} \cdot B \cdot C \cdot D$. Small angle assumptions are also made about the slip and wheel angles so the vehicle dynamics can be

written as:

$$\dot{Z} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{v_y} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} v_x\cos(\psi) - v_y\sin(\psi) \\ v_y\cos(\psi) + v_x\sin(\psi) \\ r \\ \frac{1}{m}(F_{yr} + F_{yf} - mv_x r) \\ \frac{1}{I_z}(-bF_{yr} + aF_{yf}) \end{bmatrix} \tag{9}$$

$$F_{yf} = C_{\alpha_f}\alpha_f \tag{10}$$

$$F_{yr} = C_{\alpha_r}\alpha_r \tag{11}$$

$$\alpha_f = \delta - \frac{v_y + ar}{v_x} \tag{12}$$

$$\alpha_r = -\frac{v_y - br}{v_x} \tag{13}$$

Compute the front and rear cornering stiffness C_{α_f} and C_{α_r} using the vehicle parameters defined above.

1.2 Mild Maneuver

1.2.1 Simulate Reference Trajectory with Linear Tire Dynamics [5 points] Using Euler's method with a timespan of $T = 0 : 0.01 : 1$ and an initial condition of $Z_{eq}(0) = [0\ 0\ 0\ 0\ 0]^T$ simulate the trajectory of the system using the steering input $\delta_{eq}(t) = \frac{\pi}{180}\sin(2\pi t) - 0.00175$. Use the dynamics described in equations (9-13). Save your equilibrium trajectory as a 5x101 double named `Z_eq`.

1.2.2 LQR [5 points] Now linearize the system described by equations (9-13) about the trajectory you found in part 1.2.1. Use the provided function `lqr_LTV` to find the optimal feedback gains, K, at each time step when $Q = I_{5\times5}$ and $R = 0.5$. Your answer, `K`, should be a 1x101 cell array where each cell is a 1x5 double.

1.2.3 Solve for Tire Forces [5 points] Using the reference trajectory you found in part 1.2.1, calculate the front and rear slip angles and lateral tire forces, at each timestep, using both linear assumptions (equations 10-13) and the Pacejka model (equations 2-7). Plot your results. What is the maximum percent (out of 100) error between the two models? Use the formula

$$100 * \max\left\{\frac{|\text{linear tire force} - \text{Pacejka tire force}|}{|\text{Pacejka tire force}|}\right\}$$

Save your answer as a double called `tireforce_percent_error`.

Note: the point here is not to re-simulate the system with the full nonlinear dynamics before solving for the Pacejka tire forces. The idea is to see how different the linearized and Pacejka tire forces are *along the same system trajectory*.

1.2.4 Feedback with Nonlinear Tire Dynamics [5 points] Using Euler's method with a timespan of $T = 0 : 0.01 : 1$ and an initial condition of $Z(0) = [0\ 0\ 0\ 0\ 0]^T$ simulate the trajectory of the system using the feedback gains you found in part 1.2.2. Assume your vehicle dynamics are described by equations (1-7).

Remember that at each time step, the wheel angle will be given by $\delta(i) = K\{i\}(Z_{eq}(i) - Z(i)) + \delta_{eq}(T(i))$, where $i = 1 : 1 : \text{length}(T)$ is the time *index*. Notice that the steering input δ is applied at each discrete time step in T, but δ_{eq} is a function of time (not time step). So, we evaluate δ_{eq} on at the time $T(i) = \Delta t \cdot i - \Delta t$, which ranges from 0 to 1 [s] with $\Delta t = 0.01$ [s].

Additionally, assume the vehicle has a maximum steering angle of $\pm\frac{45\pi}{180}$ radians. Plot the x,y position and heading of the reference and actual trajectory. What is the maximum position error? Use the formula

$$\max\left\{\sqrt{(x_{eq} - x)^2 + (y_{eq} - y)^2}\right\}$$

Save your answer as a double called `max_distance_error`.

Problem 2: Tire Forces: Extreme Maneuver [25 Points]

Problem 2 continues with all of the information from Problem 1. We will now consider what happens when the vehicle is subject to a more dynamic scenario.

2.1 Extreme Maneuver

2.1.1 Simulate Reference Trajectory with Linear Tire Dynamics [5 Points] Using the same timespan and initial condition, repeat part 1.2.1 using the steering input $\delta_{eq}(t) = \frac{10*\pi}{180}\sin(2\pi t) - 0.0175$. Use the dynamics described in equations (9-13). Save your equilibrium trajectory as a 5x101 double named `Z_eq`.

2.1.2 LQR [5 Points] Now linearize the system described by equations (9-13) about the trajectory you found in part 2.1.1. Use the provided function lqr_LTV to find the optimal feedback gains, $K2$, at each time step when $Q = I_{5\times5}$ and $R = 0.5$. Your answer, `K`, should be a 1x101 cell array where each cell is a 1x5 double.

2.1.3 Solve for Tire Forces [5 Points] Repeat part 1.2.3, but use the reference trajectory you found in part 2.1.1, Plot your results. What the is maximum percent (out of 100) error between the two models? Save your answer as a double called `tireforce_percent_error`.

2.1.4 Feedback with Nonlinear Tire Dynamics [10 Points] Repeat part 1.2.4, use the same timespan and initial condition, but use the reference trajectory you found in part 2.1.1 and the feedback gains from 2.1.2. Plot the x,y position and heading of the reference and actual trajectory. What is the maximum position error? Save your answer as a double called `max_distance_error`.

2.2 Additional Practice (Not Graded)

2.2.1 Change your simulations in parts 1.2.4 and 2.1.4 so that the system starts at $Z(0) = [0\ 0\ 0.1\ 0\ 0]^T$. Why do you think the position error for the mild manuever seems worse now? Using the same initial condition, extend the timespan to $T = 0 : 0.01 : 10$. Do both of the trajectories end up converging?

Problem 3: Visual Odometry

Visual Odometry is the process of estimating a robot's motion (translation and rotation with respect to a reference frame) by observing a sequence of images of its environment. You will be implementing different parts of a simple Visual Odometry pipeline, and will be working with the following pairs of stereo images from the KITTI Visual Odometry dataset:

Figure 1: The two image stereo pairs from KITTI. The stereo pair at time t is shown in the top row, and the stereo pair at $t+1$ is shown in the bottom row. The left column contains the left stereo images, and the right column contains the right stereo images.

A tutorial/general overview of Visual Odometry is given in `http://rpg.ifi.uzh.ch/docs/Visual_Odometry_Tutorial.pdf`

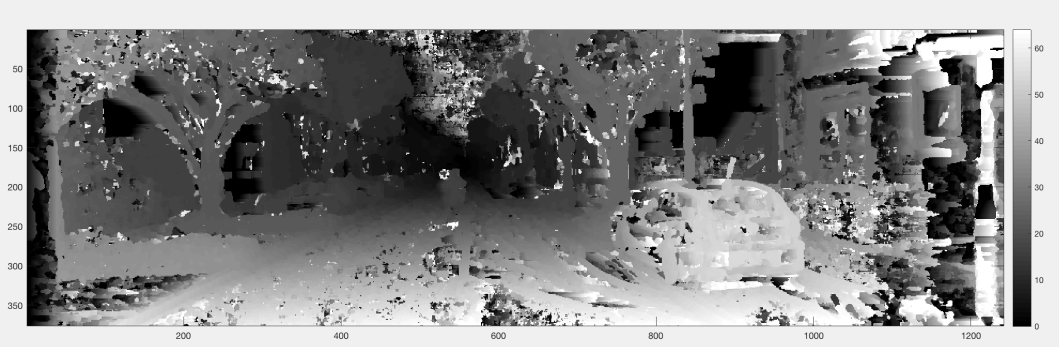
3.1 Stereo Correspondence/Matching: Compute Disparity Map :

Given a pair of rectified stereo images, the problem of stereo matching is defined as follows: for a pixel at coordinate $xl = (x_l, y_l)$ in the left stereo image, find the coordinate $xr = (x_r, y_r)$ of the corresponding pixel in the same row in the right image. The difference $d = xr - x1$ is called the disparity at that pixel, and if we perform this correspondence matching for all pixels in the left image, we will have computed a disparity map for the stereo pair of images.

For further clarification: suppose a particular 3D point in the physical world is located at the position (x, y) in the left image, and the same feature is located on $(x+d, y)$ in the right image, then the location (x, y) on the disparity map holds the value d.

As discussed in class, image rectification makes two stereo images 'parallel', i.e, the rows in each image of a stereo image pair are aligned, which means the y coordinates of corresponding points in each image are the same. Hence, we only need to perform matching along the rows of

Figure 2: Disparity Map for the KITTI stereo pair at time t



each image as opposed to matching along the rows and columns of each image. Thus, we can define disparity at each point in the image plane as: $d = xl - xr$.
You will need to write a function
`[D] = genDisparityMap(I1, I2, min_d, max_d, w_radius)`
that takes as input the left and right images of a stereo pair, the minimum disparity value, the maximum disparity value, and the window radius (window width). The output `D` is the disparity map; this is what will be submitted to the Cody Autograder. The function template for this algorithm is on Cody under 'Learner Template'.
For your implementation, adhere to the following specifications:

- Your stereo pair image input will be from KITTI (the left column shown in Figure 1).
- For each pixel in the left image, extract the 7 by 7 pixel window around it (`w_radius=3.0` input parameter) and use it as a template to search along the same row in the right image for the region of pixels that best matches it.
- Along the right image's row, search over all disparity values d in a disparity region of 0 to 64 pixels (the `min_d` and `max_d` input parameters, respectively) around the pixel's location in the right image.
- Use the sum of absolute differences (SAD) as the similarity metric to compare the left image template to the region in the right image.

PLEASE NOTE THE FOLLOWING:

1. Your disparity map for the two KITTI images should resemble Figure 2.
2. You are **NOT** allowed to use the Matlab function `disparity()`. It implements a more complicated variation of the block matching algorithm, and if you use it your disparity map will not match the autograder solution.
3. An explanation of disparity maps is given in the lecture slides `Lecture_05_matt.pdf`, beginning on slide 47.
4. The following link provides a nice conceptual explanation of pseudo code for this problem: `https://sites.google.com/site/5kk73gpu2010/assignments/stereo-vision#TOC-Update-Dispar`

3.2 Generate 3D Point Cloud using Image Triangulation :

In naive image triangulation, the 3D points X can be determined as,

$$x = P_{\text{left}}X$$

$$x' = P_{\text{right}}X$$

However, because we have errors/noise in the image measurements x and x', the epipolar lines given by $x = \mathbf{P}_{\text{left}}X$ and $x' = \mathbf{P}_{\text{right}}X$ will not necessarily intersect at a point X. This means we need to utilize a method that estimates the 3D point given the uncertainty in the image measurements.
The method for linear triangulation is an analogue to the Direct Linear Transform (which we learned earlier this semester). This means that we want to write our estimation problem in the form of $AX = 0$, and then use the SVD to solve for the optimal solution. The matrix $\mathbf{A}$ is constructed as follows:

$$\mathbf{A} = \begin{bmatrix} x\mathbf{P}_3^{\text{left}} - \mathbf{P}_1^{\text{left}} \\ y\mathbf{P}_3^{\text{left}} - \mathbf{P}_2^{\text{left}} \\ x'\mathbf{P}_3^{\text{right}} - \mathbf{P}_1^{\text{right}} \\ y'\mathbf{P}_3^{\text{right}} - \mathbf{P}_2^{\text{right}} \end{bmatrix}$$

where $\mathbf{P}_{\mathbf{i}}^{\text{left}}$ and $\mathbf{P}_{\mathbf{i}}^{\text{right}}$ are the rows of $\mathbf{P}_{\text{left}}$ and $\mathbf{P}_{\text{right}}$, and (x, y) and (x', y') are corresponding points in the left and right image. The derivation of the $\mathbf{A}$ matrix is stems from the relationship between an image point (using the left image as an example) x and 3D point $\mathbf{P}_{\text{left}}X$ defined by the cross-product $x \times (\mathbf{P}_{\text{left}}X) = 0$, which eliminates the homogeneous scale factor. Note that the first two rows of $\mathbf{A}$ are in fact the cross product $x \times (\mathbf{P}_{\text{left}}X) = 0$ and the second two rows are $x' \times (\mathbf{P}_{\text{right}}X) = 0$.
You will write a function
`point_cloud = gen_pointcloud(Ileft, Iright, Pleft, Pright)`
that calculates a 3D point cloud from the pair of stereo images $Ileft$ and $Iright$ and camera matrices $Pleft$ and $Pright$. The output,`point_cloud`, is the `numPointsx3` matrix that holds the 3D locations of the point cloud; this is what will be submitted to the Cody Autograder. Please see the function template provided in Cody for pseudocode.
You will be given the left and right camera matrices $\mathbf{P}_{\text{left}}$ and $\mathbf{P}_{\text{right}}$, which are $3x4$ projection matrices. Assume that these matrices are the ground truth projection matrices, which means that our only sources of error is in the measured points x in the left image and x' in the right image.

You are given the function:
`[matchedPoints1, matchedPoints2, matchedf1, matchedf2] = detectFeatureMatches(I1,I2)`
that extracts SURF features from the images $I1$ and $I2$, and uses an exhaustive pairwise distance metric to find the 'matches'/correspondences between each images' extracted features. The outputs are the locations of the features for the first and second image, and the features for the first

and second image, respectively, sorted such that `matchedPointsL(i,:)` is the correspondence of `matchedPointsR(i,:)`.

PLEASE NOTE THE FOLLOWING:

1. You are not allowed to use the Matlab `triangulate()` function or the `reconstructScene()` function.
2. A good overview of linear image triangulation is `https://books.google.com/books?id=si3R3Pfa98QC&pg=PA312&lpg=PA312&dq=image+triangulation+maximum+likelihood+estimation&source=bl&ots=aSo3jucc3R&sig=b6ls80MOgKDjekp5eAh_AR-Vebw&hl=en&sa=X&ved=0ahUKEwiOwMz0lPXWAhXl6YMKHSHUCkQQ6AEINjAC#v=onepage&q=image%20triangulation%20maximum%20likelihood%20estimation&f=false`
3. Another good overview that also includes descriptions of other (more accurate) methods to perform triangulation is chapter 2 in the document: `https://www.cs.ubc.ca/grads/resources/thesis/May09/Dubrofsky_Elan.pdf` In this pdf, the linear triangulation method is referred to as 'minimizing the algebraic distance' and is discussed in section 2.2.1 .

3.3 3D to 3D Motion estimation between frames at time t and $t+1$:

Our goal now is to compute the transformation $\mathbf{T_{t+1}}$ that describes the motion between the stereo pair $(I^t_{\text{left}}, I^t_{\text{right}})$ and the stereo pair $(I^{t+1}_{\text{left}}, I^{t+1}_{\text{right}})$:

$$\mathbf{T_{t+1}} = \begin{bmatrix} \mathbf{R}_{t,t+1} & \mathbf{t}_{t,t+1} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$$

While there are a variety of ways to accomplish this estimation, we will follow this outline:
Simple 3D to 3D Motion Estimation Algorithm Steps:

1. For the stereo pair at time t, find the correspondences between the F_t in I^t_{left} and I^t_{right}. Similarly, do this for F_{t+1}, I^{t+1}_{left} ,and I^{t+1}_{right}.
2. Match features F_t and features F_{t+1} between I^t_{left} and I^{t+1}_{left}.
3. Generate point clouds for W_t and W_{t+1} for each stereo image pair at time t and time $t+1$ using linear triangulation.
4. Estimate the transformation $[R \mid t]$, that minimizes the reprojection error. In math-speak, we want to minimize the following:

   $$\sum (j_t - \mathbf{P} * \mathbf{T} * W_{t+1})^2 + (j_{t+1} - \mathbf{P} * \mathbf{T^{-1}} * W_t)^2)$$

   where
   W_t is the 3D point cloud (in homogeneous coordinates) of time frame t

W_{t+1} is the 3D point cloud (in homogeneous coordinates) of time frame $t+1$
j_t is the 3D homogeneous coordinates of the features F_t in I^t_{left}
j_{t+1} is the 3D homogeneous coordinates of the features F_{t+1} in I^{t+1}_{left}
P is the projection matrix of the left camera
T is the $4x4$ homogeneous transformation matrix

This estimation can be done using least squares, or using iterative methods like ICP. However, iterative optimization techniques can be sensitive to noise and susceptible to locally optimum solutions, which means that they require a good initialization to be accurate. On the other hand, stochastic optimization techniques such as RANSAC can find optimal alignments even when substantial noise is present in the input... so use RANSAC. Please see `http://danielwedge.com/ransac/`. :)

5. Concatenate the obtained transformation with previously estimated global transformation.

6. Repeat from 1 at each time step (we will only be doing this for t and $t+1$, so we skip this step for now!).

You will implement a function

```
[R,t] = motionEstimation3Dto3D(It0_left, It0_right, It1_left, It1_right)
```

that performs the above steps. The output, `[R,t]` are the `3x3` rotation matrix and `3x1` translation vector calculated by RANSAC; these will be submitted to the Cody Autograder. Please refer to Cody for the function template/pseudocode.
Your `motionEstimation3Dto3D` will require you to implement the following:

- Calculate left-right image correspondences using SURF features (using `detectFeatureMatches` and Matlabs's `extractFeatures`.
- implement RANSAC to solve the 3D to 3D point cloud alignment. Use the reprojection error as your distance function. Note that we are considering frame $t+1$ to be fixed (and thus are transforming frame t into the coordinate system of frame $t+1$). Do not implement a 'break' step in your RANSAC algorithm that stops running the iterations/fits if it reaches the desired number of inliers.
  Use the following RANSAC coefficients:

```
numPtsToSample = 3;     % the minimum number of correspondences needed to fit a model
iterNum = 2000;         % number of iterations to run the RANSAC sampling loop
thDist = 1.5;           % inlier distance threshold; units are in pixels
randomseed = 2.8;       % use rng(2.8) to set your random seed!!!!!!!
```

You are provided with the following resources:

- As in the previous problem, you are given the left and right camera matrices P_{left} and P_{right}, which are $3x4$ projection matrices. Assume that these matrices are the ground truth projection matrices, which means that our only sources of error is in the measured points x in the left image and x' in the right image.
- You are given the function:
  `[matchedPoints1, matchedPoints2, matchedf1, matchedf2] = detectFeatureMatches(I1,I2)`
  that extracts SURF features from the images $I1$ and $I2$, and uses an exhaustive pairwise distance metric to find the 'matches'/correspondences between each images' extracted features. The outputs are the locations of the features for the first and second image, and the features for the first and second image, respectively, sorted such that `matchedPointsL(i,:)` is the correspondence of `matchedPointsR(i,:)`.
- You are given the function:
  `point3D = linear_triangulation(point_left, point_right, Pleft, Pright)`
  that takes in a point from the left image, `point_left`, a point from the right image, `point_right`, and the left and right camera projection matrices. It outputs the 3D point corresponding to `point_left` and `point_right`.