MATLAB Assignment 3

October 2, 2014

Due: October 9, 2014

When working on this assignment you might want to take a look at MATLAB code that was developed by students who took ME451 in previous years. The students back then did not come up with identical solutions. Take a look at their solutions and develop your own.
2010: `http://sbel.wisc.edu/Courses/ME451/2010/SimEngine2D/index.htm`
2011: `http://sbel.wisc.edu/Courses/ME451/2011/SimEngine2D/index.htm`

Turning in your assignment: place all your files in a directory called ”lastName_Matlab_03”, zip that directory, and upload the resulting file ”lastName_Matlab_03.zip” in the appropriate Dropbox Folder at Learn@UW.

Problem 1. Extend the MATLAB code you wrote for the previous assignment so that it can also parse information for the following types of constraints: *RelativeX*, *RelativeY*, *RelativeDistance*, *RevoluteJoint*, *TranslationalJoint*. At the end of this assignment you should be able to parse the file model3.adm that is available online at the class webpage. The grader will test your code just like in the second MATLAB assignment, by issuing the command

```
>> runme(bodyID, constraintID);
```

Notes:

- A motion can be prescribed for a **revolute joint** indicating how the two bodies move relative to each other – this will be discussed in class in three lectures from now. Therefore, we need to provision for an attribute `fun` for the *RevoluteJoint* constraint type. However, specifying a motion on a revolution joint is not required. If no motion is to be applied to a joint, the value `"NONE"` should be used to indicate this situation. Any other string passed as the value for the property `fun` should be interpreted as a function of time and handled as such. To interpret it the right way, recall the very first MATLAB assignment where you had to read in a function of time and evaluate it at various values of the time t
- When parsing a **translational joint**, for simplicity make sure you define your $\vec{\mathbf{v}}_i$ and $\vec{\mathbf{v}}_j$ vectors such that $\|\vec{\mathbf{v}}_i\| = \|\vec{\mathbf{v}}_j\| = 1$. For added flexibility in modelling, you might want to always normalize them upon reading the vector in to accept situations, like in the JSON snippet below, where `vP1`$= [1\ 1]^T$ is not a unit vector
- As for the *RevoluteJoint*, a constraint of type *TranslationalJoint* may or may not have a motion prescribed. In the example below, the `"NONE"` keyword was used, indicating that no motion is to be applied to this joint. Your code must properly deal with both situations; i.e., a function is provided or not.
- In all constraint definitions, points associated with a body, e.g., the attribute `sP1` for an *RevoluteJoint* constraint, are assumed to be given in the LRF associated with that body. Likewise, points on the ”ground”, e.g., the attribute `Pground` for an *AbsoluteDistance* constraint, are assumed to be provided in the GRF.

- The JSON snippets below contain only samples of the JSON objects for the respective constraints. In an ADM file, they must be included as elements in the array value for the `"constraints"` key. See the `model3.adm` file provided on the course website.

```
{
    "name": "rel_x",
    "id": 4,
    "type": "RelativeX",
    "body1":  1,
    "sP1": [0.1, 0.4],
    "body2":  2,
    "sP2": [0.4, -0.3],
    "fun": "2.0"
}

{
    "name": "rel_y",
    "id": 7,
    "type": "RelativeY",
    "body1":  2,
    "sP1": [-0.4, 0.4],
    "body2":  3,
    "sP2": [-0.4, 0.3],
    "fun": "1.0"
}

{
    "name": "rel_dist",
    "id": 21,
    "type": "RelativeDistance",
    "body1": 1,
    "sP1": [1.2, -1.3],
    "body2": 4,
    "sP2": [4.1, 10.2],
    "fun": "2.3"
}

{
    "name": "rev_joint",
    "id": 3,
    "type": "RevoluteJoint",
    "body1": 1,
    "sP1": [2, 0],
    "body2": 2,
    "sP2": [-3, 0],
    "fun": "NONE"
}

{
    "name": "trans_joint",
    "id": 9,
    "type": "TranslationalJoint",
    "body1": 2,
    "sP1": [0.6, -3.1],
    "vP1": [1, 1],
    "body2": 3,
    "sP2": [0.3, -4.1],
    "vP2": [0, 1],
    "fun": "NONE"
}
```