Assignment 3

Assignment Objective

In this assignment, our objective is to learn how to implement a simple version of a physics simulator with frictional contact interactions in the context of planar bouncing. Here, we are given a square object in the plane with known inertial and contact parameters. The objective is to predict the trajectory of the object given the initial state of the object.

Instructions

In this assignment, we will ask you to fill out missing pieces of code in a Python script. You will submit the completed code to Canvas. Your code is passed to an auto-grader that will verify the correctness of your work and assign you a score out of 100.

- Download the assignment script `assignment_3.py` from the Files menu of Canvas. There are also two supplementary pieces of code to download: `assignment_3_render.py` and `LCPSolve.py` – these scripts are used to solve LCPs and visualize the objects trajectories, you do NOT need to modify them.
- For each "Part" of the HW assignment, fill in the missing code as described in the problem statement.
- Submit the completed code to Canvas.
- The contribution of each part to your total score is noted in the subtitle.
- The assignment is due on October 18^{th}, 2021.
- You will need `numpy`, `cvxpy`, and `matplotlib` libraries.

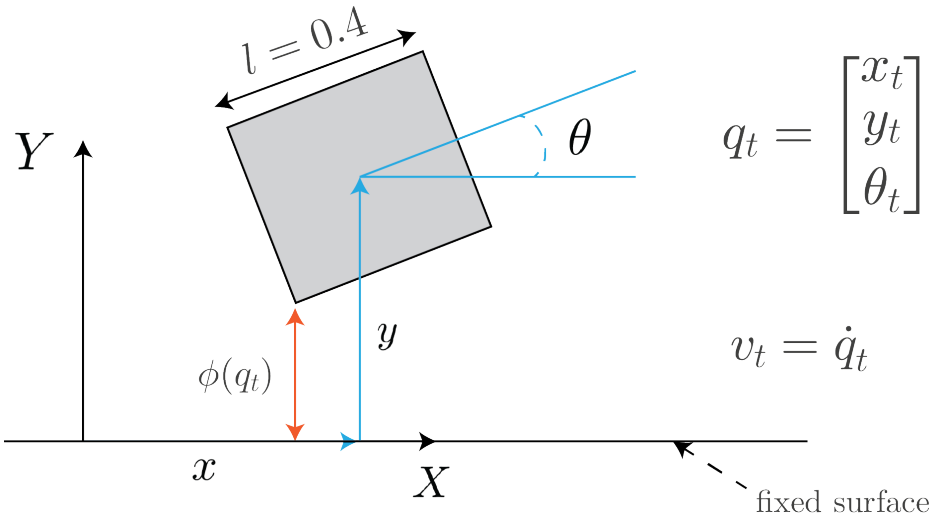

Figure 1. Square object undergoing planar dynamic frictional interaction with the surface.

Part 1 – Contact distance and Jacobian (30 points)

Consider the square object depicted in Fig. 1. This object is going to undergo frictional contact with the horizontal surface. The object can only make contact at its lowest corner, the corner closest to the surface. Given the object configuration $\boldsymbol{q}_t = (x_t, y_t, \theta_t)$, our goal is to write a function that calculates the

distance from the surface to the closest corner of the square $\phi(\boldsymbol{q}_t)$ and the contact Jacobian for this point $\mathbf{J}_c(\boldsymbol{q}_t) = [\boldsymbol{J}_t \ \boldsymbol{J}_n]_{3\times 2}$ where the first column is the tangential component of the Jacobian and the second is the normal component. Note the dependence of both on the configuration of the object and its geometry (length of it's sides).

To this end, complete the function `get_contacts` in `assignment_3.py` to return $\phi(\boldsymbol{q}_t)$ and $\mathbf{J}_c(\boldsymbol{q}_t)$:

$$\mathbf{J},\ \phi = \text{get\_contacts}(\boldsymbol{q}_t)$$

where $\boldsymbol{q}_t$ is a `(3, )` numpy array, ϕ is a scalar value and $\mathbf{J}$ is a `(3, 2)` numpy array.
Hint 1: Write your normal $\boldsymbol{n}$ and the location of the contact with respect to the center of mass $\boldsymbol{r}$ in the world frame. This way $\mathbf{J}$ is expressed in the world frame. This will make the following parts easier.
Hint 2: When finding the lowest point on the object (the one that actually makes contact), you can keep track of all 4 corners and find the one that is closest to the contact surface. Avoid writing conditional if/else statements, this will increase your chances of making a mistake.
Hint 3: Make sure that contact detection is working correctly for any of the candidate corner points. This will not be an issue if you follow hint 2. However, if you're using an if/else structure, there is a higher chance that you will make a mistake.

Part 2 – LCP Formulation (30 points)

If we assume that contact occurs, we know that we can solve for the contact forces by solving the following LCP:

$$\underbrace{\begin{bmatrix} \boldsymbol{J}_n^T\mathbf{M}^{-1}\boldsymbol{J}_n\Delta t & -\boldsymbol{J}_n^T\mathbf{M}^{-1}\boldsymbol{J}_t\Delta t & \boldsymbol{J}_n^T\mathbf{M}^{-1}\boldsymbol{J}_t\Delta t & 0 \\ -\boldsymbol{J}_t^T\mathbf{M}^{-1}\boldsymbol{J}_n\Delta t & \boldsymbol{J}_t^T\mathbf{M}^{-1}\boldsymbol{J}_t\Delta t & -\boldsymbol{J}_t^T\mathbf{M}^{-1}\boldsymbol{J}_t\Delta t & 1 \\ \boldsymbol{J}_t^T\mathbf{M}^{-1}\boldsymbol{J}_n\Delta t & -\boldsymbol{J}_t^T\mathbf{M}^{-1}\boldsymbol{J}_t\Delta t & \boldsymbol{J}_t^T\mathbf{M}^{-1}\boldsymbol{J}_t\Delta t & 1 \\ \mu & -1 & -1 & 0 \end{bmatrix}}_{\mathbf{V}} \begin{bmatrix} f_n \\ f_{t,1} \\ f_{t,2} \\ \lambda \end{bmatrix} + \underbrace{\begin{bmatrix} \boldsymbol{J}_n^T((1+\epsilon)\boldsymbol{v}_t + \Delta t\mathbf{M}^{-1}\boldsymbol{f}_e) \\ -\boldsymbol{J}_t^T((\boldsymbol{v}_t + \Delta t\mathbf{M}^{-1}\boldsymbol{f}_e) \\ \boldsymbol{J}_t^T(\boldsymbol{v}_t + \Delta t\mathbf{M}^{-1}\boldsymbol{f}_e) \\ 0 \end{bmatrix}}_{\boldsymbol{p}} \geq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} f_n \\ f_{t,1} \\ f_{t,2} \\ \lambda \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

where the coefficient of friction is $\mu = 0.3$, the coefficient of restitution is $\epsilon = 0.5$, and the inertia matrix is $\mathbf{M} = \text{diag}\{m, m, mr_g^2\}$ with $m = 0.3$, $r_g^2 = l^2/6$, and $l = 0.2$. We assume that gravity points in the negative y direction; i.e. $\boldsymbol{g} = (0, -9.81, 0)$ and that gravity is the only external force $\boldsymbol{f}_e$ (asides from contact) that is applied to the object. Assume that the time step length $\Delta t = 0.01$. These parameters are defined as global variables in the python script.

Complete the function `form_lcp` in `assignment_3.py` such that the inputs are $\mathbf{J} = \begin{bmatrix}\boldsymbol{J}_t & \boldsymbol{J}_n\end{bmatrix}$ (which you calculated in the previous part) and $\boldsymbol{v}$ (the velocity of the object) and returns the LCP matrix and vector pair $\mathbf{V}$ and $\boldsymbol{p}$:

$$\mathbf{V},\ \boldsymbol{p} = \text{form\_lcp}(\mathbf{J}, \boldsymbol{v})$$

where $\mathbf{V}$ is a `(4, 4)` numpy array and $\boldsymbol{p}$ is a `(4, )` numpy array. Note that we have provided the function `solve_LCP` that takes as input $\mathbf{V}$ and $\boldsymbol{p}$ and returns the contact forces as $\boldsymbol{f}_c = \begin{bmatrix} f_n & f_{t,1} & f_{t,2}\end{bmatrix}^T$ as a `(3, )` numpy array.
Hint 1: You will note that we can write:

$$\hat{\mathbf{J}}^T\mathbf{M}^{-1}\hat{\mathbf{J}}\Delta t = \begin{bmatrix} \boldsymbol{J}_n^T\mathbf{M}^{-1}\boldsymbol{J}_n\Delta t & -\boldsymbol{J}_n^T\mathbf{M}^{-1}\boldsymbol{J}_t\Delta t & \boldsymbol{J}_n^T\mathbf{M}^{-1}\boldsymbol{J}_t\Delta t \\ -\boldsymbol{J}_t^T\mathbf{M}^{-1}\boldsymbol{J}_n\Delta t & \boldsymbol{J}_t^T\mathbf{M}^{-1}\boldsymbol{J}_t\Delta t & -\boldsymbol{J}_t^T\mathbf{M}^{-1}\boldsymbol{J}_t\Delta t \\ \boldsymbol{J}_t^T\mathbf{M}^{-1}\boldsymbol{J}_n\Delta t & -\boldsymbol{J}_t^T\mathbf{M}^{-1}\boldsymbol{J}_t\Delta t & \boldsymbol{J}_t^T\mathbf{M}^{-1}\boldsymbol{J}_t\Delta t \end{bmatrix}, \quad \hat{\mathbf{J}} = \begin{bmatrix} \boldsymbol{J}_n \\ -\boldsymbol{J}_t \\ \boldsymbol{J}_t \end{bmatrix}$$

Try to use matrices and vectors with matrix multiplication to minimize your chances of making a mistake when forming $\mathbf{V}$ and $\boldsymbol{p}$.
Hint 2: Try to use `np.matmul` over `np.dot`, the former is the preferred function for matrix multiplication.
Hint 3: Think very carefully about $\boldsymbol{f}_e$ and how it relates to $\boldsymbol{g}$ (the gravitational acceleration).

Part 3 – Step the Simulation Forward (30 points)

In this part, we're going to write our dynamics solver. This solver takes as input the current configuration of the object and its velocity, and calculates the next time step configuration and velocity. The calculation of the next time step state depends on whether the object is in contact or not. We determine whether the object is in contact by checking if $\phi < \delta$. For this problem, assume $\delta = 0.001$. If the object is not in contact then we may write:

$$\begin{aligned} \boldsymbol{v}_{t+1} &= \boldsymbol{v}_t + \Delta t \mathbf{M}^{-1} \boldsymbol{f}_e \\ \boldsymbol{q}_{t+1} &= \boldsymbol{q}_t + \Delta t \boldsymbol{v}_{t+1} \end{aligned}$$

and if the object is in contact then we first call the `form_lcp` and `solve_LCP` functions to solve for the contact forces then update the states as:

$$\begin{aligned} \mathbf{V},\ \boldsymbol{p} &= \text{form\_lcp}(\mathbf{J}, \boldsymbol{v}) \\ \boldsymbol{f}_c &= \text{solve\_LCP}(\mathbf{V}, \boldsymbol{p}) \\ \boldsymbol{v}_{t+1} &= \boldsymbol{v}_t + \Delta t \mathbf{M}^{-1}(\boldsymbol{f}_e + \boldsymbol{J}_n f_n - \boldsymbol{J}_t f_{t,1} + \boldsymbol{J}_t f_{t,2}) \\ \boldsymbol{q}_{t+1} &= \boldsymbol{q}_t + \Delta t \boldsymbol{v}_{t+1} + \boldsymbol{q}_p \end{aligned}$$

where $\boldsymbol{f}_c = \begin{bmatrix} f_n & f_{t,1} & f_{t,2} \end{bmatrix}$ and $\boldsymbol{q}_p = (0, \delta, 0)$ is a correction to the pose of the object to account for possible penetration because of time-stepping. To this end, complete the function `step` in `assignment_3.py` where the inputs are the current time step configuration and velocity of the object $\boldsymbol{q}_t$ and $\boldsymbol{v}_t$ and the outputs are the next time step configuration and velocity:

$$\boldsymbol{q}_{t+1}, \boldsymbol{v}_{t+1} = \text{step}(\boldsymbol{q}_t, \boldsymbol{v}_t)$$

We will use this function to simulate the trajectory of the object.
Hint 1: It is okay if your algorithm allows a small amount of penetration at contact, as long as the object bounces back (and does not continue to penetrate the surface) then you will receive full marks.
Hint 2: If you have consistent penetration issues (object wanders into the surface and stays there), print out ϕ (the distance function) and the corner at which your algorithm thinks contact is happening to see if you can diagnose the issue.
Hint 3: Make sure that you are applying the external forces and contact forces correctly (in the correct order). If enough external force is not applied, then the object is not pushed away from the surface.

Part 4 – Simulation Trajectory (10 points)

Now that we have a function to step our dynamics forward in time, let's finish off the simulation by writing a loop to generate a trajectory. Complete the function `simulate` in `assignment_3.py` that takes as input the initial configuration and velocity of the object $\boldsymbol{q}_0$ and $\boldsymbol{v}_0$ and returns the trajectory of the object for 150 time steps:

$$\mathbf{q}, \mathbf{v} = \text{simulate}(\boldsymbol{q}_0, \boldsymbol{v}_0)$$

where $\mathbf{q}_{3\times150}$, $\mathbf{v}_{3\times150}$ numpy arrays and the first column of each matrix is the initial conditions of the trajectory. You may check your solution by passing $\mathbf{q}$ to the render function in `assignment_3.py`:

$$\text{render}(\mathbf{q})$$

If done correctly, the render function will visualize the trajectory of the block in an animated loop. You will need the `matplotlib` library installed for this functionality.