Development of MATLAB-Based 3D Multibody Kinematics and Dynamics Simulation Engine

Michael F. Vignos

University of Wisconsin-Madison

# Table of Contents

# Introduction

# Problem Statement

The goal of this work was to develop a 3D multibody kinematics and dynamics simulation engine within MATLAB. The theory used to develop this simulation engine was taken from the following sources:

1. Haug, Edward J. *Computer Aided Kinematics and Dynamics of Mechanical Systems*. Boston: Allyn and Bacon, 1989. Print.
2. Negrut, Dan. *Mechanical Engineering 751: Advanced Computational Multibody Dynamics.* University of Wisconsin-Madison, Madison, WI. Fall 2016. Lecture.

# simEngine3D Framework

As previously mentioned, all code for this simulation engine was developed using MATLAB R2014b. It is likely that simEngine3D will perform properly for future versions of MATLAB as the functions used are not version specific. However, this has not been verified.

## Organization of GitHub Repository

The GitHub repository for this code is named simEngine3D-Vignos. Within this repository there are 3 main folders: *simEngine3DCode*, *testExamples*, and *ME751assignments*. The folder *simEngine3DCode* contains all of the MATLAB code that is used to run this simulation engine. The folder *testExamples* contains example driver files that were used to validate this simulation engine. These files contain examples of kinematics, dynamics, and inverse dynamics analyses. The results of these driver files were compared to results of either other validated simulation packages, the results reported in *Computer Aided Kinematics and Dynamics of Mechanical Systems* (1), or analytical results computed by hand for validation.

## Object-Oriented Programming

The framework of simEngine3D was written using object-oriented programming. Using object-oriented programming allows for the creation of MATLAB classes, which can be leveraged by the user to develop a multibody system in a hierarchical manner (Fig. 1). The classes that exist in this code are contained within the folder *simEngine3DCode* and are as follows: *multibodySystem.m, body.m, CDconstraint.m, Dconstraint.m, DP1constraint.m, DP2constraint.m, and simEngine3DUtilities.m*. In addition to these classes, there is also a *plot* folder that contains various functions that can be used to display and animated the multibody system. The purpose of each class and the *plot* commands will be covered in a bit more detail in the following sections.

### multibodySystem.m

An instance of this class contains all of the bodies, constraints, and externally applied forces and torques that are used to perform a simulation. This class also contains all of the methods needed to perform kinematics, inverse dynamics, and dynamics analyses. Following completion of a simulation, the instance of this class used to define the multibody system will contain all of the data stored throughout the simulation (e.g. kinematics of each body, reaction forces and torques, etc.). Additionally, since the these data are stored at each time step, a simulation can be stopped prematurely and the data contained within the instance of this class can be visualized to see what was occurring with the simulation up to the stopping time.

### body.m

An instance of this class contains all of the attributes of a body (e.g. mass, body number, kinematics, etc.) that are used when performing a simulation. This class also contains functions that are commonly used to compute variables related to a single body (e.g. the orientation matrix of a body, the total torque applied to a body, etc.). Additionally, the state information throughout a simulation is stored within a body class. This approach of having each body in the system defined as its own class allows for improved organization and makes it easier to compute state information for a single body in post-processing.

### Basic Constraint Classes: CDconstraint.m, Dconstraint.m, DP1constraint.m, and DP2constraint.m

Instances of each of these classes are similar in that they define the bodies impacted by each constraint, they contain the attributes of each of the four basic constraints, and they contain functions used to compute attributes of these constraints that are needed when performing a simulation (e.g. current state of the constraint, the right hand side of the acceleration equation, partial derivatives of the constraint, etc.). All higher level constraints (i.e. joints) used in defining a multibody system are composed of instances of these basic constraints.

### simEngine3DUtilities.m

This class is simply a collection of functions that are commonly used when performing a simulation (e.g. computing the distance between two points, computing a skew symmetric matrix from a vector, etc.).

### plot Folder

This folder contains a collection of functions that can be used to display the position of all bodies in a system at a specific state or to create an animation to visualize the output of a simulation.

## Example Model Definition

Figure 2 below contains a screen shot of an example model definition within a driver script. This model is a simple pendulum with two bodies (the mass and the ground) and a revolute joint defined between them. The revolute joint is actually composed of 5 basic constraints. This concept of defining joints using basic constraints will be further discussed in the following section. More thorough examples of model definitions can be found in the *testExamples* folder in the *simEngine3D-Vignos* repository.

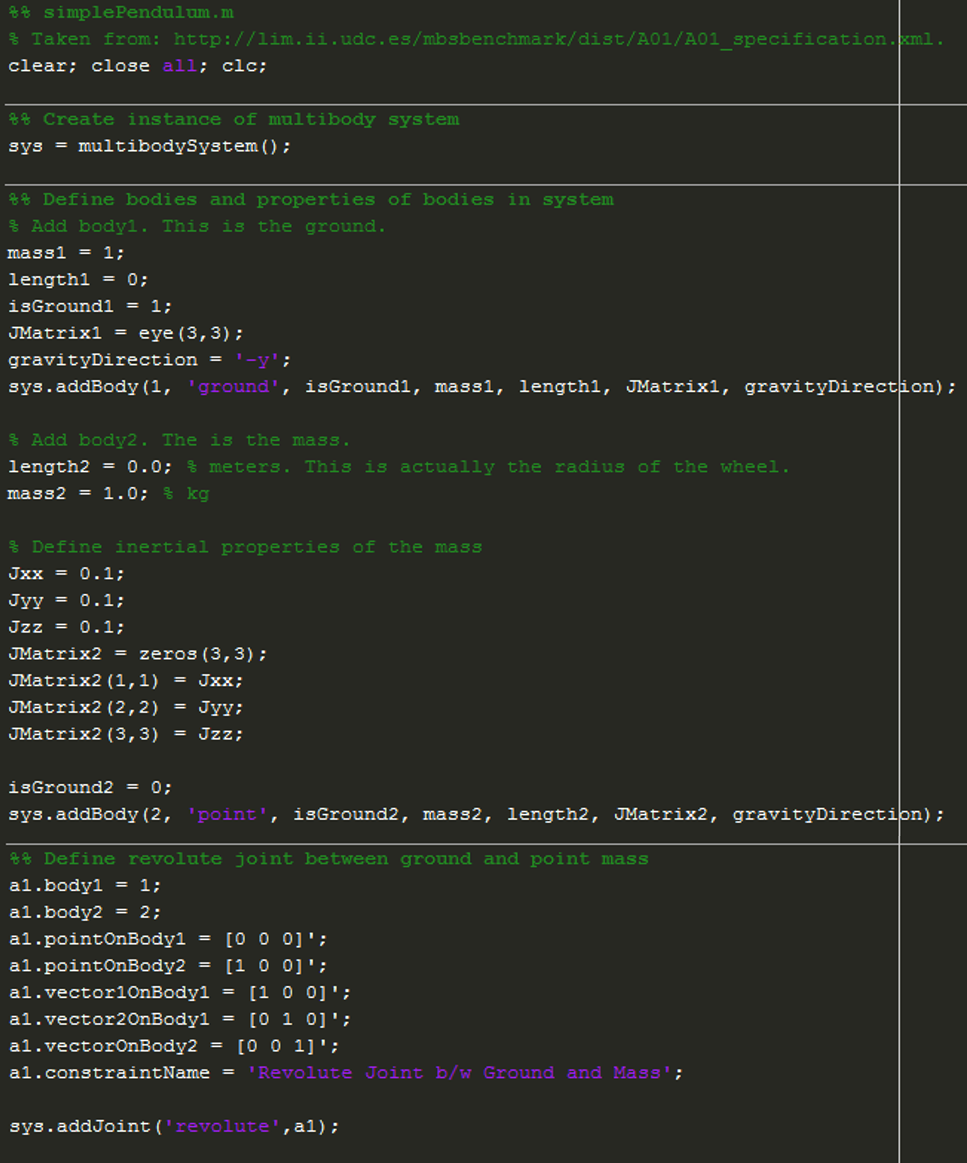


Figure : Example driver file showing the definition of a simple pendulum. This example contains two bodies (the ground and a point mass) and a revolute joint between them.

# System Constraints

## Kinematic Constraints

Kinematic constraints are constraints that are added into a multibody system to define the geometry of the motion of the system. In other words, kinematic constraints between two bodies define the relative motion that is allowed between these bodies (i.e. they remove degrees of freedom of each body). Within simEngine3D there are three different levels of constraints: basic constraints, intermediate constraints, and joints. Both intermediate constraints and joints are made up of a collection of basic constraints. When developing a model, it is most common to define constraints using joints, as these

### Basic Constraints

### Intermediate Constraints

### Joints

## Driving Constraints

# Externally Applied Forces and Torques

## Translational-Spring-Damper-Actuators

# Analysis of Mechanisms

## Kinematics Analysis

## Dynamics Analysis

### Methods of Computing Iteration Matrix

## Inverse Dynamics Analysis

## Assembly Analysis

## Prescribing Initial Velocities

# Validation Efforts

## Validation of Joints

## Comparison to Benchmark Problems

# 

Things to do still:

~~Implement simple pendulum~~

~~Implement N-bar mechanism~~

~~Implement method to prescribe initial velocities~~

~~Implement example to validate a cylindrical joint (could maybe be a simple pendulum with a cylindrical joint instead of a revolute joint??)~~

Implement method to prescribe angular velocity at a joint, rather than using a DP1 constraint. A DP1 constraint is plagued with too many singularity issue.

~~Implement method to remove redundant constraints??~~ Not sure how robust my method is.

Implement ability to read model parameters from a file???

~~Flyball governor mechanism?~~

Bricard’s mechanism?

Andrew’s mechanism?

Things done:

~~Implemented ability to prescribe all basic constraints, all intermediate constraints, and all joints discussed in class.~~

~~Implemented ability to perform kinematics, inverse dynamics, and dynamics analyses.~~

~~Validated all joints~~

~~Validated ability to prescribe constant torque  
Implemented kinematics, inverse dynamics, and dynamics analyses~~

~~Implemented different methods of computing iteration matrix~~

~~Implemented various test cases validated by book and by group that created the benchmark problems~~

~~Implemented method to prescribe initial velocities for dynamics analysis.~~