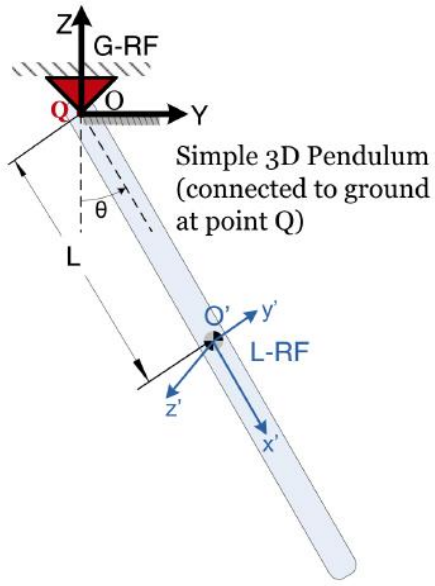
Fall 2019 ME751 Final Project Report
University of Wisconsin-Madison



simEngine3D
for Rigid Body Dynamics Simulation

Subodh Subedi

December 14, 2019

Abstract

Various interactions based on different kinematic constraints between two rigid bodies have been explored in this project. The project is build upon the assignments from the course, HW 6 and HW7 in particular. Both the approaches of r-w and r-p formulations have been used at different stages. Comparable results have been obtained for analysis using both types of formulation. The GitHub link for the repository contains all the codes and necessary readme documents.
A *input.txt* has been created. These files contain inputs for the simEngine3D that simulates the rigid body interaction between two bodies. Different basic level, intermediate level and high level geometric constraints have been coded and their results for position, velocity and acceleration as well as jacobian have been computed. The results are saved as a .txt file.
The work is more involved on the Revolute Joints from HW 6. For a pendulum with revolute joint with the ground, the position, velocity and acceleration have been obtained. The results can be viewed on a plot and/or a video of the simulation for the user defined time length. Results for concentrated forces and concentrated torques experienced by bodies connected by Translational spring damper actuator (TSDA) or rotational spring damper actuator (RSDA) have also computed. Inclusion of contact and friction was attempted but couldn't be completed. Part of the code for incorporating friction and contact has also been included in the GitHub repo.

Contents

1. General information

Home Department: Department of Mechanical Engineering
Status: Graduate Student (PhD)
Name: Subodh Subedi

Licensing Statement

2. Problem statement

The default project for the ME751 course was choosen. The following were the goals of the project

- Provide model definition through an input file (.txt or an excel file or any other formats)
- All types of joints and constraints discussed in class
- TSDA
- Friction and contact, implemented via penalty

The reason, this particular project was choosen was that it builds on the concepts learned in class as well as some of the problems from the HW. It attempts to mathematically defined different geometric constraints as well as solve the problems using the course material. Also, many of my HW submissions had bugs and the results weren't correct. This project helped me fix those bugs and build a better engine.
This project would help me in designing the suspension system of a FSAE race car, as it involved different types of joints. With a well build simEngine3D, it would allow better control over the mathematical definition of joints and constraints for the simulation of suspension system of FSAE race car.

3. Solution description

Part A:

Firstly, the low level constraints were defined (DP1, DP2, CD and D). Based on the constraint definition, flags, and ground status of the bodies fed in through the 'input_Constraints.txt' file to the simEngine3D_constraints, the values for following are obtained.

1. Value of the expression of the constraint
2. Right-hand side of the velocity equation
3. Right-hand side of the acceleration equation
4. Jacobian (Partial derivatives ϕ_p and ϕ_r)

These results were saved as a .txt file for each of the low level constraints (DP1, DP2, CD and D).

Implementation

Input definitions: 'input_Constraints.txt'

This .txt file contains all the information about attributes of the system, type of constraints, and flags defining the user requirements for displaying different output information. This also has the option to define a body as ground or unground.

Main Engine: simEngine3D_Constraints.m

This engine reads the input using readInput.m and saves the user input system attributes and requirement in the workspace. It then runs the different files defining different constraints listed below as per the user requirements

cons_dp1.m
cons_dp2.m
cons_d.m
cons_cd.m

These constraint functions use following matlab functions

1. Amatrix: Input: Euler Parameters (e0 (1 x 1), e(3 x 1)

   Output: A(3 x 3) matrix
2. Bmatrix: Input: p(4x1) is the Euler Parameter

   a (3x1) is the vector in local coordinate system

   Output: B(4 x 3) matrix
3. tilda: Input: a (3 x 1) vector

   Output: atilda (cross-product matrix(3 x 3) for the vector)
4. savefile: Saves results from computation for different geometric constraints

   Input: Results (array of all the computed results)
   Constraint (name of the goemetric constraint
   groundcondition (describes the status of body J)
   Output: File with all the computed results

Part B:

The intermediate level and high level geometric constraints (joints) were then defined. All of the defined joints have one of the body as ground. This can very well be changed in the code to define any other body. The following joints have been defined using the low-level constraints, driving functions and Euler Normalization constraints.

- Perpendicular 1 constraint: 2 DP1 Constraints
- Perpendicular 2 constraint: 2 DP2 Constraints
- Revolute Joint 2 DP1 and 3 CD Constraints
- Cylindrical Joint 2 DP1 and 2 DP2 Constraints
- Spherical Joint 3 CD Constraints
- Universal Joint 1 DP1 and 3 CD Constraints

Based on the user input for type of joint and constraints, results for position, velocity and acceleration are generated. The 'input.txt' file contains all the attributes of the sytem which, when fed into the

simEngine3D generates the results and saves them as per user requirement(in workspace, a plot, and/or video).

The work on Revolute joint is more involved. For a pendulum connected to the ground using a revolute joint, based on the user's requirement, both kinematic and inverse dynamic analysis can be carried out. The results can be saved as .txt file, a plot for the defined time interval and/or a video of the pendulum motion.
This part involved solving TSDA, RSDA. Translational spring-damper-actuator(TSDA) and Rotational Spring Damper Actuator (RSDA) systems were modeled. The inputs for the both the system could be fed in the 'input.txt' file and the *simEngine3D* would generate results for concentrated force and concentrated torques experienced by the bodies respectively.
An attempt was made to implement RSDA on the pendulum connected to the ground through a revolute joint. The code somehow wasn't able to take the concentrated torque experienced by the pendulum in the position, velocity and acceleration analysis. Thus, this integration of two system hasn't been included in the final work.

Implementation

Input Definition

The inputs for the two-body system is defined in the 'input.txt' file. This file contains all the information necessary to define different joints and user requirements for postprocessing of the results. Apart from defining the attributes for the system, user can define the type of joints, ground condition, time for simulation for a kinematic or dynamic analysis. Also, for post processing, user can choose output saved in workspace and/or plot and/or video, only for the Pendulum with a revolute joint.

Main engine: simEngine3D

This is the main engine. 'readInput.m' processes all the inputs from the user and then feeds them to the simEngine3D. The engine then calls on different functions based on the user's requirements defined in input.txt file to generate results for kinematic and dynamic analysis of system.
It uses following functions:

1. *Amatrix:* Input: Euler Parameters (e0 (1 x 1), e(3 x 1)

   Output: A(3 x 3) matrix
2. *Bmatrix:* Input: p(4x1) is the Euler Parameter

   a (3x1) is the vector in local coordinate system
   Output: B(4 x 3) matrix
3. *tilda:* Input: a (3 x 1) vector

   Output: atilda (cross-product matrix(3 x 3) for the vector)
4. *getp:* Input: Orientation matrix A (3 x 3)

   output: euler parameters p(4 x 1)
5. *getpnorm:* calculates Euler Parameter Normalization Contraints from p and pDot

   Input: p and pDot

Output: Euler parameter normalization constraints

6. *universalJoint*

   Takes input for attributes for a two-body system with one grounded and outputs position, velocity and acceleration for the other body

7. *cylindricalJoint*

   Takes input for attributes for a two-body system with one grounded and outputs position, velocity and acceleration for the other body

8. *perpendicular1*

   Takes input for attributes for a two-body system constrained with perpendicular 1 constraint and outputs position, velocity and acceleration for the other body

9. *perpendicular2*

   Takes input for attributes for a two-body system constrained with perpendicular 2 constraint and outputs position, velocity and acceleration for the other body

10. revoluteJoint

    Takes inputs for attributes for a pendulum defined in 'input.txt' and then carries out kinematic and inverse dynamic analysis. The results are then saved based on user requirements (image and/or video).

11. saveImage

    Plots results for position, velocity and acceleration and then saves them as an image

12. makeVideo

    Simulates the results for position, velocity and acceleration and then saves the results as a video.

13. TSDA

    Incorporates translational spring-damper-actuator system based on the user defined values for mass, spring stiffness, damping coefficient and actuator function. Computes the concentrated force experienced by the bodies connected by such a system. Plots the force variation with time and then saves the plot as an image.

    Expression of the TSDA force:

$$f^{TSDA}(i, \bar{\mathbf{s}}^P, j, \bar{\mathbf{s}}^Q, k, l_0, c, h, t) = k(l_{ij} - l_0) + c\dot{l}_{ij} + h(l_{ij}, \dot{l}_{ij}, t)$$

    where, h = `(theta0 + thetaDot*sin(2*pi*t));`

14. RSDA

    Incorporates Rotational spring-damper-actuator system based on the user defined values for mass, spring stiffness, damping coefficient and actuator function. Computes results for concentrated torque experienced by the bodies connected by such a system. The plot for torque variation with time is saved as an image.

    Expression of the RSDA torque:

$$n^{RSDA}(i, \bar{\mathbf{a}}_i, \bar{\mathbf{b}}_i, j, \bar{\mathbf{a}}_j, \bar{\mathbf{b}}_j, k, \theta_0, c, h, t) = k(\theta_{ij} - \theta_0) + c\dot{\theta}_{ij} + h(\theta_{ij}, \dot{\theta}_{ij}, t)$$

where, h = `(theta0 + thetaDot*sin(2*pi*t));`

Part C

Also, an attempt to model friction and contact using DEM-penalty approach has been made for simulating the granular motion. All the attributes for two bodies have been defined and few calculations have been made. After a lot of unsuccessful attempts, to calculate normal and tangential frictional force, it was decided to pursue the work in future.

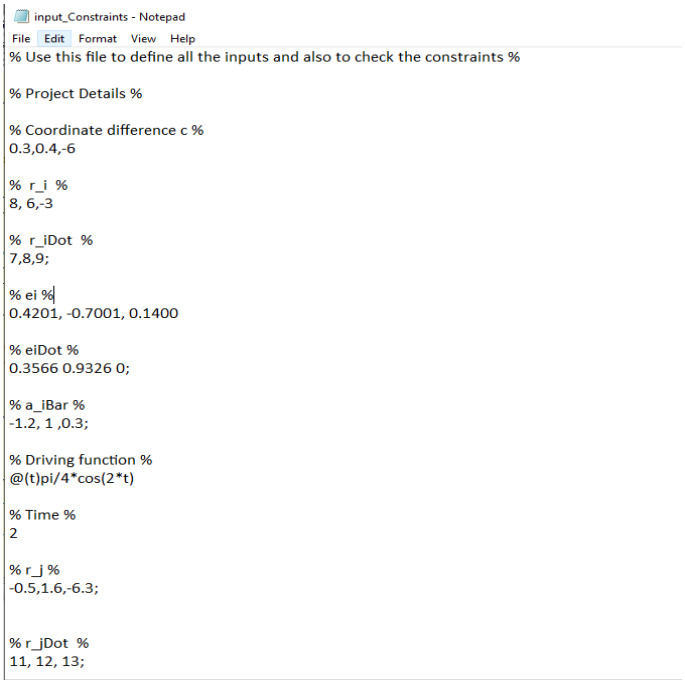
Implementation:

Friction.m : This file has the attributes of the granular particles under consideration. This could very well be incorporated using the earlier defined method of using a '.txt' file for input definition. The results for normal and tangential forces couldn't be computed because of some bug in the code. Thus, the code is still a work-in progress.

Indicate how you went about implementing your solution. Explain data structures, algorithms used, code structure, function you implemented, etc. Provide a panoramic snapshot of your Final Project effort.

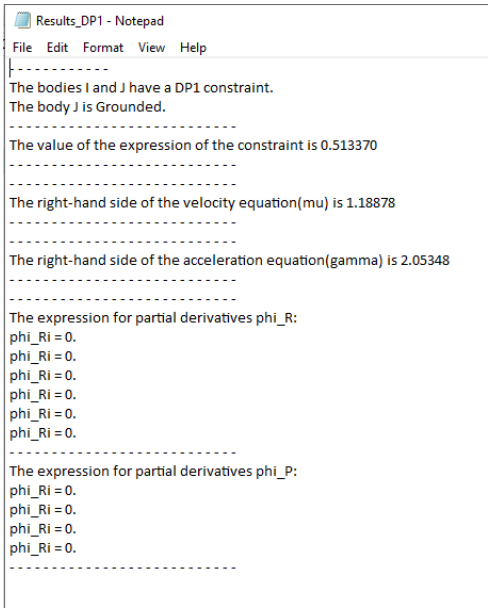
4. Overview of results. Demonstration of your project

1. Constraints: To compute the value of constraint, velocity, acceleration and jacobian for different constraints, use the following flowchart.
    a. Open **Project\Constraints** folder
    b. Define the attributes, ground condition, flags in the '**input_Constraints.txt**' file



    c. Open and run ***simiEngine3D_constraints.m***

d. This displays results on the command window as well as saves the computed results based on user input as a '.txt' file.



2. Joints and Pendulum: This is the main focus of the project.
    a. Open **Project\Joints** folder
    b. Input the requirements in ***'input.txt'*** file
    c. Run ***simEngine3D.m***

Based on the user input on the following lines in the '*input.txt*' file, various results are obtained. The way results are presented depend on the options choosen. Only for the simulation involving revolute joint on a pendulum described in the adjacent figure, user can generate **plot and/or video** for the position, velocity and acceleration.
For dynamic analysis on pendulum, user can generate a plot for the reaction torque.
For TSDA and RSDA systems, user can generate plots for variation of concentrated forces and concentrated torques experienced by bodies in the system and save them as images.

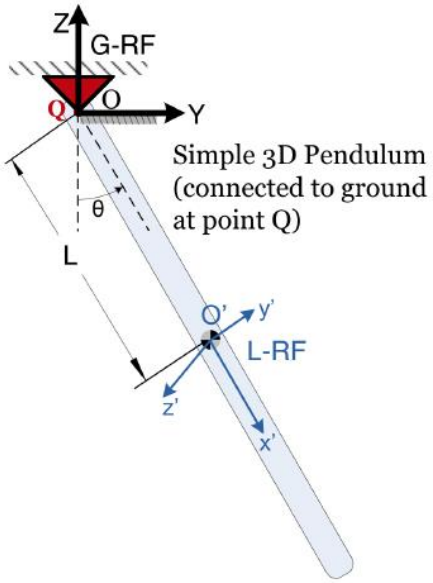


Figure 1: Pendulum with revolute joint with ground

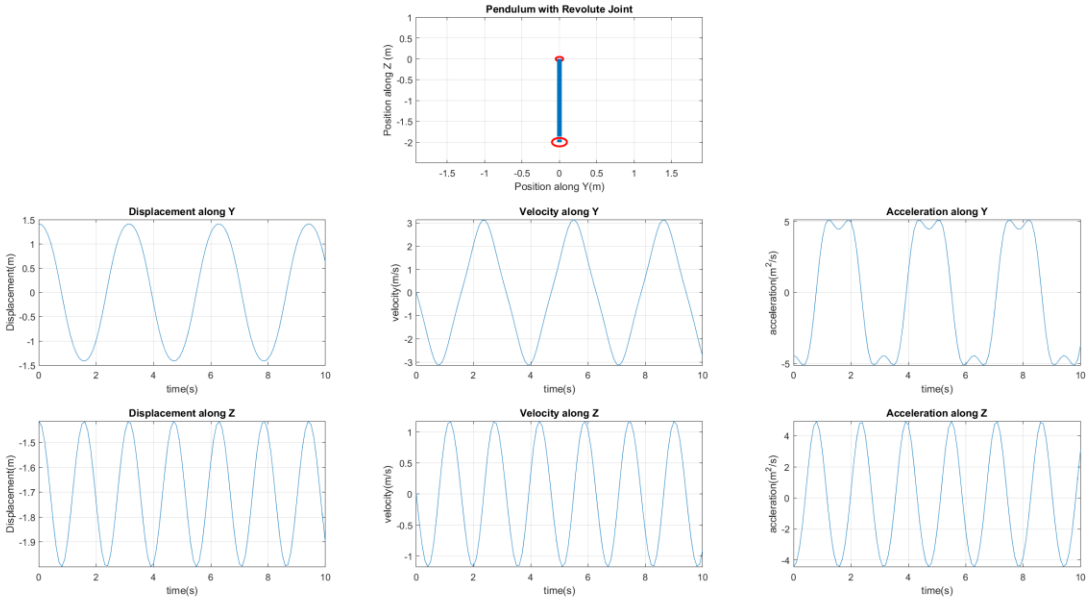


Figure 2: Plots for position, velocity and acceleration of a pendulum defined by a revolute joint with the ground.

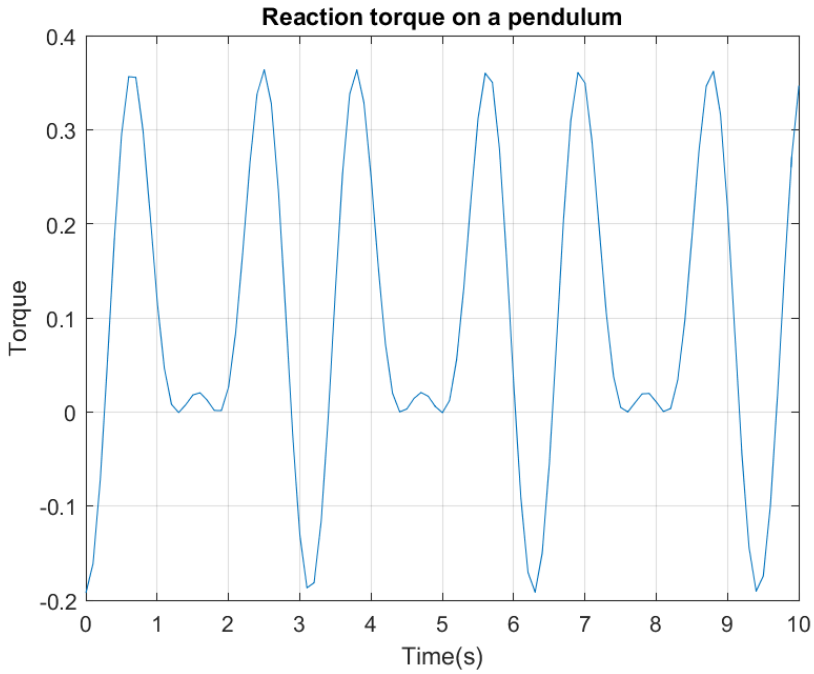


Figure 3: Reaction torque obtained from inverse dynamic analysis of Pendulum

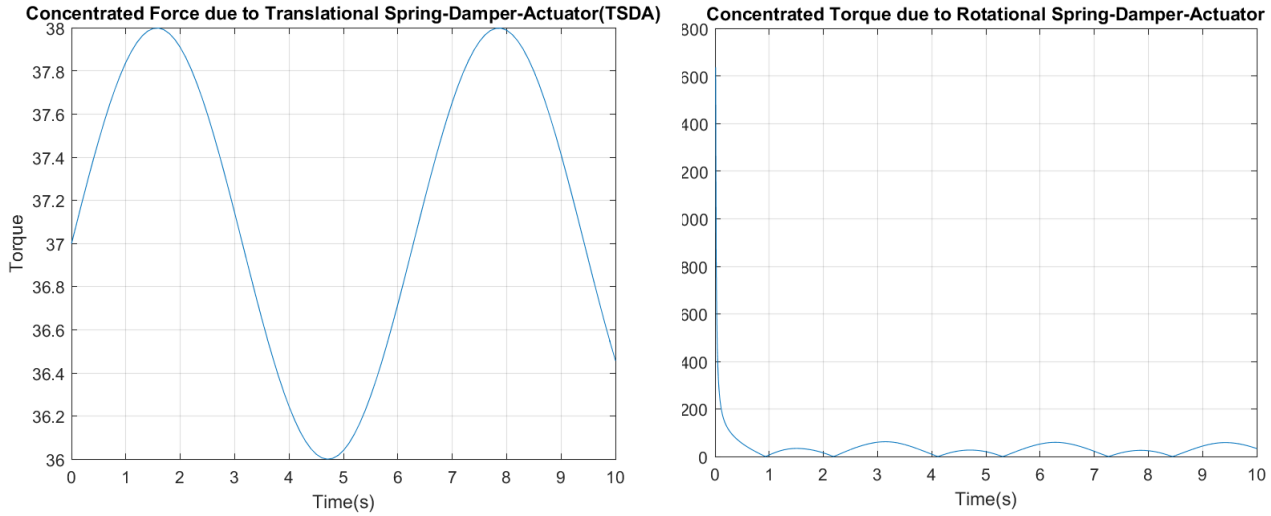


Figure 4: Variation of concentrated force and concentrated torque for TSDA and RSDA systems respectively.

The code for Friction and Contact isn't complete as I couldn't get the block diagonal matrix working. I get the qDot as a 3x1 vector whereas my q is a (14 x 1).

Explain here what you obtained, explained why the results are good/bad. This is the place where you talk about the outcomes of your Final Project effort. It is not the end of the world if your code doesn't work as anticipated. Explain here how far you have made it.

Most often, you have a comparison against sequential code, perhaps via a scaling analysis. Make sure you include plots and/or tables to show your results.

5. Deliverables:

1. simEngine_Constraints: Deals with low level constraints for computation of user defined outputs.

   (**Projects\Constraints**)
2. simEngine: Main engine that deals with all the intermediate constraints, high level constraints(joints). Focus mainly on the pendulum connected to the ground with a revolute joint. Carries out kinematic as well as inverse dynamic analysis. Generates results for the simulation of pendulum motion with change in time. Also, deals with TSDA and RSDA to compute results for concentrated force and torques experienced by the bodies under such system.
   (Projects\Joints)
3. Friction: Incomplete code to simulate motion of two granular bodies incorporating friction and contact between them via DEM-penalty approach.
   (Projects\Friction)

6. Conclusions and Future Work

A text file input system has been incorporated to define body attributes as well as user requirements for type of analysis and result postprocessing. All the geometric constraints have been implemented. Different joints have been defined mathematically and then based on attributes of bodies, kinematic and inverse dynamic analysis have been performed on them. A video has been generated that shows the time evolution of the pendulum motion.

RSDA and TSDA have been implemented for two bodies. An attempt has been made to incorporate friction and contact in bodies using DEM-Penalty method.

As incorporation of RSDA in the pendulum couldn't be made, it is proposed as a future work. Also, solving the friction and contact problem would be carried out in future.

This project is a great learning exercise for my goal of designing the suspension system of a FSAE race car. The simEngine build in this project works only for two body system. A future work would be to make it work for multi-body system with different types and number of joints to replicate a FSAE race car suspension system.

References

[1] All the formulations used here are referenced from the course slides ME571, Fall 2019.

Acknowledgement: The code for revolute joint for pendulum was build on Drithi Shetty's code for HW 6. Modifications were made on my own submission to fix the bugs and to obtain correct results.