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

Michael F. Vignos

University of Wisconsin-Madison

# Table of Contents

# Introduction

Include a sentence or two about the purpose of this document

State that the purpose of this document is not to go into a very detailed, technical description of theory used to develop this simulation engine, but rather to give the reader a general understanding of how to use this engine, the capabilities and features of this engine, and an overview of the efforts to validate this simulation engine.

# 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.

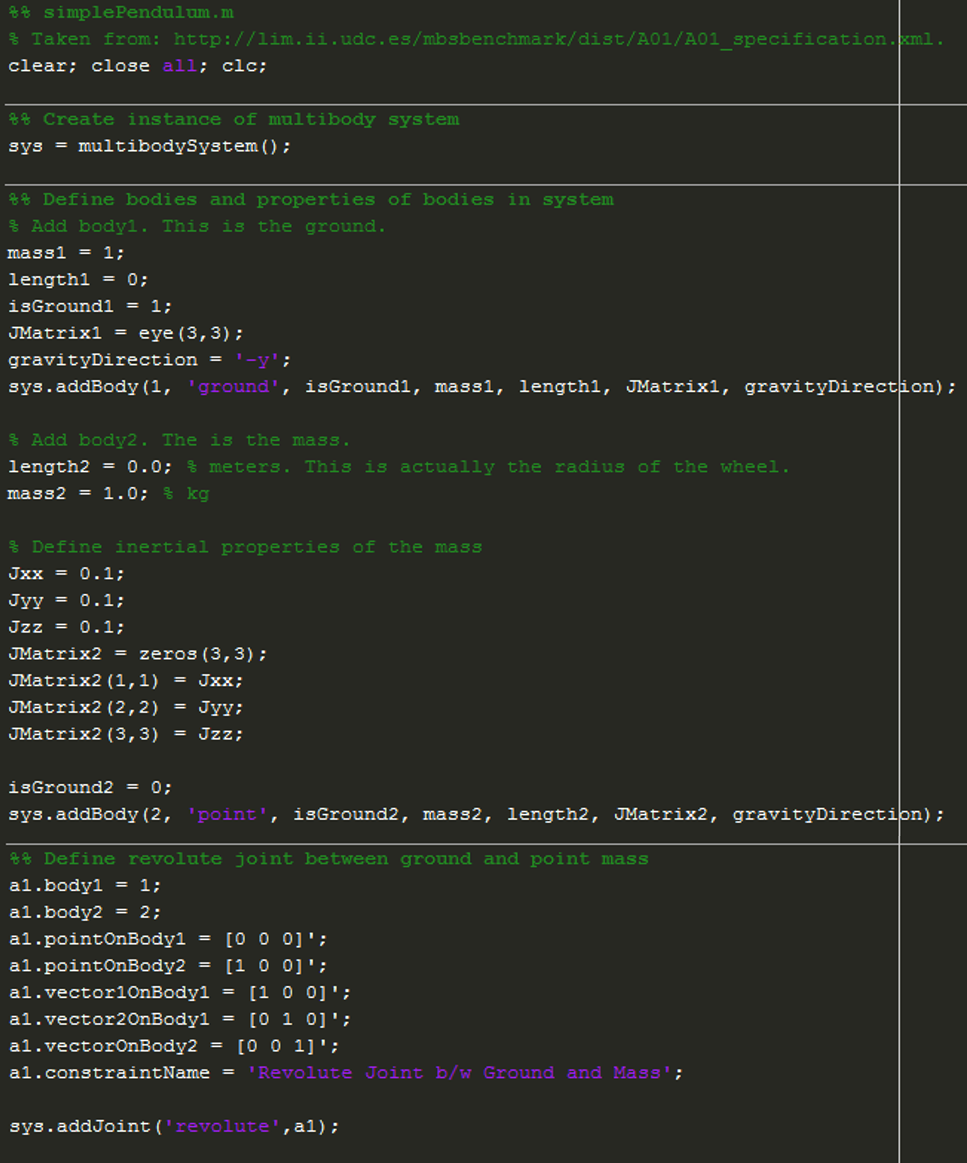


Figure 1: Example driver file showing the definition of a simple pendulum. This example contains two bodies (the ground and a point mass) and a revolute joint between them.

# 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 between bodies using joints, as the definition of these joints are most easily understood.

### Basic Constraints

As previously discussed, there are four basic constraints that are used as the foundation to all kinematic constraints in simEngine3D. These constraints are a coordinate distance constraint (CD constraint), a distance constraint (D constraint), a dot product one constraint (DP1 constraint), and a dot product two constraint (DP2 constraint). Each of these basic constraints removes one degree of freedom from a multibody system when it is added into the system.

#### Coordinate Difference (CD) Constraint

A CD constraint defines the distance between two bodies for a single Cartesian coordinate (x, y, or z). The attributes for this constraint are as follows:

1. The two bodies to which the constraint is applied (i.e. body 1 and body 2)
2. A point attached to body 1 in the body 1 reference frame
3. A point v body 2 in the body 2 reference frame
4. The Cartesian coordinate that is being constrained
5. The prescribed distance between the coordinates of the two bodies

#### Distance (D) Constraint

A D constraint defines the distance between two bodies. The attributes for this constraint are as follows:

1. The two bodies to which the constraint is applied (i.e. body 1 and body 2)
2. A point attached to body 1 in the body 1 reference frame
3. A point attached to body 2 in the body 2 reference frame
4. The prescribed squared distance between the two bodies. It is important to note that in the implementation of this constraint the user must provide the square of the distance.

#### Dot Product One (DP1) Constraint

A DP1 constraint constrains the dot product between two vectors, one attached to body 1 and one attached to body 2, to be equal to a prescribed function. This function is commonly set to zero, which constrains the two vectors to be orthogonal. The attributes for this constraint are as follows:

1. The two bodies to which the constraint is applied (i.e. body 1 and body 2)
2. A vector attached to body 1 in the body 1 reference frame
3. A vector attached to body 2 in the body 2 reference frame
4. A function prescribing the value of the dot product between the two vectors. If these vectors are defined to be orthogonal, this function must be zero.

#### Dot Product Two (DP2) Constraint

Similar to a DP1 constraint, a DP2 constraint constrains the dot product between two vectors to be equal to a prescribed function. However, in this case one vector is attached to body 1 and the second vector is defined as the vector between two points. One of these points is attached to body 1 and the other is attached to body 2. The attributes for this constraint are as follows:

1. The two bodies to which the constraint is applied (i.e. body 1 and body 2)
2. A vector attached to body 1 in the body 1 reference frame
3. A point attached to body 1 in the body 1 reference frame
4. A point attached to body 2 in the body 2 reference frame
5. A function prescribing the value of the dot product between the two vectors. If these vectors are defined to be orthogonal, this function must be zero.

### Intermediate Constraints

In simEngine3D there are two intermediate constraints: perpendicular one (B1) constraint and perpendicular two (B2) constraint. Both of these constraints are composed of two basic constraints.

#### Perpendicular One (B1) Constraint

A B1 constraint constrains a vector attached to body 2 to be orthogonal to a plane attached to body 1. The plane on body 1 is defined by two vectors attached to body 1. The attributes for this constraint are as follows:

1. The two bodies to which the constraint is applied (i.e. body 1 and body 2)
2. A vector attached to body 1 in the body 1 reference frame
3. Another vector attached to body 1 in the body 1 reference frame. This vector and the previous vector define the plane on body 1.
4. A vector attached to body 2 in the body 2 reference frame

#### Perpendicular Two (B2) Constraint

Similar to a B1 constraint, a B2 constraint constrains a vector to be orthogonal to a plane attached to body 1. However, in this case the vector is defined as the vector between a point attached to body 1 and a point attached to body 2. The attributes for this constraint are as follows:

1. The two bodies to which the constraint is applied (i.e. body 1 and body 2)
2. A vector attached to body 1 in the body 1 reference frame
3. Another vector attached to body 1 in the body 1 reference frame. This vector and the previous vector define the plane on body 1.
4. A point attached to body 1 in the body 1 reference frame
5. A point attached to body 2 in the body 2 reference frame

### Joints

To date, there are six joints that have been implemented and validated in simEngine3D. Each of these joints are constructed from a set of basic constraints. Below is a description of the attributes needed to define each of these joints. However, to gain a better understanding of how the basic constraints are used to create each of these joints, please see the methods in the *multibodySystem* class used to create each of these joints (e.g. *createSphericalJoint(), createCylindricalJoint(),* etc.)

#### Spherical Joint Attributes

1. The two bodies connected by this joint (i.e. body 1 and body 2)
2. A point attached to body 1 in the body 1 reference frame.
3. A point attached to body 2 in the body 2 reference frame. This second point will be constrained to be in the same location as the point defined in 2.

#### Revolute Joint Attributes

1. The two bodies connected by this joint (i.e. body 1 and body 2)
2. A point attached to body 1 in the body 1 reference frame.
3. A point attached to body 2 in the body 2 reference frame. This second point will be constrained to be in the same location as the point defined in 2.
4. A vector attached to body 1 in the body 1 reference frame. This vector is the 1st vector used to define the plane in which the revolute joint moves.
5. A second vector attached to body 1 in the body 1 reference frame. This vector is the 2nd vector used to define the plane in which the revolute joint moves.
6. A vector attached to body 2 in the body 2 reference frame. This vector is orthogonal to the plane on body 1 (i.e. the plane defined by the vectors defined in 4 and 5). In other words, this is the vector about which the revolute joint rotates.

#### Cylindrical Joint Attributes

1. The two bodies connected by this joint (i.e. body 1 and body 2)
2. A point attached to body 1 in the body 1 reference frame that is along the translational axis of body 1.
3. A point attached to body 2 in the body 2 reference frame that is along the translational axis of body 2.
4. A vector attached to body 1 in the body 1 reference frame. This vector is the 1st vector used to define the plane orthogonal to the translational axis of this joint.
5. A second vector attached to body 1 in the body 1 reference frame. This vector is the 2nd vector used to define the plane orthogonal to the translational axis of this joint.
6. A vector attached to body 2 in the body 2 reference frame. This vector is orthogonal to the plane on body 1 (i.e. the plane defined by the vectors defined in 4 and 5).

#### Translational Joint Attributes

1. The two bodies connected by this joint (i.e. body 1 and body 2).
2. A point attached to body 1 in the body 1 reference frame that is along the translational axis of body 1.
3. A point attached to body 2 in the body 2 reference frame that is along the translational axis of body 2.
4. A vector attached to body 1 in the body 1 reference frame. This vector is the 1st vector used to define the plane orthogonal to the translational axis of this joint.
5. A second vector attached to body 1 in the body 1 reference frame. This vector is the 2nd vector used to define the plane orthogonal to the translational axis of this joint.
6. A vector attached to body 2 in the body 2 reference frame. This vector is orthogonal to the plane on body 1 (i.e. the plane defined by the vectors defined in 4 and 5).
7. A second vector attached to body 2 in the body 2 reference frame. This vector is parallel to the plane on body 1 (i.e. the plane defined by the vectors defined in 4 and 5) and perpendicular to the vector defined in 4.

#### Universal Joint Attributes

1. The two bodies connected by this joint (i.e. body 1 and body 2)
2. A point attached to body 1 in the body 1 reference frame
3. A point attached to body 2 in the body 2 reference frame. This second point will be constrained to be in the same location as the point defined in 2.
4. A vector attached to body 1 in the body 1 reference frame that defines the rotation axis of this joint for body 1.
5. A vector attached to body 2 in the body 2 reference frame that defines the rotation axis of this joint for body 2. This vector is orthogonal to the vector defined in 4.

#### Revolute-Cylindrical Joint Attributes

Note: When defining a revolute-cylindrical joint, the two origins of the bodies that are connected by this joint typically also need to be constrained to one another using a distance constraint.

1. The two bodies connected by this joint (i.e. body 1 and body 2)
2. A point attached to body 1 in the body 1 reference frame that is along the translational axis of body 1.
3. A point attached to body 2 in the body 2 reference frame that is along the translational axis of body 2.
4. A vector attached to body 1 in the body 1 reference frame that defines the rotational axis of the revolute portion of the joint.
5. A vector attached to body 2 in the body 2 reference frame that is along the translational axis of this joint. This vector is orthogonal to the vector defined in 4.
6. A vector attached to body 2 in the body 2 reference frame. This vector is the 1st vector used to define the plane orthogonal to the translational axis of this joint. This vector is also orthogonal to the vector defined in 5.
7. A second vector attached to body 2 in the body 2 reference frame. This vector is the 2nd vector used to define the plane orthogonal to the translational axis of this joint. This vector is also orthogonal to the vector defined in 5.

## Driving Constraints

A driving constraint is used to specify the time-dependent behavior of one of the bodies in the multibody system. To date, all driving constraints in simEngine3D are prescribed by providing a time-dependent function for a basic constraint. As described in the Basic Constraint section, each basic constraint has a function as an input that defines the value of the constraint. By setting this input to be a time-dependent function, a basic constraint can be used as a driving constraint. For example, a DP1 constraint can be used to drive a revolute joint by prescribing the angle with respect to time of the free degree of freedom of this joint.

# Externally Applied Forces and Torques

## Constant Forces and Torques

## Variable 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.~~