MEU44B17-202526 Multibody Dynamics (4B17)

Leg Kick Assignment

Finn O'Connor | 22336740

Coláiste na Tríonóide, Baile Átha Cliath
Trinity College Dublin
Ollscoil Átha Cliath | The University of Dublin

Contents

Instructions and Deliverables

The following instructions and deliverables were provided in the assignment brief. I have combined some of the instructions at the top of the file to the deliverables to ensure that each step was carried out correctly.

1) A Summary in your own words of what the model represents (roughly 1 page).
    a) Develop the model to implement a leg/foot so that the foot kicks the ball which then bounces off the vertical wall.
    b) Orient the wall surface so that it is oriented obliquely with respect to the global X, Y and Z axes and repeat the simulation so that the ball bounces at an angle.
2) Demonstrate the relationship between the global and local components of a vector of your choice by using the ball cg acceleration as an example (roughly 1 page).
3) Show by example the effects of changing the integration timestep and the initial conditions and the contact stiffness (roughly 1 page).
    a) Double the stiffness of the contact between the foot and ball and test the effect on the acceleration time-history of the ball.
4) Use your results to show the relationship between the rotation matrix and its eigenvectors/eigenvalues and the actual orientation of the ball). Lecture theory weeks 1-3 is needed for this (roughly 1 page).
    a) Use supplied MATLAB scripts to extract the linear/angular position of the ball at time points of your choice, before and after the ball bounces off the wall.

1) Summary of the Model

This following report describes the behaviour and mechanics behind how a leg, foot, wall, and football interact when producing a kicking motion. The model was built using the MADYMO (MAthematical DYnamic MOdels) software used to simulate the behaviour of two or more rigid bodies contacting one another with a force interaction. The software has a variety of uses from simulating car crashes to how human body parts interact when coming in contact with another rigid body. MADYMO is comprised of multiple different packages each used during simulation. We use various components within the software. Firstly, the general physics is defined using the XMADgic preprocessing software, an XML-editor used to prepare the file to be solved using the MADYMO Solver. Once the software solves the conditions, the results are loaded up in the postprocessing software called MADPost. Visual time-history plots and animations can be viewed using MADPost, enabling the user to extract valuable data and results from the simulation.

Within the XML file, there were already predefined rigid bodies in the form of the ball and wall systems. In order to carry out the simulation, a leg system needed to be added. The leg system is comprised of a series of rigid bodies, ellipsoids, and joints. A rigid body is defined as an idealised solid object in which the distance between any two points within that body remain constant regardless of any external forces or torques that are applied to it. The leg system is made up of 2 rigid bodies – the *thigh_body* and the *leg_foot_body*. A joint is a mechanical connection between two rigid bodies constraining their relative motion in specific ways. An ellipsoid is essentially a 3D oval or stretched sphere and is used to define 3D geometric shapes – in this case for the sake of contact definition. The initial positions and velocities of the knee and hip joint also need to be defined. These were given to us as predetermined initial conditions in the assignment brief but can be altered to yield different results.

There are also a variety of other parameters that are used to control the simulation such as:

- The total simulation time and the timestep.
- Rigid body properties such as the mass and inertia of the ball and leg system.
- Contact definitions such as the contact stiffness and damping and friction coefficients.
- The X, Y, and Z orientation of the wall.

The purpose of this model is to simulate a leg kicking a ball, which then impacts an angled wall, to study contact forces, kinematics, and coordinate transformations.

2) Relationship Between the Global and Local Components of a Vector

The following section analyses the difference between the global and local variables for the Y and Z components of a chosen vector. My vector of choice was the linear acceleration of the ball system centre of gravity. A global coordinate system is a fixed universal frame of reference, such as the earth's geographic coordinate system. On the other hand, a local coordinate system is a reference frame defined within a specific area or object with respect to the global system. In this case the local coordinate system is defined on the ball in its starting position. The local and global coordinate systems are aligned at the balls starting position. As the ball is kicked, it begins to spin, and the initial position vectors of the ball and local coordinate system are rotated. This can be done mathematically by applying a rotation matrix to the initial vector. Mathematically this process can be defined by simply saying:

$$v_{local} = A^{21} \cdot v_{global}$$

- ## Linear Acceleration in the Y-Direction:

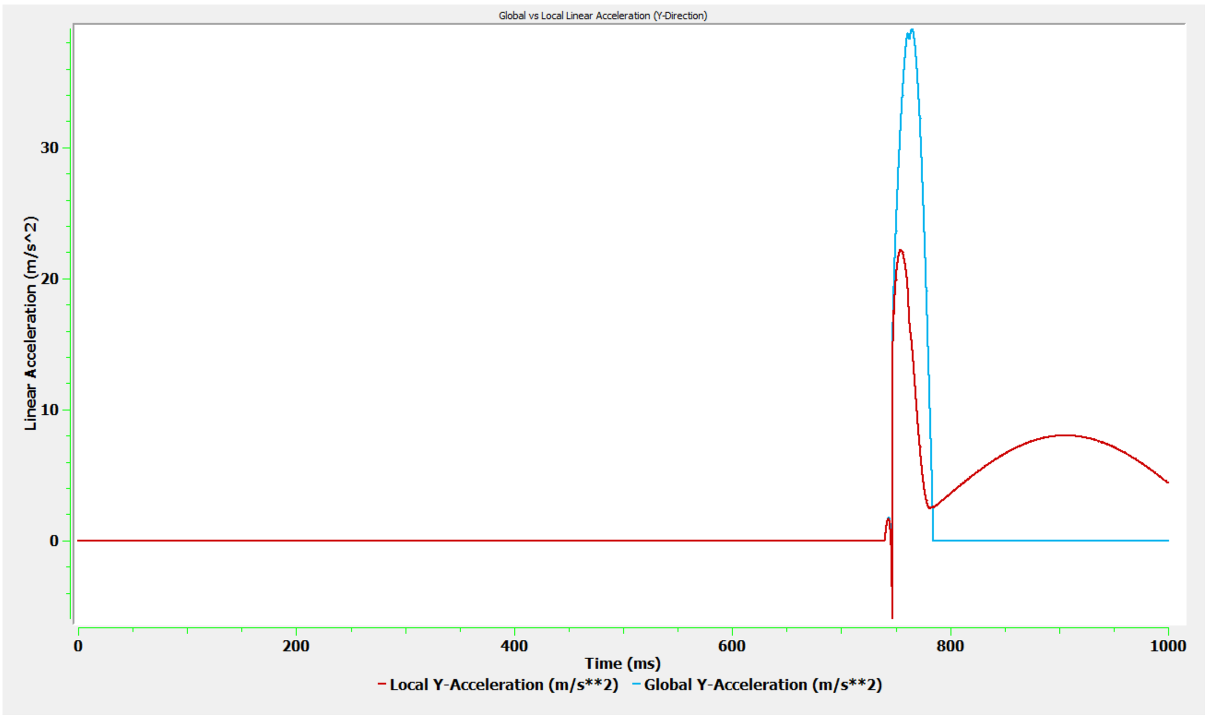


Figure 1: Global vs local linear acceleration in the y-direction with respect to time

The ball-wall instance of contact takes place at approximately 750ms. We can see clearly from the above graph that the y-axis does not experience any acceleration until the moment of contact occurs. This is because, up until the contact the ball is only changing acceleration on the X-Z plane. Once it strikes the wall, which is positioned obliquely on all 3 axes, the ball deviates along the y-axis also.

- ## Linear Acceleration in the Z-Direction:

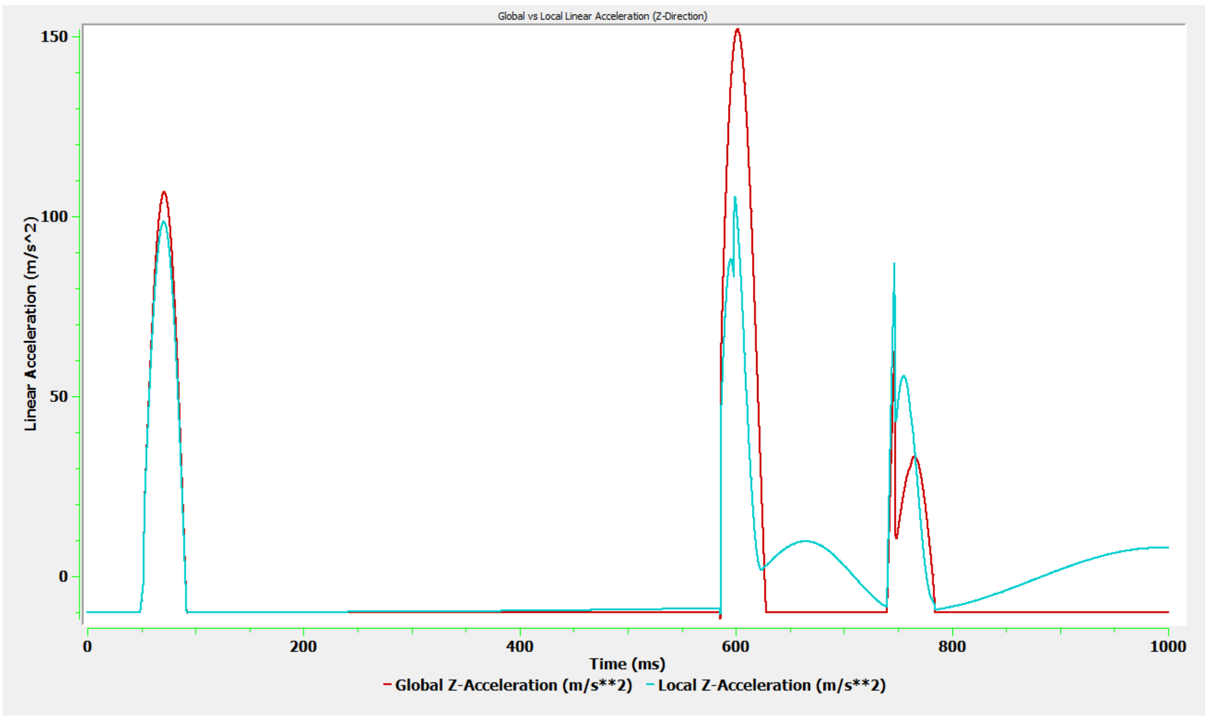


Figure 2: Global vs local linear acceleration in the z-direction with respect to time

When studying the global and local linear accelerations on the z-axis, it is clear that there is acceleration even before the ball hits the wall. It is important to note that compared to the y-direction, there is a constant acceleration which comes in the form of gravity – hence why both curves begin at -9.81 m/s^2 on the y-axis. The curves differ slightly after the first contact, is due to the ball-foot contact causing a slight rotation, and thus acceleration, of the ball's local coordinate system. The first bounce causes this acceleration to be even greater. Once the ball hits the wall, the local coordinate system deviates off the wall at a skew axis due to the wall being positioned obliquely and the accelerations differ massively.

3) Integration Timestep, Initial Conditions and the Contact Stiffness Variation

- Variation of Integration Timestep

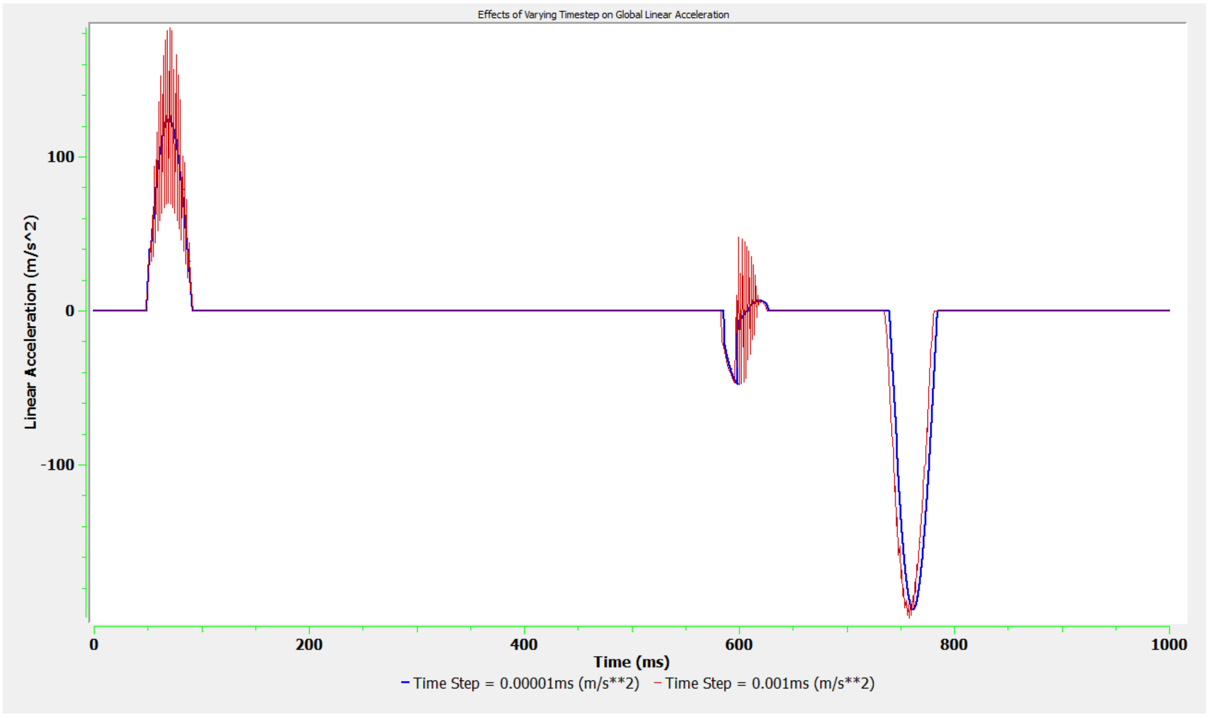


Figure 3: Linear acceleration in the global x-direction plotted against time with varied timesteps

In MADYMO, and a lot of other simulation software, the timestep (Δt) is the discrete interval used by the solver to process the equations of motion at each frame. It discretises the problem into a series of small steps, where at each step the problem is solved. The smaller the timestep is, the more computationally intensive the problem becomes, ultimately requiring more compute and time - but it yields a more accurate final result. A smaller timestep reduces the reliance on approximation methods (Runge-Kutta in the case of MADYMO) to fill the gaps between timesteps. This produces a smoother graphical output, as demonstrated by the blue curve in the above figure.

On the other hand, a larger timestep reduces the required compute and simulation run time – which ultimately gives a less accurate result. As we can see, the red curve follows the same shape as the blue curve, but it accompanied by much more 'noise'. This fuzzy effect, which is caused by numerical oscillations, occurs because the timestep is not small enough so the solver overshoots and undershoots between integration steps when approximating – resulting in instability in the output signal. The key to running a successful simulation is finding a point where the accuracy of the results isn't compromised by a lack of compute, and conversely where an unnecessary amount of compute is used up for no reason.

From running the simulations at different timesteps, I determined the following outcomes:

Timestep (ms)	Effect
1×10^{-3} (red)	Inaccurate results, lots of noise generated upon contact
1×10^{-5} (blue)	Optimal timestep where neither accuracy nor computer were compromised
1×10^{-8} (no plot)	Unnecessarily computationally intensive, highly accurate results.

- Variation of Initial Conditions (Ball Mass)

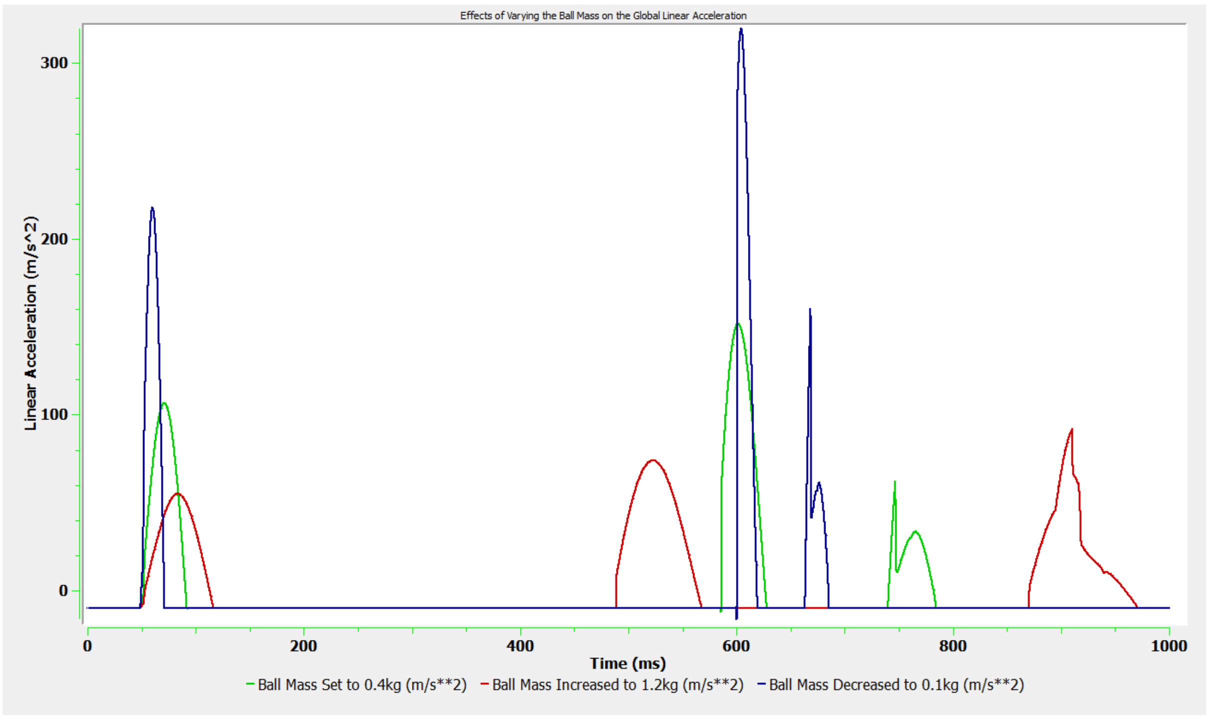


Figure 4: Linear acceleration in the global z-direction plotted against time with varied ball mass

For the next part, we had to choose an initial condition to vary and observe the effects that it had on a vector of our choice. I chose to analyse the effects that the mass of the ball had on the global linear acceleration in the z-axis. The initial mass of the ball was set to 0.4kg. The governing principle behind analysing the effects that the mass has on the acceleration can be related to Newton's Second Law:

$$a = \frac{F}{m}$$

It is expected that by decreasing the mass of the ball, there would be an increase in the overall linear acceleration – and conversely by increasing the mass the acceleration would decrease. From the above figure, we can see that this is true. The red curve (1.2kg) has a much smaller peak acceleration than the other two. It also experiences its second impulse of acceleration much earlier than the other two curves. This is because the second impulse comes from it bouncing on the ground twice before hitting the wall due to it having less displacement in the z-direction upon impact from the foot – whereas the 0.1kg ball (blue curve) hits the wall before hitting the ground due to it being so light. The red curve also has much wider horizontal impulses – which can be attributed to the 1.2kg ball needing a greater force impulse to change its momentum and also having greater inertial forces. The lighter balls have smaller contact durations and have sharper oscillations – reflecting their lower inertia.

- Variation of Contact Stiffness

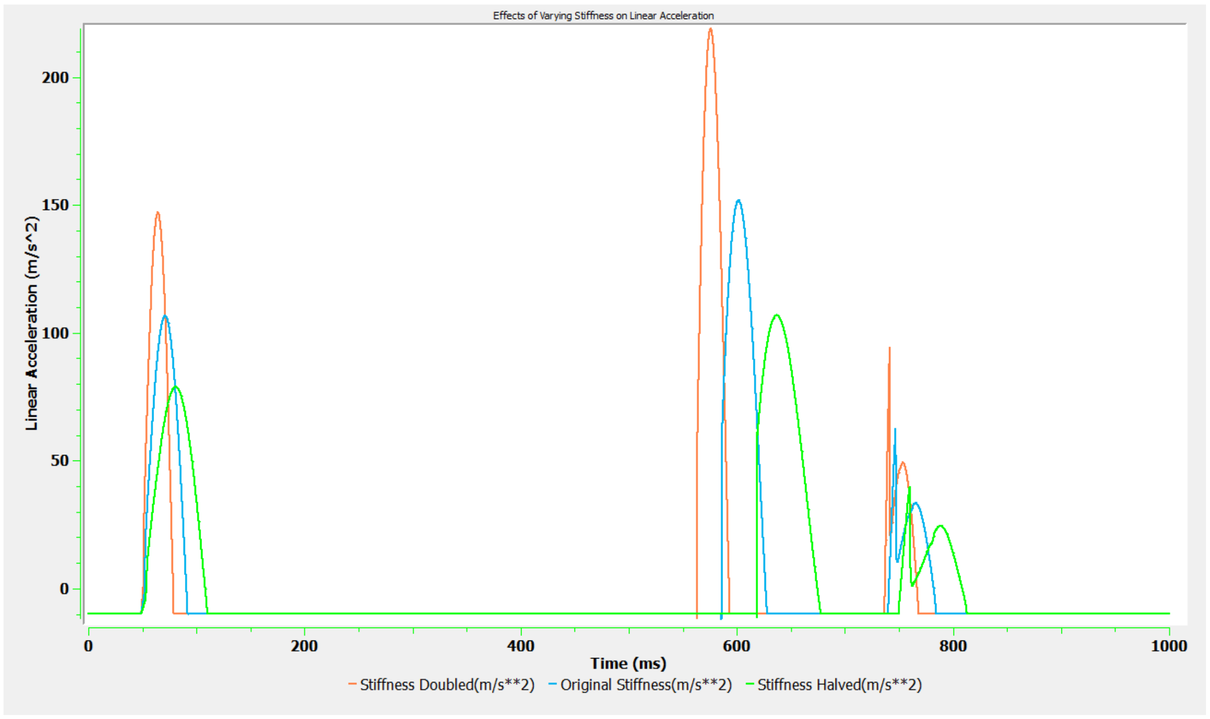


Figure 5: Linear acceleration in the global z-direction plotted against time with varied contact stiffness

As we stated previously, this simulation contains rigid bodies. In MADYMO, the contact stiffness defines the force-penetration relationship used to approximate and simulate the deformation and elasticity that would occur in real-life. A higher stiffness would produce steeper force-penetration curves, leading to larger contact forces over shorter time intervals. A lower stiffness spreads the contact force over a longer duration – leading to lower peak forces. Similarly to the ball's mass effect on linear acceleration, the contact force can be related to acceleration through Newton's Second Law:

$$a = \frac{F}{m}$$

This is verified through the above figure. When the contact stiffness is increased, it generates a higher contact force over a shorter interval, which results in a greater peak linear acceleration.

4) Eigenvectors/Eigenvalues of the Rotation Matrix and Ball Rotation Relationship

Once the ball is kicked by the leg/foot system it undergoes a rotational transformation.

For an eigenvector, the corresponding eigenvalue is the scale factor of the transformation.

$$[A^{21}]\{v\} = \lambda\{v\}$$

Every rotation matrix $[A^{21}]$ has 3 eigenvalues:

$$\lambda_1 = 1$$

$$\lambda_{2,3} = \frac{tr[A^{21}] - 1}{2} \pm \sqrt{\left[\frac{tr[A^{21}] - 1}{2}\right]^2 - 1}$$

Roots 2 and 3 can be simplified to:

$$\lambda_{2,3} = \cos\phi \pm i\sin\phi$$

Where the angle of rotation (ϕ) can be solved using the formula:

$$\cos^{-1}\left(\frac{tr[A^{21}] - 1}{2}\right), \quad \text{where tr}[\text{A}^{21}] \text{ is the trace of the rotation matrix}$$

For a 3D rotation matrix there is always one real eigenvalue whose corresponding eigenvector is the axis of rotation. The other two eigenvalues are complex conjugates of the form $e^{\pm i\theta}$ where θ is the rotation angle.

At t = 270ms:

$$Position\ Vector = [0.8881 \quad 0.0000 \quad -0.5374]$$

The y-coordinate remains 0 as the ball lies entirely on the X-Z plane and has not deviated along the y-axis yet. The rotation matrix can be calculated by substituting in the angle of rotation.

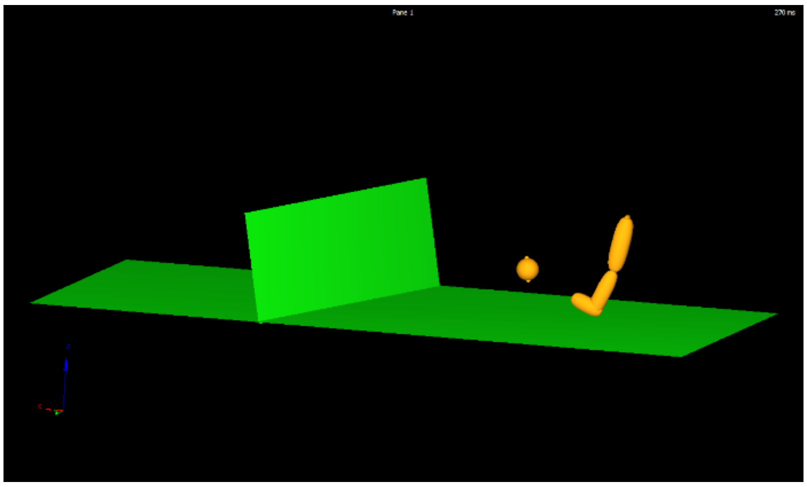
$$A^{21}_{desired}\ (Rotation\ Matrix) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} = \begin{bmatrix} 0.9922 & 0.0000 & -0.1244 \\ -0.0000 & 1.0000 & -0.0000 \\ 0.1244 & 0.0000 & 0.9922 \end{bmatrix}$$

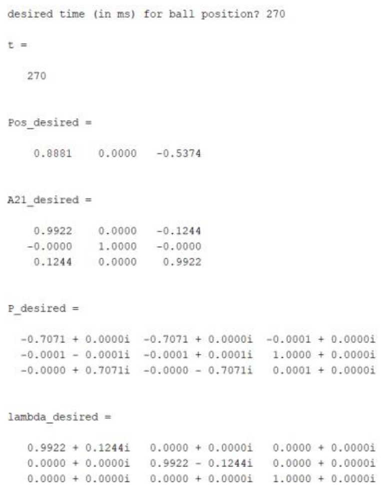
The rotation matrix contains a y-coordinate value of [0, 1, 0] reflective of how the position vector is 0.

$$P_{desired}\ (Eigenvectors) = \begin{bmatrix} -0.0000 - 0.7071\text{i} & -0.0000 + 0.7071\text{i} & -0.0001 + 0.0000\text{i} \\ 0.0001 - 0.0001\text{i} & 0.0001 + 0.0001\text{i} & 1.0000 + 0.0000\text{i} \\ -0.7071 + 0.0000\text{i} & -0.7071 + 0.0000\text{i} & 0.0001 + 0.0000\text{i} \end{bmatrix}$$

$$\lambda_{desired}\ (Eigenvalues) = \begin{bmatrix} 0.9922 + 0.1244i & 0.0000 + 0.0000i & 0.0000 + 0.0000i \\ 0.0000 + 0.0000i & 0.9922 - 0.1244i & 0.0000 + 0.0000i \\ 0.0000 + 0.0000i & 0.0000 + 0.0000i & 1.0000 + 0.0000i \end{bmatrix}$$

$$\phi_{desired}\ (Angular\ Position) = 7.1462°$$





```
desired time (in ms) for ball position? 270

t =

   270


Pos_desired =

    0.8881    0.0000   -0.5374


A21_desired =

    0.9922    0.0000   -0.1244
   -0.0000    1.0000   -0.0000
    0.1244    0.0000    0.9922


P_desired =

  -0.7071 + 0.0000i  -0.7071 + 0.0000i  -0.0001 + 0.0000i
  -0.0001 - 0.0001i  -0.0001 + 0.0001i   1.0000 + 0.0000i
  -0.0000 + 0.7071i  -0.0000 - 0.7071i   0.0001 + 0.0000i


lambda_desired =

   0.9922 + 0.1244i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.9922 - 0.1244i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i
```

Figure 6 & 7: Ball in motion @ 270ms after being kicked and MATLAB output

At t = 850ms:

$$Position\ Vector = [2.2971 \quad 0.0909 \quad -0.6550]$$

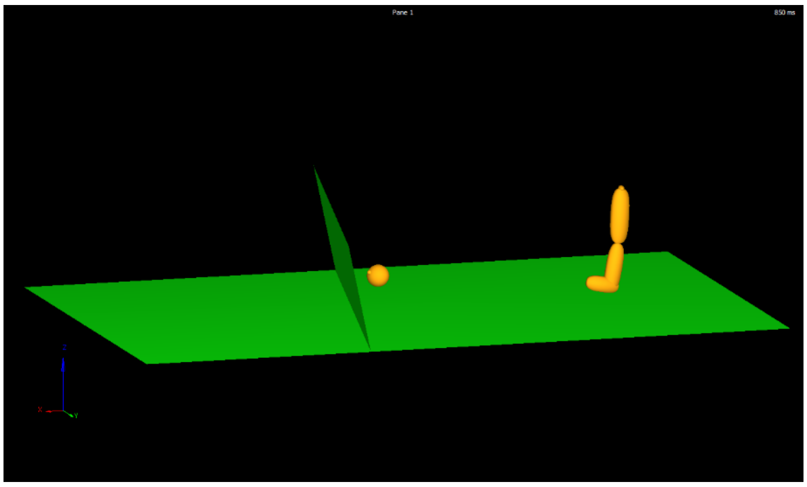
Once the ball has hit the wall, the y-coordinate becomes a non-zero value due to the wall being positioned obliquely to the global coordinate axes. The ball now begins to rotate about a skewed axis.

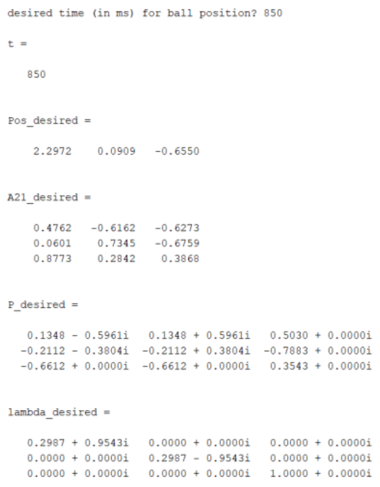
$$A^{21}_{desired}\ (Rotation\ Matrix) = \begin{bmatrix} 0.4762 & -0.6162 & -0.6273 \\ 0.0601 & 0.7345 & -0.6759 \\ 0.8773 & 0.2842 & 0.3868 \end{bmatrix}$$

$$P_{desired}\ (Eigenvectors) = \begin{bmatrix} 0.1348 - 0.5961i & 0.1348 + 0.5961i & 0.5030 + 0.0000i \\ -0.2112 - 0.3804i & -0.2112 + 0.3804i & -0.7883 + 0.0000i \\ -0.6612 + 0.0000i & -0.6612 + 0.0000i & 0.3543 + 0.0000i \end{bmatrix}$$

$$\lambda_{desired}\ (Eigenvalues) = \begin{bmatrix} 0.2986 + 0.9543i & 0.0000 + 0.0000i & 0.0000 + 0.0000i \\ 0.0000 + 0.0000i & 0.2987 - 0.9543i & 0.0000 + 0.0000i \\ 0.0000 + 0.0000i & 0.0000 + 0.0000i & 1.0000 + 0.0000i \end{bmatrix}$$

$$\phi_{desired}\ (Angular\ Position) = 72.6105°$$





```
desired time (in ms) for ball position? 850

t =

   850


Pos_desired =

    2.2972    0.0909   -0.6550


A21_desired =

    0.4762   -0.6162   -0.6273
    0.0601    0.7345   -0.6759
    0.8773    0.2842    0.3868


P_desired =

   0.1348 - 0.5961i   0.1348 + 0.5961i   0.5030 + 0.0000i
  -0.2112 - 0.3804i  -0.2112 + 0.3804i  -0.7883 + 0.0000i
  -0.6612 + 0.0000i  -0.6612 + 0.0000i   0.3543 + 0.0000i


lambda_desired =

   0.2987 + 0.9543i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.2987 - 0.9543i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i
```

Figure 8 & 9: Ball in motion @ 850ms after hitting the wall and MATLAB output

Summary of Results:

These results demonstrate how the eigen-decomposition of the rotation matrix directly translates to describing the orientation of the ball:

- Eigenvector = rotation axis
- Eigenvalue pair (complex) = rotation angle

The two complex conjugates can be used to solve the rotation angle. The MADYMO software enabled us to solve this problem at each timestep free from any issues such as gimbal lock. This is because behind the scenes, in MADYMO and the MATLAB scripts, Euler-Rodrigues (quaternions) were used to derive correct numerical solutions at each stage.