KINEMATICS OF MULTIBODY SYSTEM

Ahmed Yesuf Nurye
ahmed.nurye.stud@pw.edu.pl

2024-12-01

ABSTRACT

To address scalability challenges in joint-coordinate methods for complex mechanisms, this project developed an alternative approach based on absolute coordinates, localizing constraint equations to individual joints. The solver employs the Newton-Raphson method to solve the nonlinear constraint equations for position and uses the Jacobian matrix to compute velocities and accelerations. This method is scalable, computationally efficient, and supports general planar mechanisms with an arbitrary number of links.

Contents

List of Figures **III**

List of Tables **III**

Task **1**

1 Introduction **2**

1.1 Objective 2

1.2 Theoretical Background 2

1.2.1 Absolute coordinates 2

1.2.2 Constraints 2

1.2.3 Jacobian 4

1.2.4 Kinematic problem 4

1.2.5 Marker points kinematics 5

1.2.6 Singularity 6

2 Method **7**

2.1 Analysis in Adams 7

2.2 Analysis in *MATLAB* 8

2.2.1 Coordinate frames assignment 8

2.2.2 Implementation description 9

3 Simulation Results and Comparison **13**

4 Conclusions **23**

A Test Mechanism **24**

A.0.1 Simulation results of the test mechanism 24

List of Figures

1 The mechanism 1
2 Coordinates 1
3 Revolute joint [1]. 3
4 Translational joint [1]. 4
5 A point on a body. 5
6 Mechanism structure implemented in *Adams* environment. 7
7 Local reference frames assignment. 8
8 Position comparison. 13
8a X position from *MATLAB* 13
8b X position from *Adams* 13
8c Y position from *MATLAB* 13
8d Y position from *Adams* 13
9 Velocity comparison. 14
9a VX velocity from *MATLAB* 14
9b VX velocity from *Adams* 14
9c VY velocity from *MATLAB* 14
9d VY velocity from *Adams* 14
10 Acceleration comparison. 15
10a AX acceleration from *MATLAB* 15
10b AX acceleration from *Adams* 15
10c AY acceleration from *MATLAB* 15
10d AY acceleration from *Adams* 15
11 Comparison of angular velocity and acceleration for *bodyC10*. 16
11a ω_z from *MATLAB* 16
11b ω_z from *Adams* 16
11c α_z from *MATLAB* 16
11d α_z from *Adams* 16
12 Position comparison for marker *K*. 18
12a X position from *MATLAB* 18
12b X position from *Adams* 18
12c Y position from *MATLAB* 18

12d Y position from *Adams* . . . 18
13 Velocity comparison for marker *K*. . . . 19
13a VX velocity from *MATLAB* . . . 19
13b VX velocity from *Adams* . . . 19
13c VY velocity from *MATLAB* . . . 19
13d VY velocity from *Adams* . . . 19
14 Acceleration comparison for marker *K*. . . . 20
14a AX acceleration from *MATLAB* . . . 20
14b AX acceleration from *Adams* . . . 20
14c AY acceleration from *MATLAB* . . . 20
14d AY acceleration from *Adams* . . . 20
15 Comparison of angular velocity and acceleration for marker *K*. . . . 21
15a ω_z from *MATLAB* . . . 21
15b ω_z from *Adams* . . . 21
15c α_z from *MATLAB* . . . 21
15d α_z from *Adams* . . . 21
16 Model in *Adams* of the test mechanism. . . . 24
17 Body 2 position along axis *X*. . . . 25
17a Result from *MATLAB* implementation. . . . 25
17b Result from *Adams* simulation. . . . 25
18 Body 2 velocity along axis *X*. . . . 25
18a Result from *MATLAB* implementation. . . . 25
18b Result from *Adams* simulation. . . . 25
19 Body 2 acceleration along axis *X*. . . . 26
19a Result from *MATLAB* implementation. . . . 26
19b Result from *Adams* simulation. . . . 26

List of Tables

1 Quantitative performance of the implemented system for *bodyC10*. . . . 17
2 Quantitative performance of the implemented system for marker *K*. . . . 22
3 *RMSE* performance of the implemented system tested against the results from *Adams*. 26

Task

1. Prepare ADAMS model of the mechanism shown in the Figure and perform kinematic analysis.

2. Write a computer program for kinematic analysis of this mechanism. The program should be written and tested in MATLAB environment. The mechanism must be described in absolute coordinates.

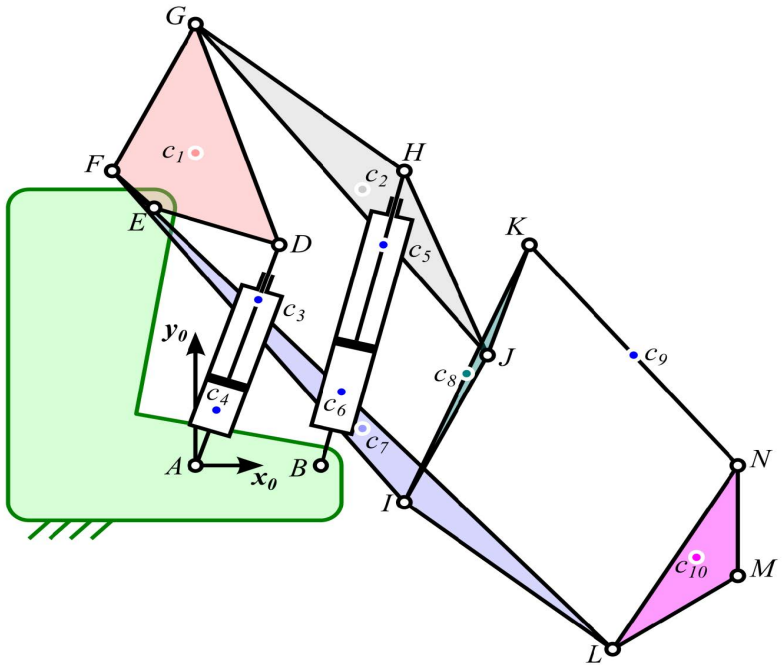


Figure 1: The mechanism

Table 1. Global coordinates of characteristic points (initial configuration)

	A	*B*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*
x [m]	0	0.3	0.2	–0.1	–0.2	0	0.5	0.5	0.7	0.8	1	1.3	1.3
y [m]	0	0	0.6	0.7	0.8	1.2	0.8	–0.1	0.3	0.6	–0.5	–0.3	0

Table 2. Global coordinates of centres of mass (initial configuration)

	*c*1	*c*2	*c*3	*c*4	*c*5	*c*6	*c*7	*c*8	*c*9	*c*10
x [m]	0	0.4	0.15	0.05	0.45	0.35	0.4	0.65	1.05	1.2
y [m]	0.85	0.75	0.45	0.15	0.6	0.2	0.1	0.25	0.3	–0.25

Figure 2: Coordinates

1 Introduction

A multi-body system is an abstract model of the real mechanical system, built under the assumption that its elements can be treated as individual rigid or deformable parts, which are connected in various ways (translational, spherical joints, etc..) subjected to various loads and able to move with respect to each other [1].

Investigation of multibody systems consists in kinematic and dynamic analysis; however, in this report we will be only focusing on the kinematic aspects. Kinematic analysis consists in calculation of positions, velocities and accelerations of the multibody system parts by only considering geometry and time.

1.1 Objective

The main goal of this project is to develop implementation of kinematic analysis tool for a plannar mechanism using absolute coordinates in *MATLAB* environment.

1.2 Theoretical Background

This section presents a short summary of the theoretical backgrounds which are necessary to perform kinematics analysis of planar mechanism in absolute coordinate. For a more detailed explanation, one can refer to the class lecture notes[1].

1.2.1 Absolute coordinates

Absolute coordinates are coordinates in which the description of position and orientation of a selected link is directly presented w.r.t the global reference frame (other links positions are irrelevant). In the planar case each body is described by three coordinates (two linear and one angular).

A very important characteristic of the absolute coordinates is that for constraint equations corresponding to particular joint, the only coordinates that intervene are the ones of the bodies related to this joint. Allowing constraint equations to be established at a local level, and the method of analysis to be independent on structure of the mechanism.

1.2.2 Constraints

Constraint are essentially conditions that determine the relative motion of the bodies in a multibody system. Constraints can be either kinematic or driving constraint.

Kinematic constraints (material constraints)

These are constraints which are imposed by all kinematic pairs of mechanism and can be defined by the set of holonomic constraint equations in the general form:

$$\boldsymbol{\phi}^K(\mathbf{q}) = \mathbf{0} \tag{1}$$

Driving constraints (non-material constraints)

These are used for uniquely describing the desired motion of the multibody system and they are in general functions of time and generalized coordinates.

$$\boldsymbol{\phi}^D(\mathbf{q}, \mathbf{t}) = \mathbf{0} \tag{2}$$

Determining and assembling constraints

Let's start with a revolute joint (kinematic pair of *V* class): Constraint for this class are obtained by requiring that point *A* of body i and point *B* of body j coincide for all possible configurations of the mechanism.

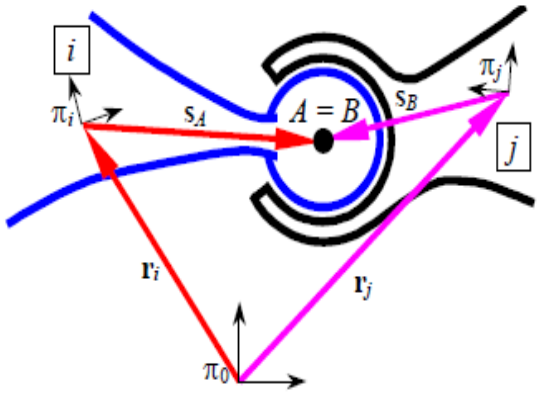


Figure 3: Revolute joint [1].

The vector form constraint equation can be written as:

$$\boldsymbol{\phi}^{\mathbf{K}\cdot} = \mathbf{r}_i + \mathbf{R}_i \mathbf{S}_A^{(i)} - (\mathbf{r}_j + \mathbf{R}_j f \mathbf{S}_B^{(j)}) \tag{3}$$

Let's proceed to translational joint, which is also kinematic pair of the *V* class. Translational joint eliminates two degrees of freedom, thus it is described by two scalar constraint equations. The first equation expresses the fact that relative orientation of the frame π_j with respect to the frame π_i remains constant:

$$\phi^{K\uparrow} = \psi_i - \psi_j - \psi^0 \tag{4}$$

The second constraint equation arises due to the fact, that point B can move with respect to point A only along the axis l.

$$\boldsymbol{\phi}^{K\angle} = \mathbf{d}^T \mathbf{v} = (\mathbf{R}_j \mathbf{v})^T (\mathbf{r}_j + \mathbf{R}_j \mathbf{S}_B - \mathbf{r}_i - \mathbf{R}_i \mathbf{S}_A) = 0 \tag{5}$$

Now that we have summarized the main idea on how kinematic constraint of revolute and translational joints are obtained, let's proceed to determining driving constraints for these joints.

Relative rotation in a revolute joint:

$$\phi^{D\angle} = \psi_i - \psi_j - \theta_{i,j}(t) \tag{6}$$

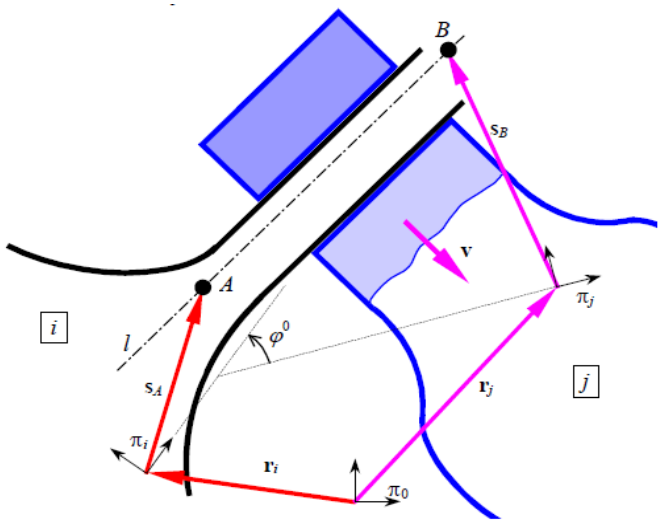


Figure 4: Translational joint [1].

Note: in the *MATLAB* implementation, for the sake of uniformity, motion is specified in the name of *fAB*.

Relative displacement in a translational joint:

$$\boldsymbol{\phi}^{D\uparrow} = (\mathbf{R}_j\mathbf{u})^T(\mathbf{r}_j + \mathbf{R}_j\mathbf{S}_B - \mathbf{r}_i - \mathbf{R}_i\mathbf{S}_A) - f_{AB}(t) = \mathbf{0} \tag{7}$$

Now that we have defined both the kinematic and driving constraints, the constraint equation can be assembled as:

$$\boldsymbol{\phi} = \boldsymbol{\phi}(\mathbf{q}, t) = \begin{bmatrix} \boldsymbol{\phi}^K \\ \boldsymbol{\phi}^D \end{bmatrix} = \mathbf{0} \tag{8}$$

1.2.3 Jacobian

Simply put, the *Jacobian* is the partial differentiation of the constraints vector, Equation 8, w.r.t the coordinate vector, $\mathbf{q}$.

$$\boldsymbol{\phi}_{\boldsymbol{q}} = \boldsymbol{\phi}_{\boldsymbol{q}}(\dot{\mathbf{q}}, \mathbf{q}, t) = \begin{bmatrix} \boldsymbol{\phi}^K_{\boldsymbol{q}} \\ \boldsymbol{\phi}^D_{\boldsymbol{q}} \end{bmatrix} \tag{9}$$

1.2.4 Kinematic problem

This section summarizes the method of obtaining solution for position, velocity and acceleration problems.

Position problem

The position problem involves in determining the generalized coordinates vector, $\mathbf{q}$, which satisfies Equation 8. However, Equation 8 is a nonlinear algebraic equation and generally using a numerical method is required to obtain the solution.

Velocity and acceleration problems

Since $\mathbf{q}$ is not known nor explicit function of time, it cannot be differentiated to directly obtain $\dot{\mathbf{q}}$ or $\ddot{\mathbf{q}}$. An alternate solution is to use chain rule of differentiation as follows:

$$\dot{\boldsymbol{\phi}}(\dot{\mathbf{q}}, \mathbf{q}, t) = \frac{d}{dt}\boldsymbol{\phi}(\mathbf{q}, t) = \boldsymbol{\phi}_{\boldsymbol{q}}\dot{\mathbf{q}} + \boldsymbol{\phi}_t = \begin{bmatrix} \boldsymbol{\phi}_{\boldsymbol{q}}^K \\ \boldsymbol{\phi}_{\boldsymbol{q}}^D \end{bmatrix} \dot{\mathbf{q}} + \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\phi}_t^D \end{bmatrix} = \mathbf{0} \tag{10}$$

Then, the velocity problem boils down to solving Equation 11

$$\boldsymbol{\phi}_{\boldsymbol{q}}\dot{\mathbf{q}} = -\boldsymbol{\phi}_t = \begin{bmatrix} \mathbf{0} \\ -\boldsymbol{\phi}_t^D \end{bmatrix} = \mathbf{0} \tag{11}$$

Similarly, the acceleration problem can be solved by solving the resulting equation after differentiating both sides of Equation 10.

$$\begin{aligned} \ddot{\boldsymbol{\phi}}(\ddot{\mathbf{q}}, \dot{\mathbf{q}}, \mathbf{q}, t) &\equiv \boldsymbol{\phi}_{\boldsymbol{q}}\ddot{\mathbf{q}} + (\boldsymbol{\phi}_{\boldsymbol{q}}\dot{\mathbf{q}})_{\mathbf{q}}\dot{\mathbf{q}} + 2\boldsymbol{\phi}_{t\mathbf{q}}\dot{\mathbf{q}} + \boldsymbol{\phi}_{tt} \\ &\equiv \begin{bmatrix} \boldsymbol{\phi}_{\boldsymbol{q}}^K \\ \boldsymbol{\phi}_{\boldsymbol{q}}^D \end{bmatrix} \ddot{\mathbf{q}} + \begin{bmatrix} (\boldsymbol{\phi}_{\boldsymbol{q}}^K\dot{\mathbf{q}})_{\mathbf{q}} \\ (\boldsymbol{\phi}_{\boldsymbol{q}}^D\dot{\mathbf{q}})_{\mathbf{q}} \end{bmatrix} \dot{\mathbf{q}} + 2\begin{bmatrix} \mathbf{0} \\ \boldsymbol{\phi}_{t\mathbf{q}}^D \end{bmatrix} \dot{\mathbf{q}} + \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\phi}_{tt}^D \end{bmatrix} \dot{\mathbf{q}} = \mathbf{0} \end{aligned} \tag{12}$$

Equation 12 can be rearranged to obtain the acceleration equation as follows:

$$\boldsymbol{\phi}_{\mathbf{q}}\ddot{\mathbf{q}} = \boldsymbol{\Gamma} \tag{13}$$

Where:

$$\boldsymbol{\Gamma} \equiv \begin{bmatrix} \boldsymbol{\Gamma}^K \\ \boldsymbol{\Gamma}^D \end{bmatrix} \equiv \begin{bmatrix} -(\boldsymbol{\phi}_{\boldsymbol{q}}^K\dot{\mathbf{q}})_{\mathbf{q}}\dot{\mathbf{q}} \\ -(\boldsymbol{\phi}_{\boldsymbol{q}}^D\dot{\mathbf{q}})_{\mathbf{q}}\dot{\mathbf{q}} - 2\boldsymbol{\phi}_{t\mathbf{q}}^D\dot{\mathbf{q}} - \boldsymbol{\phi}_{tt}^D \end{bmatrix} \tag{14}$$

1.2.5 Marker points kinematics

Consider Figure 5 for determining the position, velocity and acceleration of a point on a body.

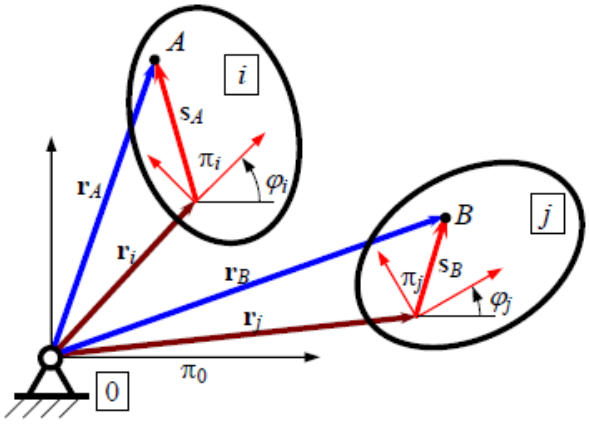


Figure 5: A point on a body.

Assume now that point *A* belongs to the body i, the position vector of point *A* is given by:

$$\mathbf{r}_A = \mathbf{r}_i + \mathbf{R}_i \mathbf{S}_A^{(i)} \tag{15}$$

The velocity of point *A* becomes (note that $\dot{\mathbf{S}}_A^{(i)} = 0$):

$$\dot{\mathbf{r}}_A = \dot{\mathbf{r}}_i + \Omega \mathbf{R}_i \mathbf{S}_A^{(i)} \dot{\psi}_i \tag{16}$$

And similarly the acceleration of point *A* can be determined as:

$$\ddot{\mathbf{r}}_A = \ddot{\mathbf{r}}_i + \Omega \mathbf{R}_i \mathbf{S}_A^{(i)} \ddot{\psi}_i - \mathbf{R}_i \mathbf{S}_A^{(i)} \dot{\psi}_i^2 \tag{17}$$

1.2.6 Singularity

A *singularity* is a point in the robot's (mechanism's) configuration space where the *Jacobian* matrix loses rank, meaning that the robot loses one or more degrees of freedom. In general by analyzing the *Jacobian* matrix we can find the singular configurations.

In this project we just have to determine when a singular configuration occurs. For that we can use the following techniques.

- Determining the rank of the *Jacobian* matrix and checking if it is full rank.
- Equivalently, we can determine the determinant of the *Jacobian* matrix and check if it is different from *zero*.

  Please note that because of the precision of *MATLAB* (and computers in general) we may not get exactly *zero*. To overcome this, a custom *zero* can be defined - in this implementation determinant with magnitude $\leq 1e^{-6}$ is considered zero.

2 Method

In this section the detailed kinematics analysis of the planar mechanism depicted in Figure 1 is presented.

First, the kinematics analysis of the mechanism using *Adams* is performed as presented in Section 2.1. This will serve as a reference to compare with. Section 2.2 delves in to the details of how a general purpose tool for kinematics analysis of planar mechanism can be developed in *MATLAB* environment.

2.1 Analysis in Adams

Performing the kinematic analysis of the mechanism is a relatively simple task as it involves only defining the bodies, joints and motions. This is only done so that it serves as a reference for the *MATLAB* implementation. The mechanism implemented in *Adams* is shown in Figure 6

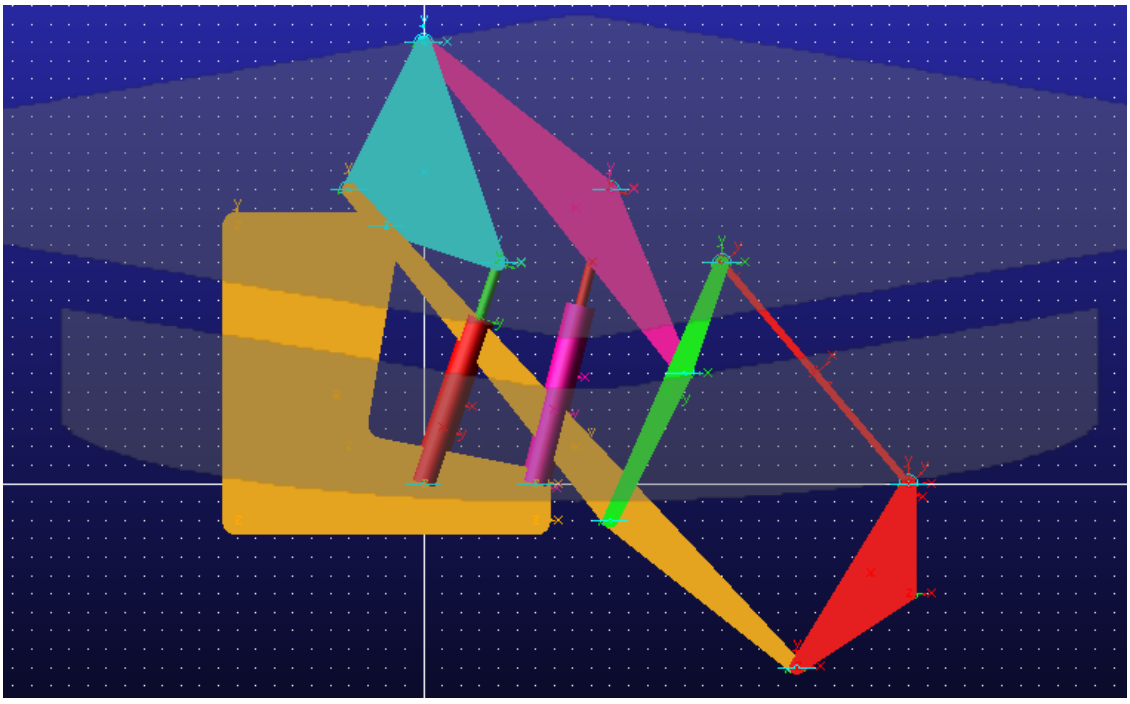


Figure 6: Mechanism structure implemented in *Adams* environment.

After defining the structure of the system, joints and motion, the kinematic analysis can be performed by simulating the system.

After a number of simulations I've decided the transnational joint motion which is defined as a function of time as

$$x_k = l_k + a_k sin(\omega_k t + \phi_k) \quad k = 1, 2$$

to be:

- For transnational joint *AD*:

$$0.05 - 0.08 sin(\pi t + \frac{\pi}{4})$$

- For transnational joint *BH*:

$$0.04 - 0.1sin(\frac{\pi}{4}t - \frac{\pi}{3})$$

As can be seen from the above equations, a different parameter (amplitude, frequency, and phase shift) is chosen for the two motions - just to make the motion more general. Of course, the parameters can be changed as long as the amplitude is kept in an acceptable range of values.

The result of the simulation in *Adams* environment will be presented in Section 3 together with the *MATLAB* simulation results.

2.2 Analysis in *MATLAB*

The *MATLAB* implementation is performed for a general case planar mechanism involving revolute (*R*) and translational (*P*) joints. Everything starting from assembling the constraints and the Jacobian to constructing the Γ vector for calculating the acceleration is performed by keeping in mind a general planar mechanism.

2.2.1 Coordinate frames assignment

The coordinate frame of the bodies is assigned at the center of mass of each body. The global frame is already assigned on the ground where joint *A* is located, $(0, 0)$. The *LRF* assignment is depicted in Figure 7.

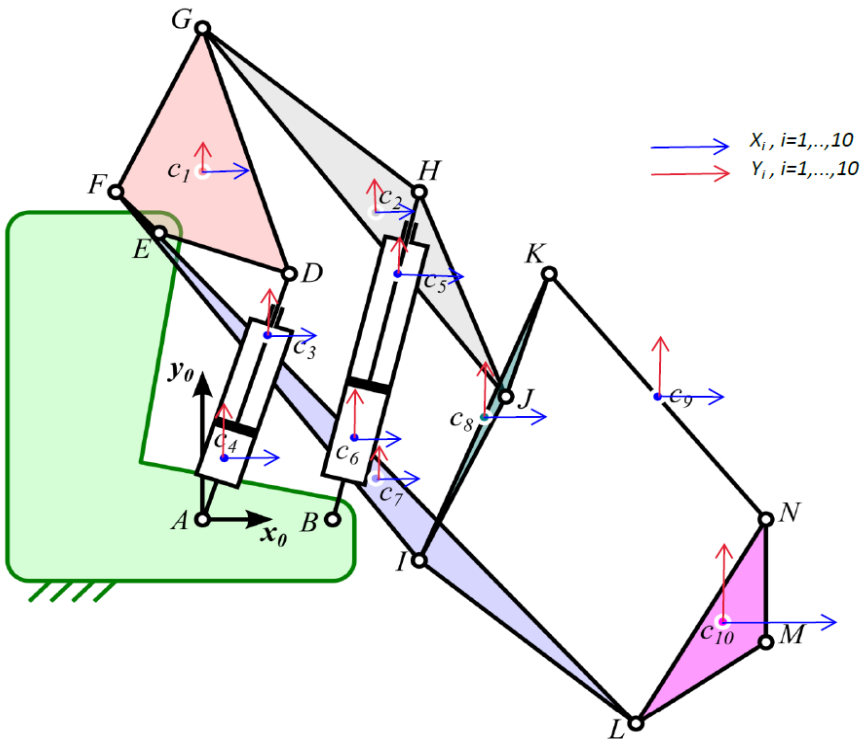


Figure 7: Local reference frames assignment.

2.2.2 Implementation description

The detailed implementation steps including some code snippets is presented below.

Defining the mechanism structure

One of the most (if not the most) important task for the implementation is to come up with an effective way of representing the mechanism that can be used in general case. For this I have chosen to use *struct*, a *MATLAB* data structure that organizes related data into fields, allowing for the grouping of diverse information under a single variable name - which is exactly what we need organizing the bodies and joints under a single mechanism.

The definition follows the pattern: *mechanismName.bodyName.bodyProperties*

bodyProperty includes:

- The location of its center of mass
- Joints that are associated with the body and their properties, which includes:
  - Joint location (if the type is P the location can be just $[NaN, NaN]'$)
  - Joint type either *P* for translational or *R* for revolute
  - Logical variable '*driving*' to indicate if there is a driving constraint associated with the joint
  - If *driving* is *true*, specify the motion as an anonymous function of time, *fAB*
  - *Special Case*: if the joint is translational and if it is directly connected to the ground, a reference point must be provided to define the axis of translation accurately.
- Initial values of the kinematic variable (position q_0, velocity d_q and acceleration dd_q)
- Marker points for which kinematic analysis is to be performed

The first object that have to be defined is the *ground* as follows:

Ground definition

```
% Initialize a struct to represent the mechanism
mechanism = struct();
% Ground: center of mass and initial values(fixed).
mechanism.ground.com = [0, 0]';
mechanism.ground.q0 = [0, 0, 0]';
mechanism.ground.dq = [0, 0, 0]';
mechanism.ground.ddq = [0, 0, 0]';
```

After the ground is defined we can start adding other bodies of the mechanism and their associated properties. The following series of code snippets demonstrates: how to define body, joints (with and without driving constraint), and how to define marker points.

Example of body definition

```
% Plate C3: center of mass and initial values
mechanism.bodyC3.com = [0.15, 0.45]';
mechanism.bodyC3.q0 = [0.15, 0.45, 0]';
mechanism.bodyC3.dq = [0, 0, 0]';
mechanism.bodyC3.ddq = [0, 0, 0]';
```

Example of joint definition without driving

```
% Joints associated with plated C3
mechanism.bodyC3.joints.joint_D.type = 'R';
mechanism.bodyC3.joints.joint_D.location = [0.2, 0.6]';
mechanism.bodyC3.joints.joint_D.driving = false;
```

Example of joint definition with driving

```
mechanism.bodyC3.joints.joint_AD.type = 'P';
mechanism.bodyC3.joints.joint_AD.location = [NaN, NaN]';
mechanism.bodyC3.joints.joint_AD.driving = true;
%Specify motion
mechanism.bodyC3.joints.joint_AD.fAB = @(t) -0.1*sin(1.5*t+0);
```

If a translational joint is directly connected to the ground, a reference point must be provided to accurately define the axis of translation. This is done as follows:

Special Case: Defining translational joint on the ground

```
mechanism.ground.joints.jointD.type = 'P';
mechanism.ground.joints.jointD.location = [NaN, NaN]';
mechanism.ground.joints.jointD.driving = false;
% Put a reference for defining the translational axis
```

```
mechanism.ground.joints.jointD.reference = [5, 4]';
```

Example of marker point definition

```
% Markers associated with bodyC3
mechanism.bodyC3.markers.D.location = [0.2, 0.6]';
```

The above code snipets demonstrated the pattern of defining mechanism. The full definition of the mechanism shown in Figure 1 can be found in the *MATLAB* script source file inside the directory of *preprocessor*: *'src/preprocessor/defineMechanism.m'*.

Assembling the constraints, Jacobian and Γ is simply accomplished by implementing the universal formulas presented in the lecture, and are repeated in Section 1.2.

Now that the mechanism is defined, let's see how one can use this implementation to perform kinematic analysis and display results.

Usage

The first step is adding all files in the source directory (*src*) to the *MATLAB PATH*. This can be simply done by executing *Initialize.m* function.

Step 1: Adding files to PATH

```
% Initialize the system by adding all files to the MATLAB PATH
Initialize;
```

The following code snippets demonstrates the rest of usage.

Step 2: Getting the mechanism definition

```
% Get the mechanism for which kinematics analysis is to be
    performed
mechanism = defineMechanism;
```

Step 3: Perform kinematic analysis for bodies of the mechanism

```
%% Kinematics analysis
% solve the kinematic problem for the mechanism bodies and their
    marker
% points.
endTime = 5;
steps = 100;
kinematics = Kinematics(mechanism, endTime, steps);
```

Now that the kinematics analysis is performed we can go a head and visualize the results.

To visualize bodies kinematic analysis result, specify the option *'b' or 'B'*, for markers specify option *'m' or 'M'*. This will generate plot of position, velocity and acceleration of each body

and markers in the mechanism respectively. If interested only in a certain body(s) or marker(s), their name(s) can be defined as cell arrays and passed to the visualization function as optional arguments.

The step by step guide of visualizing results is demonstrated in the following code snippets.

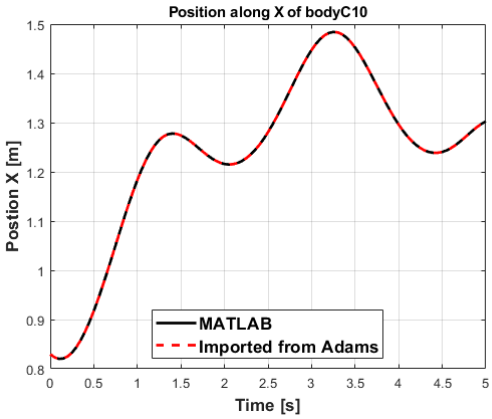
Kinematic analysis result visualization

```
%% Visualization
% To generate plot of kinematics result analysis for the entire
% bodies or markers in the mechanism use:

% Visualizer(markersKinematics, 'b')
% Visualizer(markersKinematics, 'm')

% Or we can specify the body name of interest
bodyNames = {'bodyC10'};
Visualizer(kinematics, 'b', bodyNames)

% The same thing for markers as well
bodyNames = {'bodyC8'};
markerNames = {'K'};
Visualizer(kinematics, 'm', bodyNames, markerNames)
```

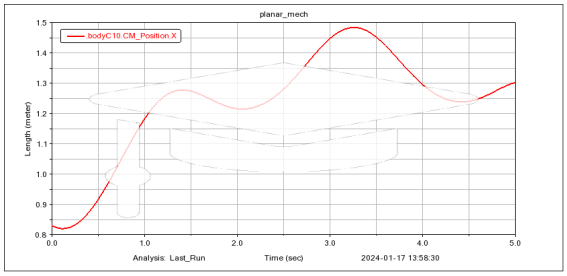
For performing comparison with results obtained from *Adams*, a separate *main.m* script is implemented. In this script both qualitative comparison (with plots) and quantitative comparison by performing root mean squared error is performed.

3 Simulation Results and Comparison

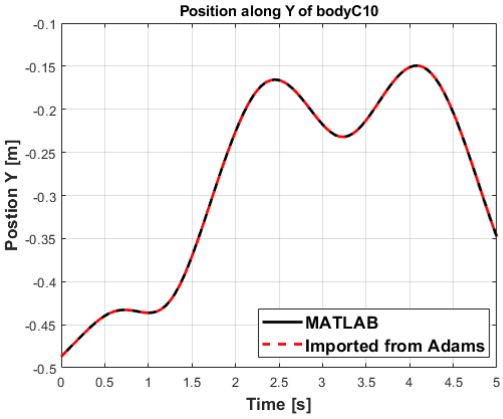
First, qualitative comparison on obtained simulation results is performed. Then, quantitative comparison is done by computing the root mean squared error. The comparison presented here is for the kinematics of *bodyC10* (the plate its *cm* is named *c10*) and for point *K*.



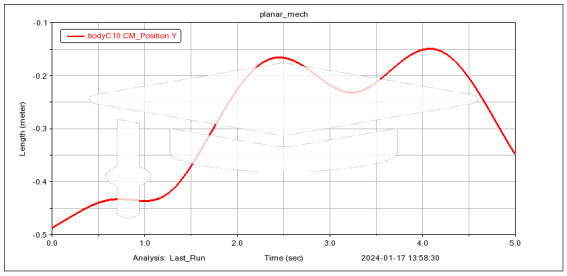
(a) X position from *MATLAB*



(b) X position from *Adams*



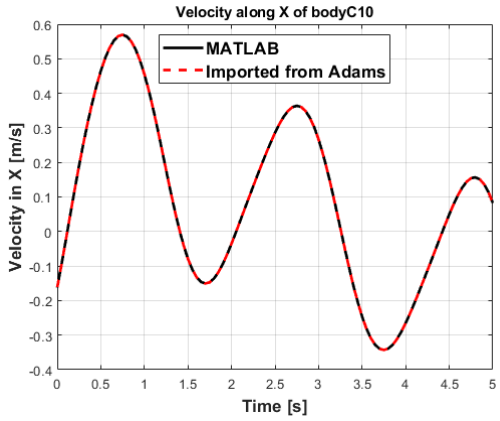
(c) Y position from *MATLAB*



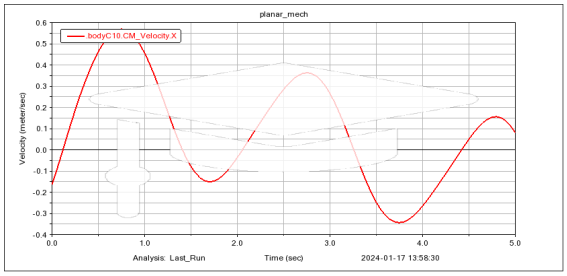
(d) Y position from *Adams*

Figure 8: Position comparison.

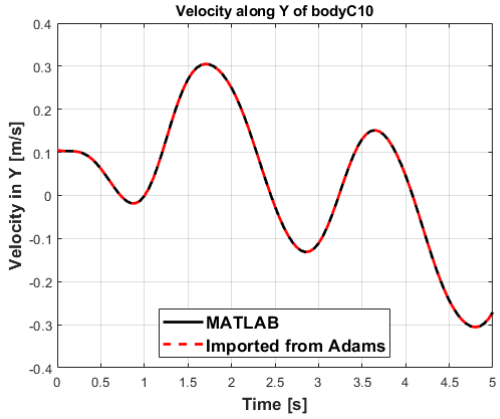
The comparison of velocity along different axes is given in the following Figures.



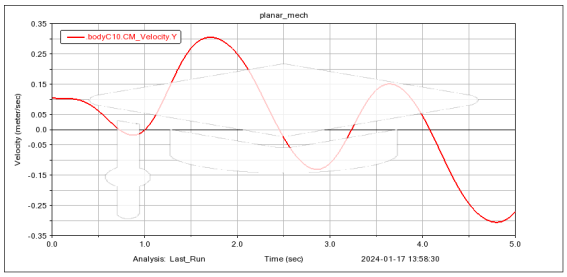
(a) VX velocity from *MATLAB*



(b) VX velocity from *Adams*



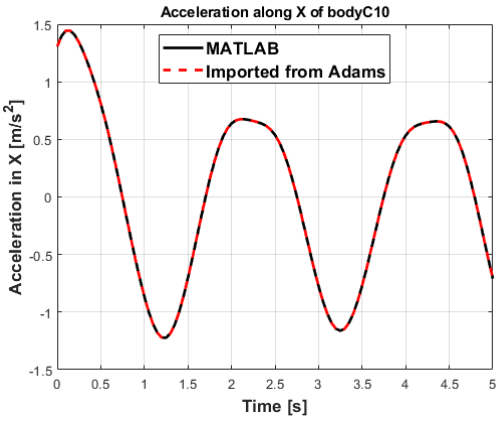
(c) VY velocity from *MATLAB*



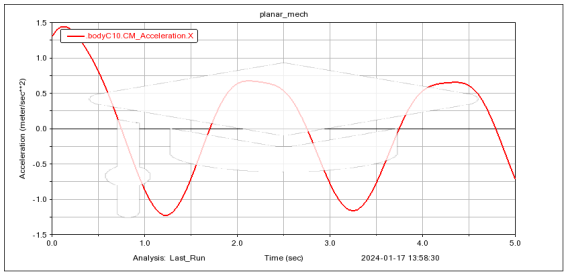
(d) VY velocity from *Adams*

Figure 9: Velocity comparison.

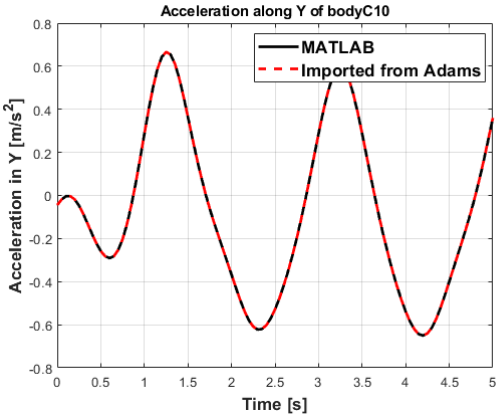
The comparison of acceleration is given in the following Figures.



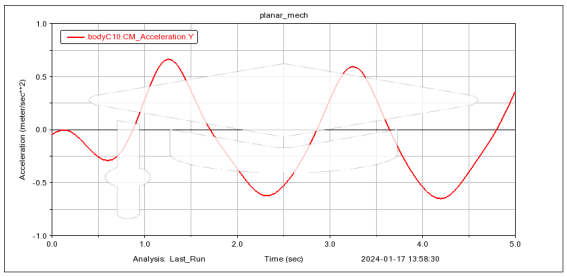
(a) AX acceleration from *MATLAB*



(b) AX acceleration from *Adams*



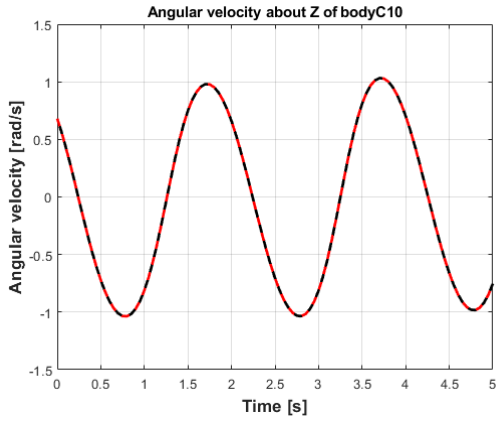
(c) AY acceleration from *MATLAB*



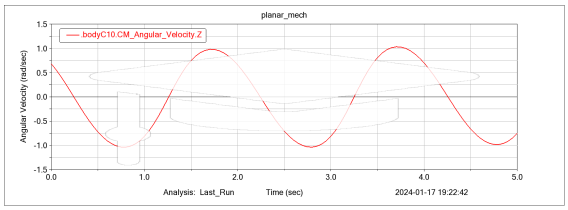
(d) AY acceleration from *Adams*

Figure 10: Acceleration comparison.

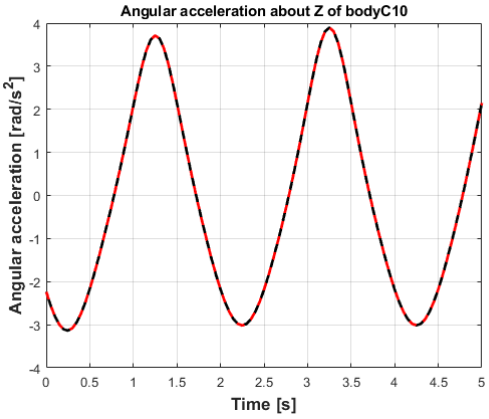
Finally, the qualitative comparison for the angular velocity and acceleration is given below.



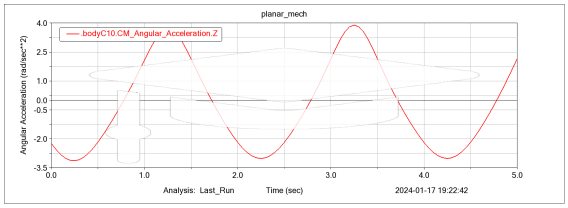
(a) ω_z from *MATLAB*



(b) ω_z from *Adams*



(c) α_z from *MATLAB*



(d) α_z from *Adams*

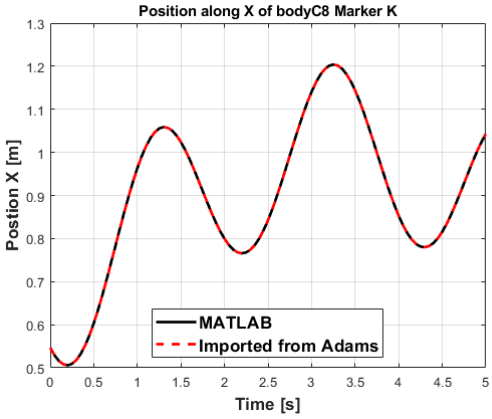
Figure 11: Comparison of angular velocity and acceleration for *bodyC10*.

Now that we have seen how the results are similar from the previous graphics, let's proceed to performing the root mean squared error to accurately determine the performance of the implemented system.

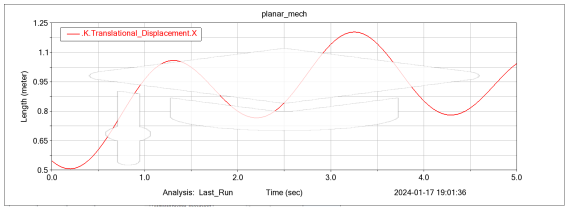
Axis	**Variable**	***RMSE***
X	Position	$2.578153e^{-07}$
	Velocity	$2.553200e^{-08}$
	Acceleration	$1.314944e^{-07}$
Y	Position	$2.812271e^{-08}$
	Velocity	$2.308130e^{-08}$
	Acceleration	$2.869656e^{-08}$
About Z	Angular velocity	$4.881307e^{-08}$
	Angular acceleration	$3.640987e^{-07}$

Table 1: Quantitative performance of the implemented system for *bodyC10*.

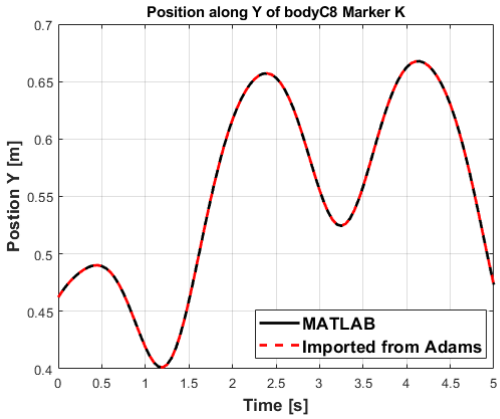
Now let's compare the kinematic analysis results obtained from the *MATLAB* implementation and *Adams* for point *K*, which is located at the joint connecting *bodyC8 and bodyC9*.



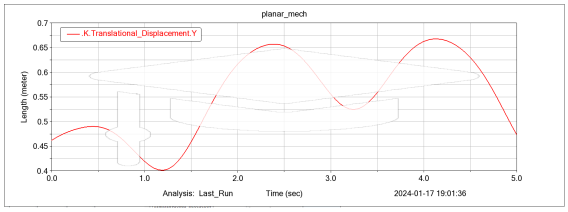
(a) X position from *MATLAB*



(b) X position from *Adams*



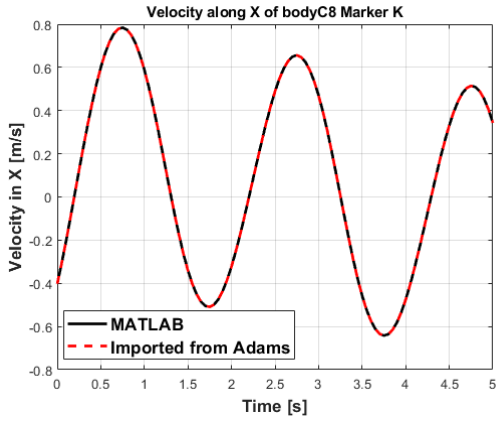
(c) Y position from *MATLAB*



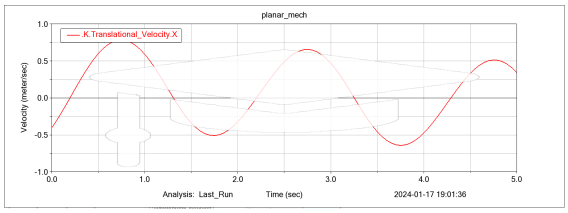
(d) Y position from *Adams*

Figure 12: Position comparison for marker *K*.

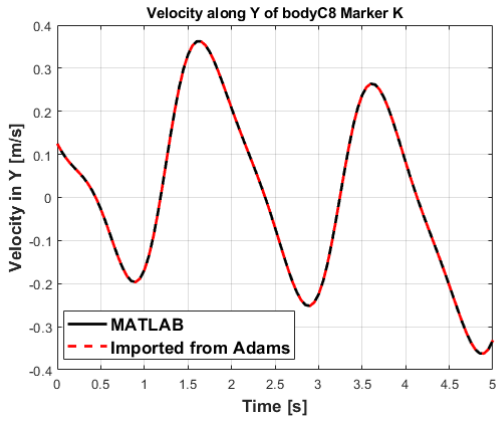
The comparison of velocity along different axes:



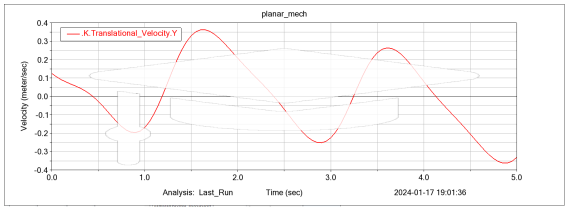
(a) VX velocity from *MATLAB*



(b) VX velocity from *Adams*



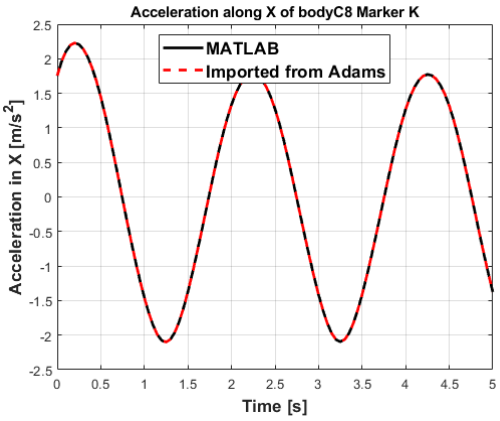
(c) VY velocity from *MATLAB*



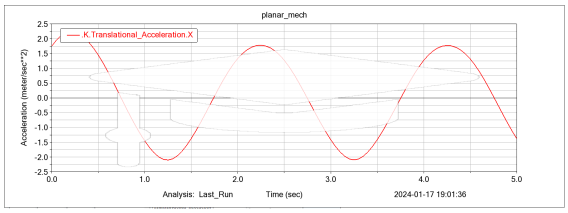
(d) VY velocity from *Adams*

Figure 13: Velocity comparison for marker *K*.

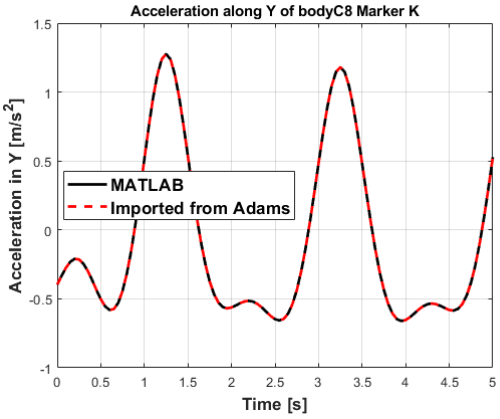
Qualitative comparison of acceleration is given in the following Figures.



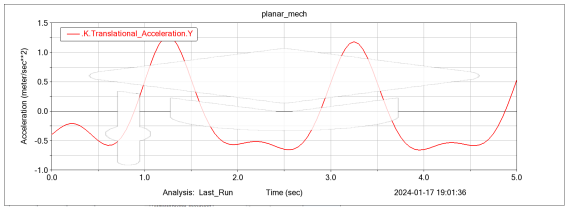
(a) AX acceleration from *MATLAB*



(b) AX acceleration from *Adams*



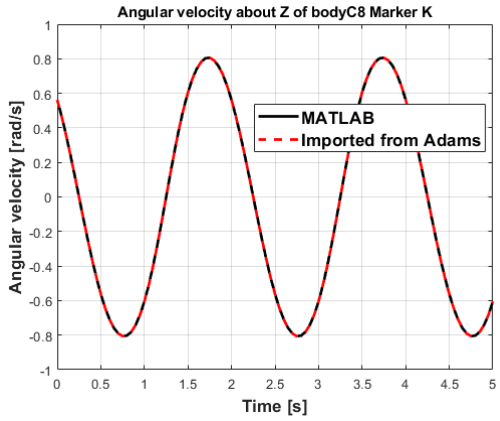
(c) AY acceleration from *MATLAB*



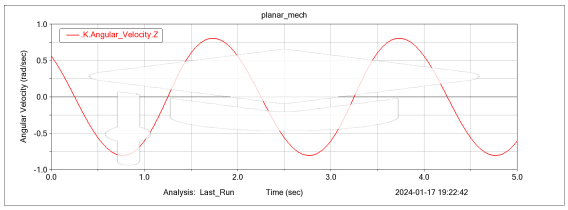
(d) AY acceleration from *Adams*

Figure 14: Acceleration comparison for marker *K*.

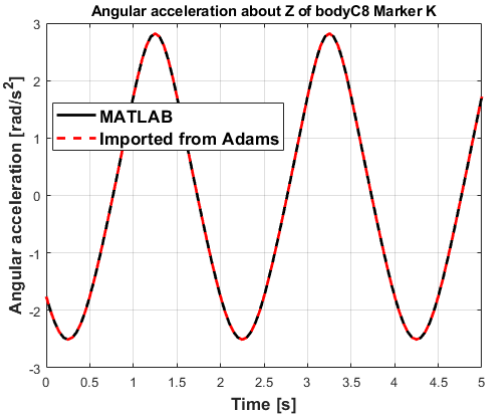
Finally, the qualitative comparison for the angular velocity and acceleration for marker K is given below.



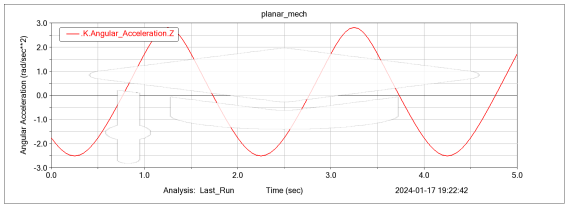
(a) ω_z from *MATLAB*



(b) ω_z from *Adams*



(c) α_z from *MATLAB*



(d) α_z from *Adams*

Figure 15: Comparison of angular velocity and acceleration for marker K.

Now that we have seen how the results are similar qualitatively from the previous graphics, let's proceed to performing the root mean squared error to accurately determine the performance of the implemented system for marker points.

Axis	**Variable**	***RMSE***
X	Position	$1.582851e^{-07}$
	Velocity	$2.767416e^{-08}$
	Acceleration	$2.528726e^{-07}$
Y	Position	$2.900402e^{-08}$
	Velocity	$2.271474e^{-08}$
	Acceleration	$1.133324e^{-07}$
About Z	Angular velocity	$2.929659e^{-08}$
	Angular acceleration	$2.350943e^{-07}$

Table 2: Quantitative performance of the implemented system for marker *K*.

Test performed for other mechanism (*i.e., the mechanism discussed in Lecture 5*) can be found in Appendix A. This is done to show that the implementation is applicable for a general case planar mechanism.

4 Conclusions

In conclusion, the detailed development and implementation of a kinematic analysis tool for a planar mechanism (involving only revolute and translational joints) using absolute coordinates in MATLAB environment was presented. The main objective was to develop the implementation in *MATLAB* environment, and compare the results with a reference analysis performed in *Adams*.

The implementation in *MATLAB* involved structuring the mechanism using a defined data structure (*MATLAB struct* was chosen), assembling constraints, *Jacobian*, and solving the kinematic problems numerically.

Simulation results were presented and compared both *qualitatively* and *quantitatively* with a reference analysis performed in *Adams*. The comparison demonstrated a high degree of similarity between the *MATLAB* implementation and *Adams* (error in the range of $1e^{-7}$ to $1e^{-8}$), validating the accuracy of the developed tool.

A Test Mechanism

The *MATLAB* implementation is tested on an other mechanism, the mechanism discussed on lecture 5, to see if it works for general case.

The *Adams* implementation of the mechanism is depicted in Figure 16.

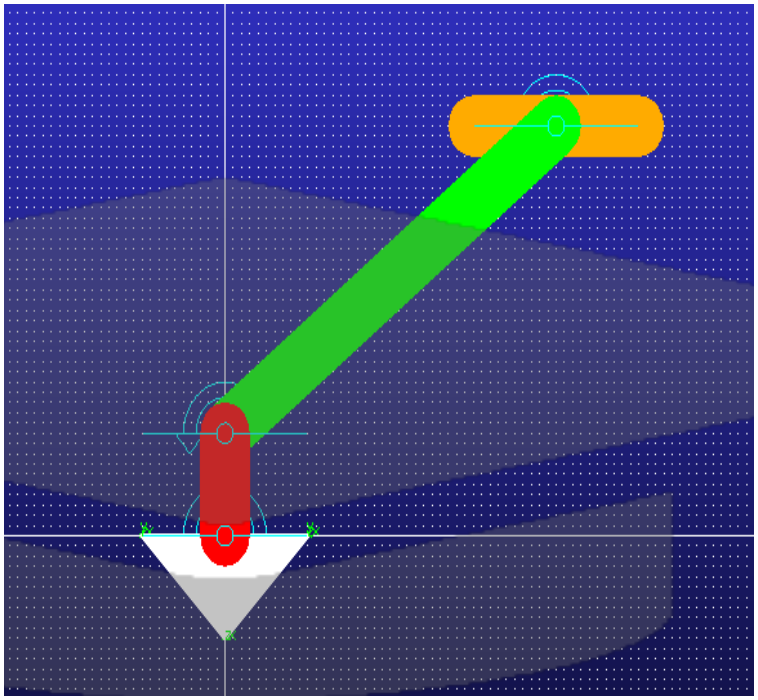
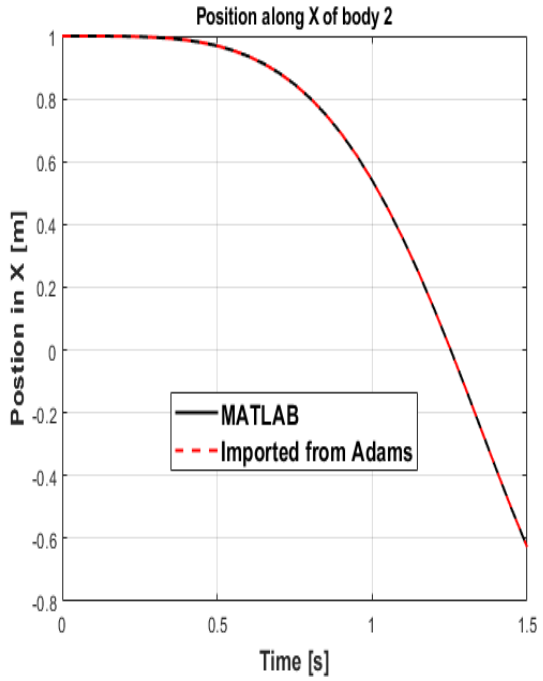


Figure 16: Model in *Adams* of the test mechanism.

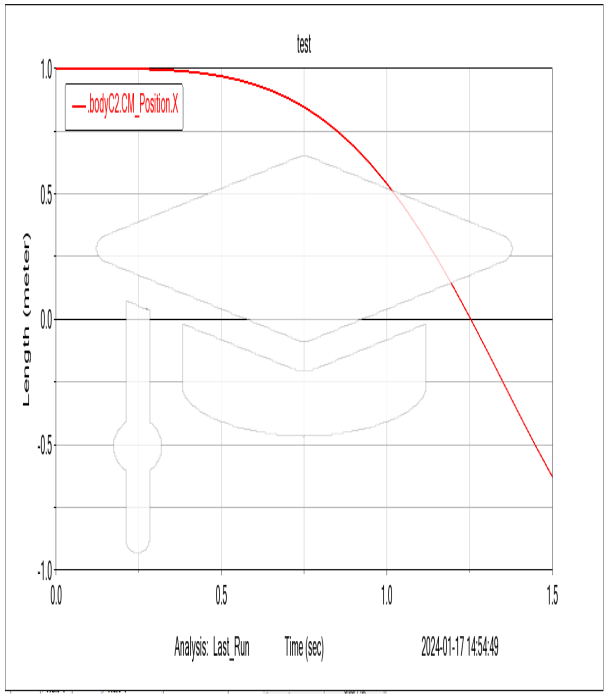
A.0.1 Simulation results of the test mechanism

The position, velocity and acceleration of body 2, the green part in Figure 16, is chosen for comparison.

The simulation result is presented in the following graphics. Only the result along axis X is presented here; however, the same result is obtained for other axes s well as bodies.

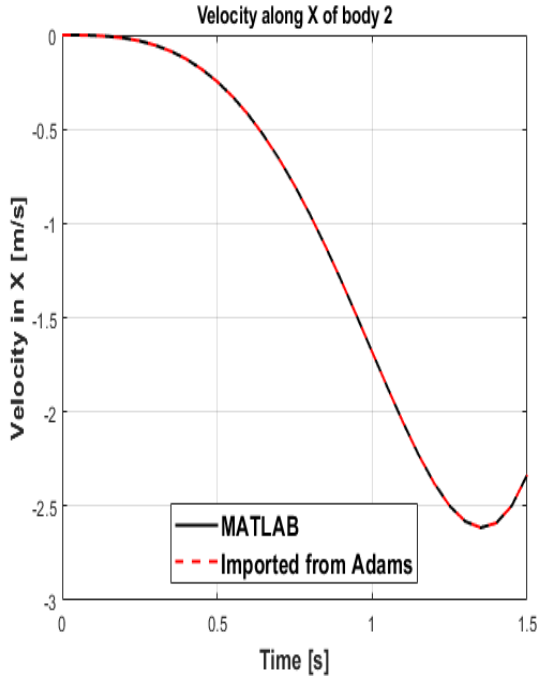


(a) Result from *MATLAB* implementation.

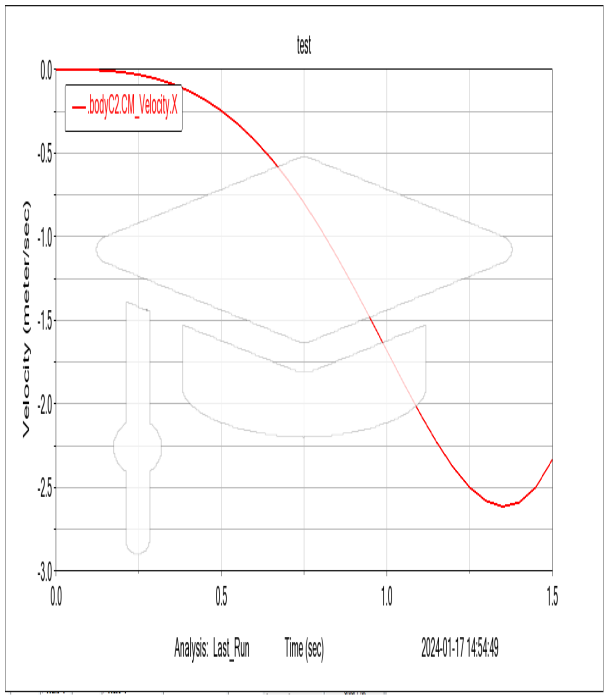


(b) Result from *Adams* simulation.

Figure 17: Body 2 position along axis *X*.

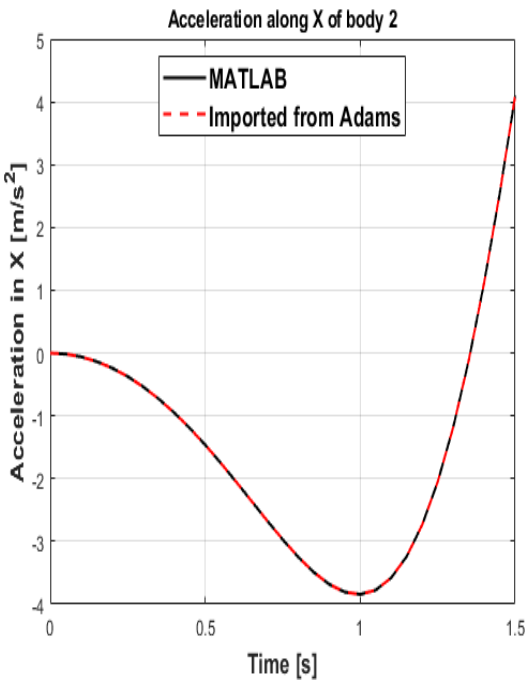


(a) Result from *MATLAB* implementation.

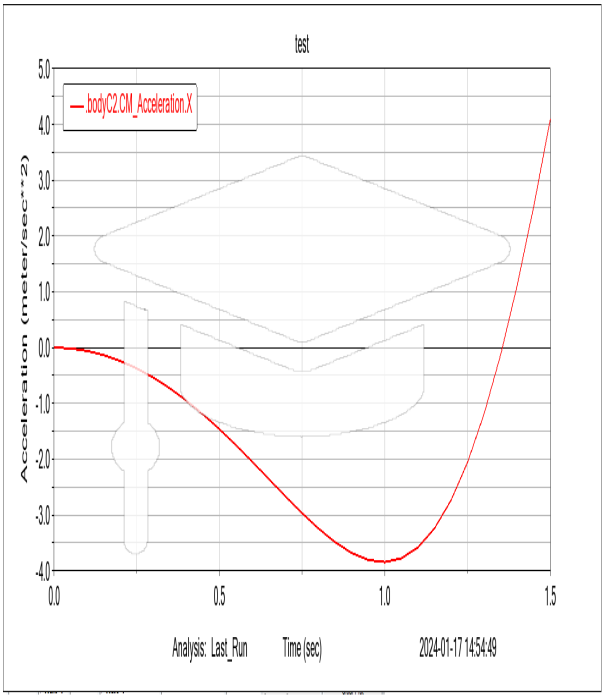


(b) Result from *Adams* simulation.

Figure 18: Body 2 velocity along axis *X*.



(a) Result from *MATLAB* implementation.



(b) Result from *Adams* simulation.

Figure 19: Body 2 acceleration along axis X.

As we can see from the above results we can confirm that the implementation can be applied to a general case planner mechanism.

To present quantitative performance analysis the root mean squared error is calculated. The result obtained is given in Table 3.

Variable	Root Mean Squared Error
Position in X	$2.415620e^{-08}$
Velocity in X	$1.689488e^{-07}$
Acceleration in X	$2.343198e^{-07}$

Table 3: *RMSE* performance of the implemented system tested against the results from *Adams*.

References

[1] M. Wojtyra J. Frączek. *Lecture notes*. 2023/24.