Toronto Metropolitan University

Department of Mechanical, Industrial, and Mechatronics Engineering
Faculty of Engineering & Architectural Science

Please select your current program below:

- **Mechanical Engineering**
- **Industrial Engineering**
- **Mechatronics Engineering**

Course Number	MT8130
Course Title	Robot Mechanics
Semester/Year	1
Section Number	N/A
Group Number	N/A

Final Project Report

Assignment Title	Cosserat-Rod Tendon-Driven Robot
Submission Date	11/25/2025
Due Date	11/25/2025

Student Name	Student ID (xxxx1234)	Signature*
Benjamin Chung	501402427	Ben

(Note: Remove the first 4 digits from your student ID)

**By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: https://www.torontomu.ca/senate/policies/academic-integrity-policy-60/.*

Contents

Abstract

This report will document the modeling and control of a Tendon-Driven Continuum Robot (TDCR). These robots exhibit soft joints with large deflection capabilities. The modeling is done using the Cosserat-Rod Theory for high accuracy in bending as the robot is actuated. As the system is highly non-linear, the control system is designed using a Sliding Mode Controller.

The overall results are successful. The model developed is relatively robust and stable at low tensions. However, the model suffers when big disturbances are applied. The control system tracks a point decently well with the exception for large ”chatter”.

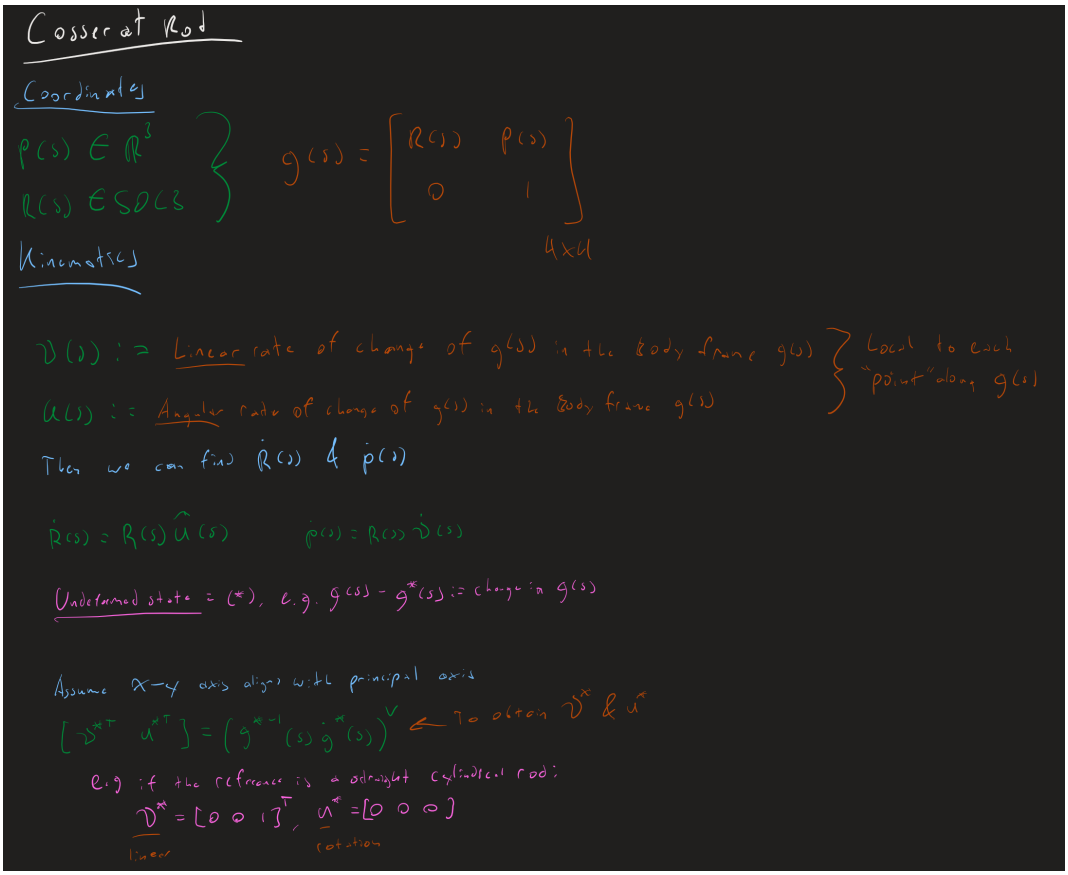
1 Introduction

Soft robotics has been an new and promising field of robotics. With many rigid-link robots, they are limited greatly by joint limits, weight, and size. Soft robots attempt to break these barriers while delivering the same performance. An example of a sot robot is a continuum robot.

This report will document the modeling and control of a Tendon-Driven Continuum Robot (TDCR). The modeling is done using the Cosserat-Rod Theory for high accuracy in deflection. As the system is highly non-linear, the control is designed using a Sliding Mode Controller.

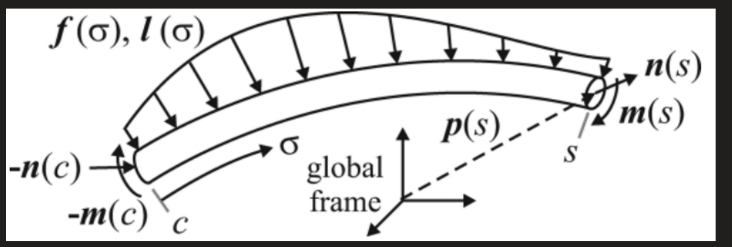
2 Cosserat Rod Theory Modeling

The following is the derivation of the Cosserat Rod Theory. As the dynamics of the system are intrinsic to the forward and inverse kinematic modeling, everything regarding modeling will be in this section for organization purposes. Do note that "inverse kinematics" is not possible for the Cosserat-Rod Model.



Equilibrium

n := internal force vector } global frame
m := moment vector
f := applied force distribution per unit s
l := applied moment distribution per unit s



So we find: after taking the derivative

$$\dot{n}(s) + f(s) = 0 \qquad \dot{m}(s) + \dot{p}(s) \times n(s) + l(s) = 0$$

Constitutive Laws (relate material properties to rod structure)

Various modes of strain:

$v_x - v_x^*$, $v_y - v_y^*$ } Transverse shear in x-x

$v_z - v_z^*$ } axial elongation in z

$u_x - u_x^*$, $u_y - u_y^*$ } bending about local x-x

$u_z - u_z^*$ } torsion about local z

So now we can make linear constitutive laws:

$$n(s) = R(s)\,K_{se}(s)\,(v(s) - v^*(s))$$
$$m(s) = R(s)\,K_{bt}(s)\,(u(s) - u^*(s))$$

Add viscous damping from Sharifi et al

$$n = R\,K_{se}\,(v - v^* + B_{se}\,v_t)$$
$$m = R\,K_{bt}\,(u - u^* + B_{bt}\,u_t)$$

Where:

$$K_{se} = \begin{bmatrix} GA(s) & 0 & 0 \\ 0 & GA(s) & 0 \\ 0 & 0 & EA(s) \end{bmatrix}$$ Shear / Extension

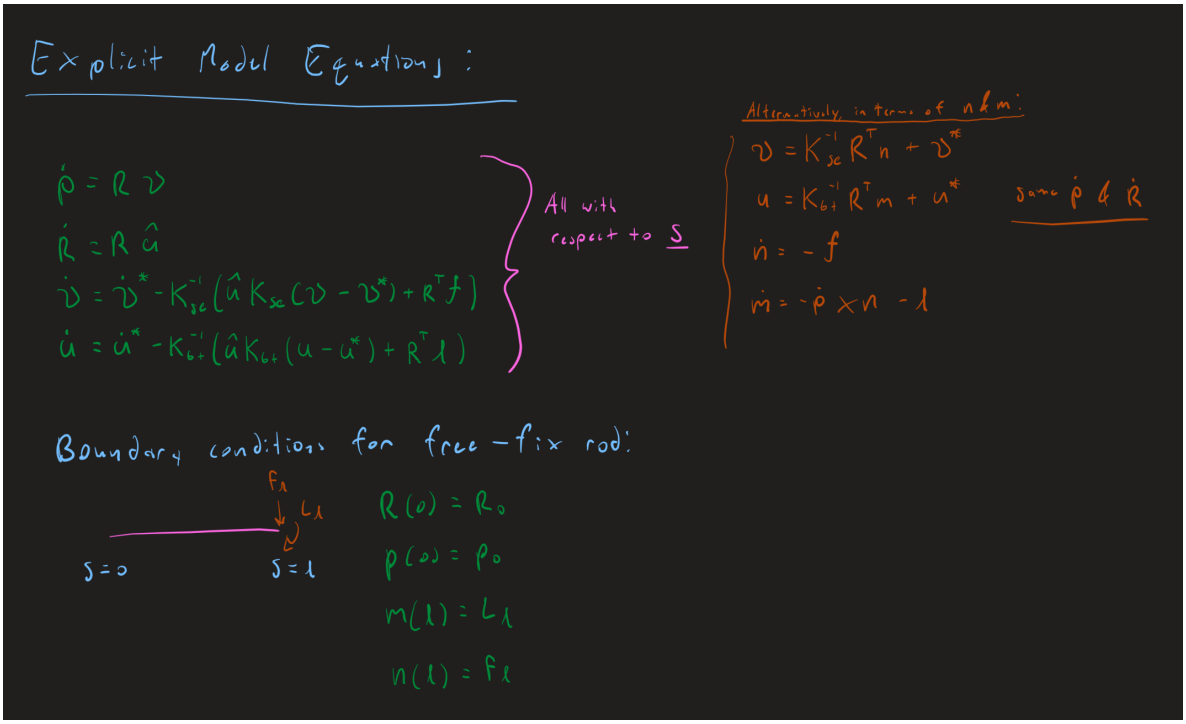
$$K_{bt} = \begin{bmatrix} EI_{xx} & 0 & 0 \\ 0 & EI_{yy} & 0 \\ 0 & 0 & EI_{zz} \end{bmatrix}$$ Bending / Torsion

$$B_{se} \approx \begin{bmatrix} B_{se} & 0 & 0 \\ 0 & B_{se} & 0 \\ 0 & 0 & B_{se} \end{bmatrix}$$

Then roughly the same for B_{bt}

$A(s)$:= area of cross-section
$E(s)$:= Young's modulus

I → Second moment of inertia



These resulting formulas make up the kinematics of the center ”back-bone” of the Robot. While we can use these formulas to roughly understand the kinematics of the robot itself, we still need a way to control it. Next, I will discuss the modeling of the tendon wires that will be used to apply forces along the Cosserat-Rod, and thus change its shape.

2.1 Tendon Modeling

For the tendon Modeling we need to make some important assumptions to simplify the model:

- Friction between the tendons and the channels in the disks is 0.
- Tension throughout the tendons is constant from $0 \rightarrow L$.
- The location of the tendons within the cross-section does not change.

With these assumptions established, we can try and consider the applied tendon forces as an applied load across the whole rod.

$$f = f_e + f_t$$
$$l = l_e + l_t$$

Where the subscript e and t represent the external and tendon loads respectively.

To describe the tendon routing path, the tendon location is defined in the local frame of the cross-sections as a function of S.

Therefore, the i^{th} tendon location is expressed by $x_i(s)$ and $y_i(s)$ to obtain the body-frame local coordinates of the tendon at each disk frame.

Then a vector from the backbone attached frame to the tendon can be given by;

$$r_i(s) = [x_i(s)\ y_i(s)\ 0]^T$$

The resulting parametric space curve before and after deformation is then given by:

$$P_i^* = R^*(s)r_i(s) + P^*(s)$$
$$P_i = R(s)r_i(s) + P(s)$$

Where the star (*) denotes the variable before deformation.

2.2 Distributed tendon Forces

Now that we can identify where the tendons are located all along the backbone, we can now start develop a model to apply these loads towards the backbone.

Consider the derivative of the static equilibrium equations for a finite section.

f_i : Distributed force applied to the i^{th} tendon

n_i : Internal force applied the tendon

$$\dot{n}_i(s) + f_i(s) = 0$$

* We consider the tendons to be ideal strings s.t. they are perfectly flexible.

↳ Cannot support internal moments or shear forces.

∴ Only Tensions γ_i

The internal force n is always tangent to the curve $p(s)$

Therefore we find:

$$n_i(s) = \gamma_i \frac{p(s)}{\|p(s)\|}$$

Solve for f_i

$$\dot{n}(s) + f_i(s) = 0$$

$$f_i(s) = -\dot{n}(s) = \gamma_i \frac{\hat{p}^2}{\|p\|^3} p_i$$

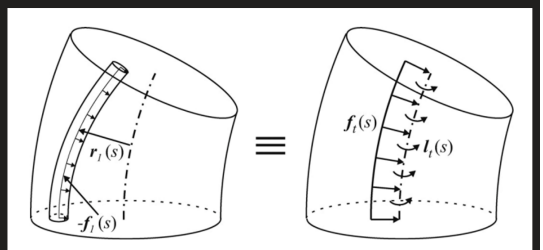


Fig. 5. A small section of rod that shows how the force distribution that the tendon applies to its surrounding medium is statically equivalent to a combination of force and moment distributions on the backbone itself.

Tendon Loads on the Backbone

The total distributed force f_t is:

$$f_t = -\sum_{i=1}^{n} f_i$$

The total distributed moment at the backbone centroid is the sum of cross-products of moment arm $\times f_i$

$$l_t = -\sum_{i=1}^{n} (p_i - p) \times f_i = -\sum_{i=1}^{n} (R r_i)^{\wedge} f_i$$

($p_i - p$: moment arm; $R r_i$: rotation $\times$ tendon location; $\wedge$: hat op for cross product)

Then we sub in our f_i from earlier:

$$f_t = -\sum_{i=1}^{n} \gamma_i \frac{\hat{\dot{p}}_i^2}{\|\dot{p}_i\|^3} \ddot{p}_i$$

$$l_t = -\sum_{i=1}^{n} \gamma_i (R r_i)^{\wedge} \cdot \frac{\hat{\dot{p}}_i^2}{\|\dot{p}_i\|^3} \ddot{p}_i$$

These formulas need to be in terms of the kinematic variables u v R p in order to be substituted back into our ODEs from before:

We differentiate p_i and get:

$$p_i = R r_i + p$$

$$\dot{p}_i = R \hat{u} r_i + R \dot{r}_i + R v = R(\hat{u} r_i + \dot{r}_i + v)$$

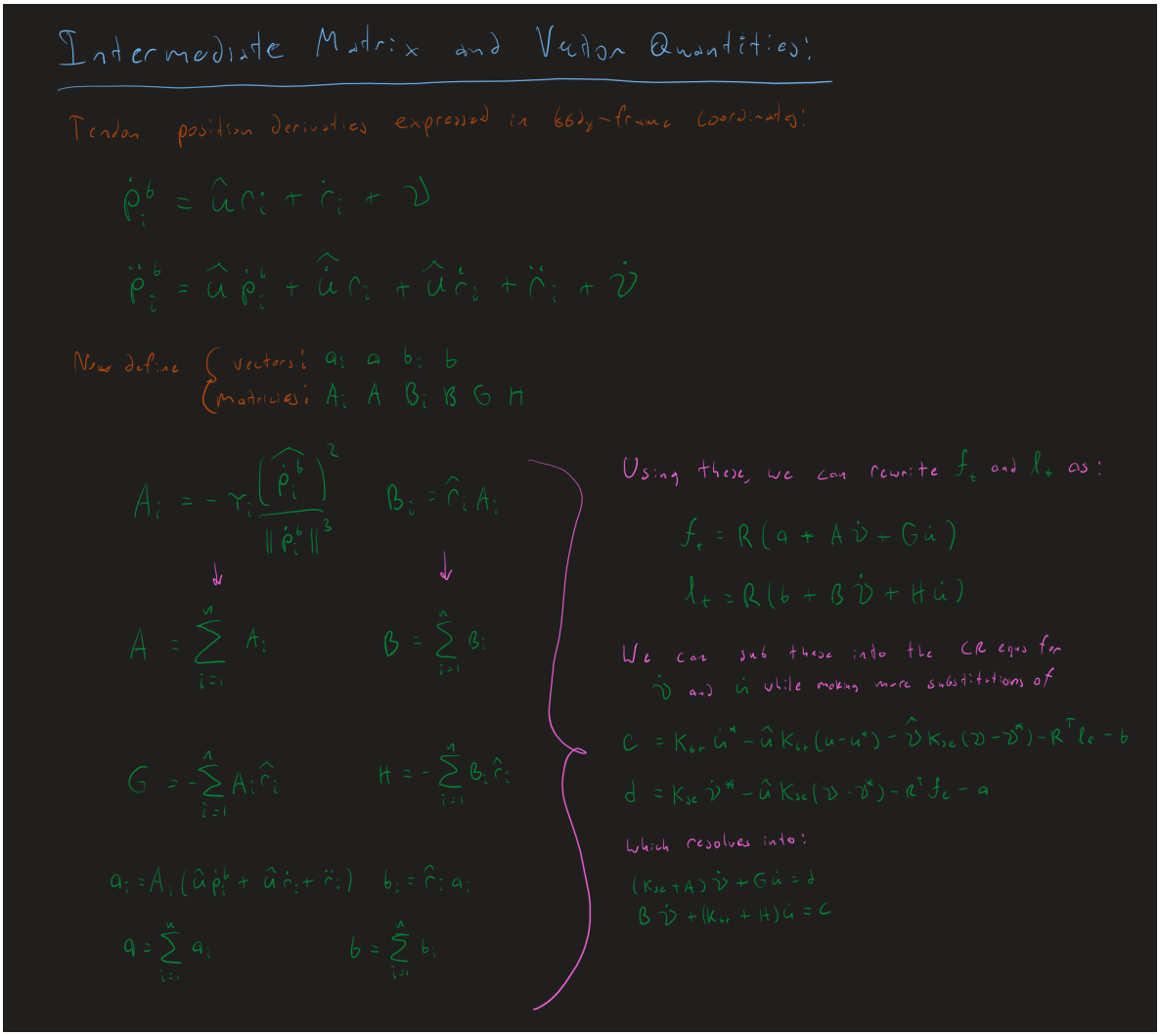
$$\ddot{p}_i = R\left(\hat{u}(\hat{u} r_i + \dot{r}_i + v) + \hat{\dot{u}} r_i + \hat{u} \dot{r}_i + \ddot{r}_i + \dot{v}\right)$$

Now we can look to sub in the tendon kinematics into the Cosserat Rod model.

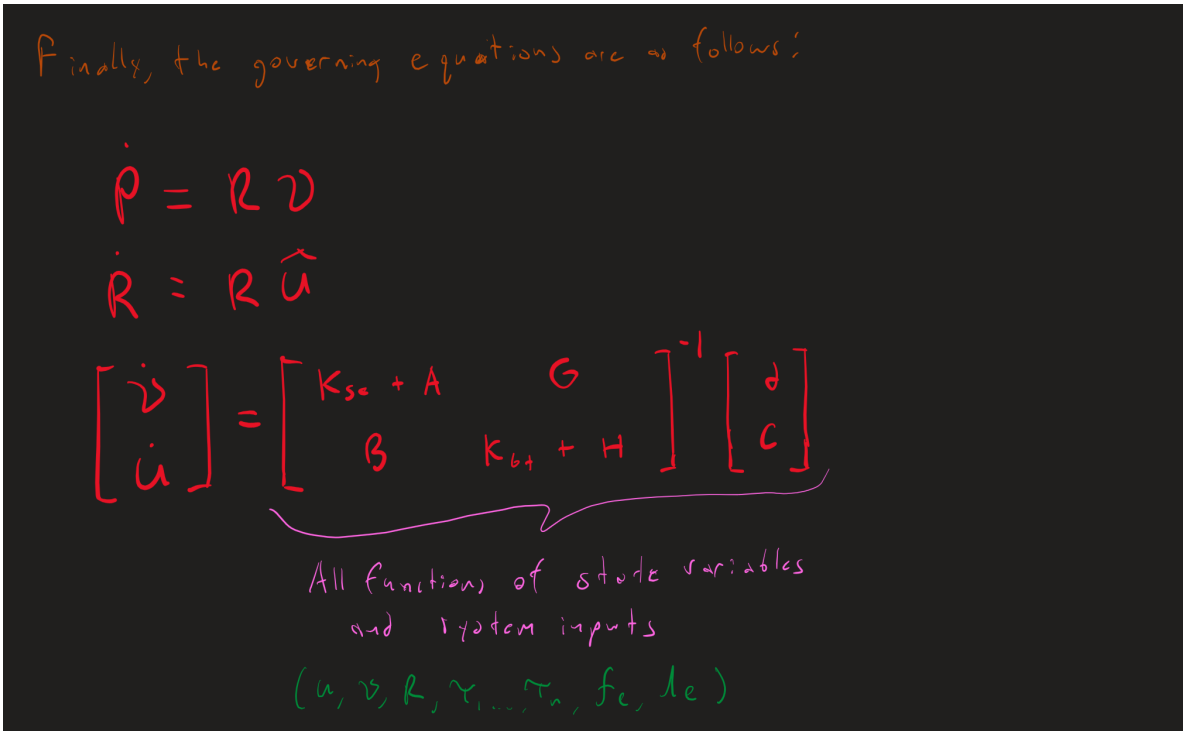
2.3 Explicit Model Equations

In the previous section, the coupled rod and tendon model was formulated in an implicit form. In order to make them easier to work with, we will now work them into an explicit, first-order, state-vector form.

To start with, we will create some variable substitutions for ease of future notation.



Finally we arrive at this result:



2.4 Boundary Conditions

Before we starting working out the dynamics, there are some boundary conditions to consider. As this will be a fixed-free-end model, we can easily assume that one end has 0 applied forces. The free end is considered as shown below:

Boundary Conditions

When tendon i terminates at point $s = l_i$ a point force is applied at its attachment point equal and opposite to the internal tendon force.

$$F_i = -n_i(l_i) = -\Upsilon_i \frac{\dot{p}_i(l_i)}{\|\dot{p}_i(l_i)\|}$$

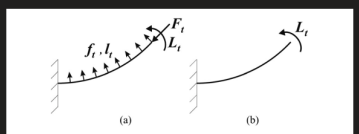


Fig. 6. (a) The coupled Cosserat rod and tendon approach includes all of the tendon loads. These loads are themselves functions of the robot shape, and therefore, the robot is treated as a coupled system. (b) The point moment approach only includes the attachment moment. For planar deformations, the two approaches predict similar robot shapes (see Fig. 1), but our experimental results in Section V show that for out-of-plane loading, the coupled approach is more accurate.

Then the moment at $s = l_i$ is:

$$L_i = -\Upsilon_i (R(l_i) r_i(l_i))^{\wedge} \frac{\dot{p}_i(l_i)}{\|\dot{p}_i(l_i)\|}$$

Arbitrary point loads:

Due to tendon termination or external loads

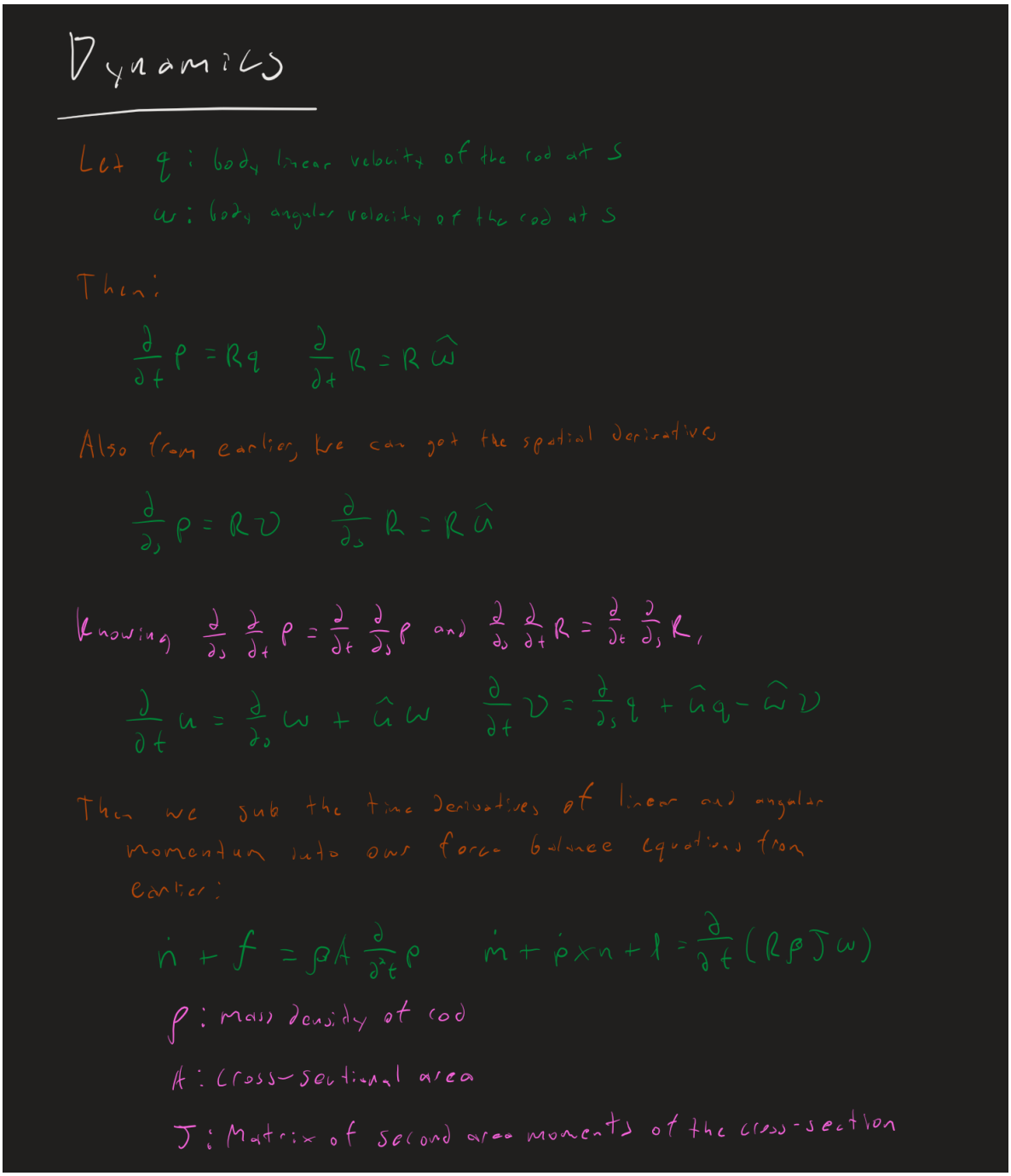
For some point $s = \sigma$, point loads $F(\sigma)$ $L(\sigma)$ are applied to the boundary and the internal force and moment change across the boundary at $s = \sigma$ by:

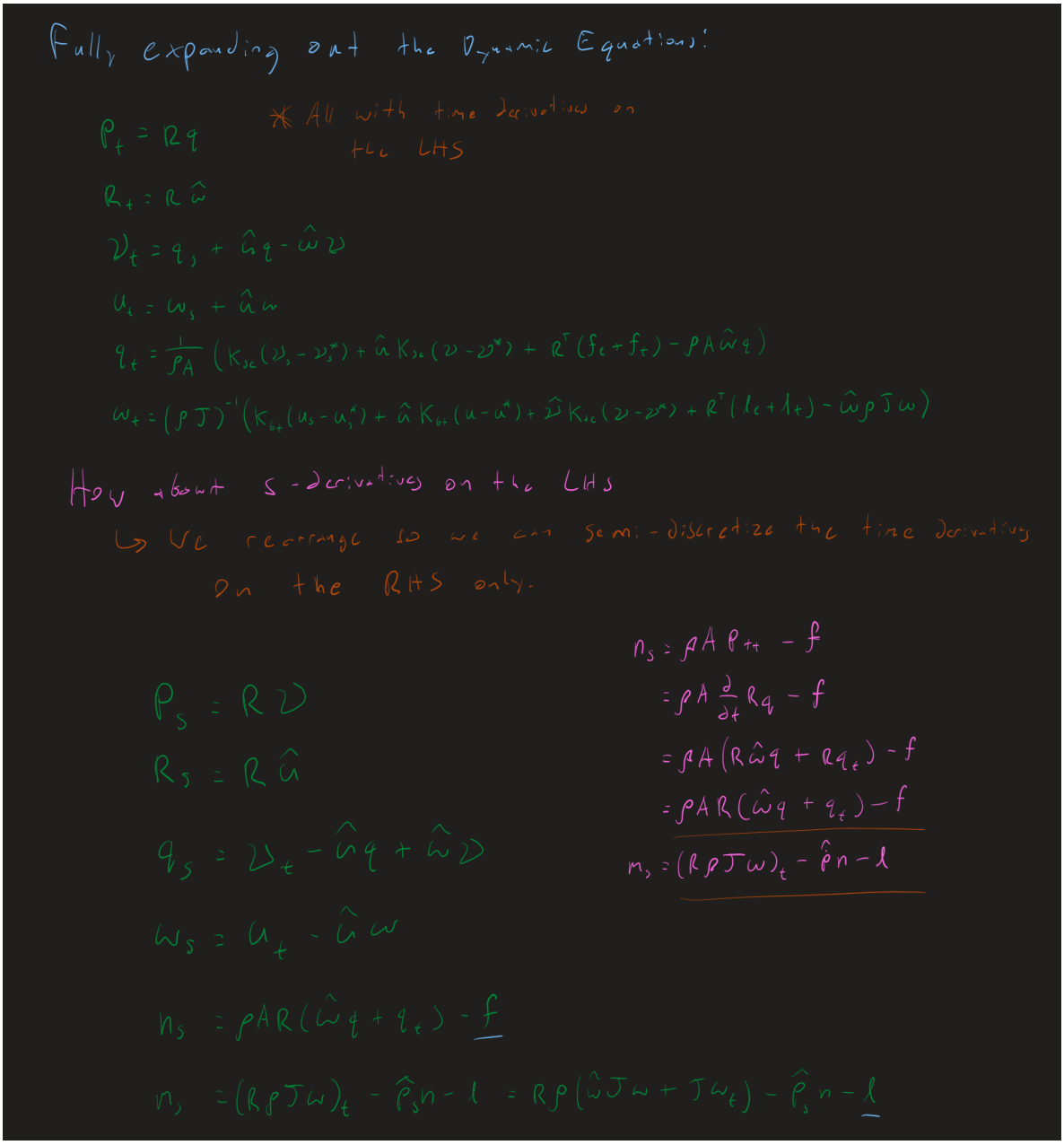
$$n(\sigma^-) = n(\sigma^+) + F(\sigma)$$
$$m(\sigma^-) = m(\sigma^+) + L(\sigma)$$

$+$ or $-$ indicate after and before $s = \sigma$

2.5 Dynamics

This sections will now go over the dynamic equations of the cosserat-rod whilst also considering the tendon forces.

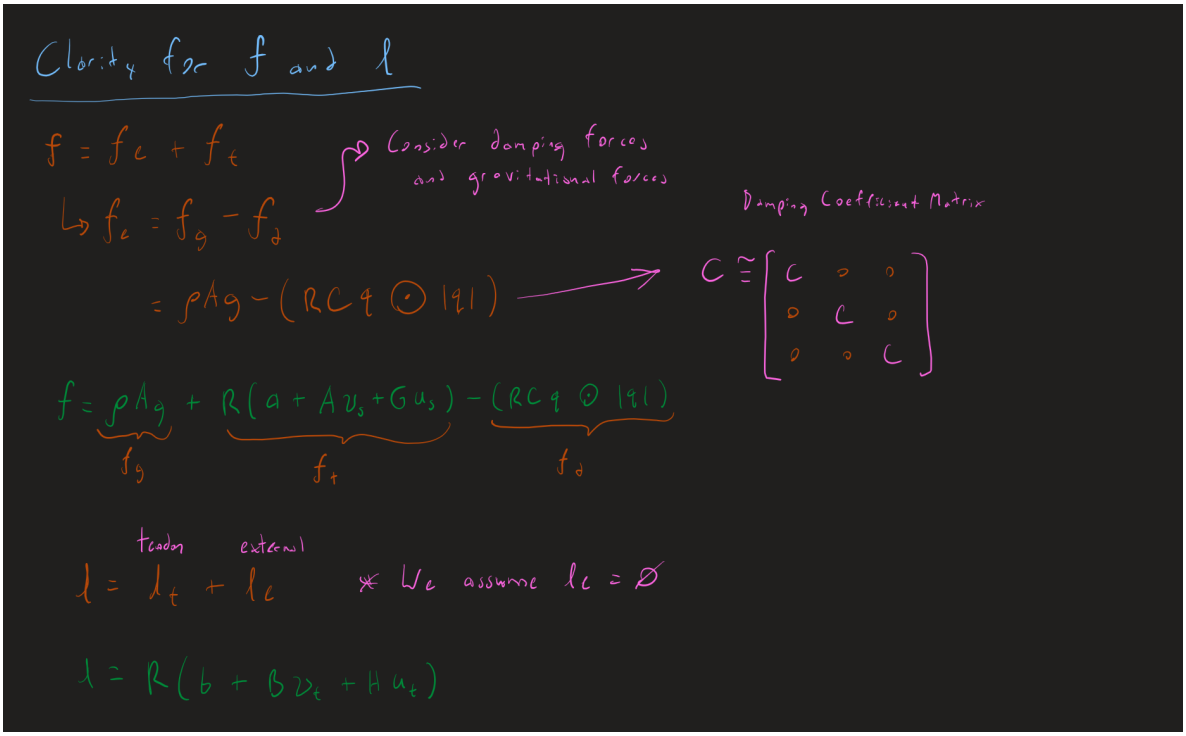




With these final equations, we can now see that they result in a set of Partial Differential Equations (PDE) in terms of spacial derivatives $\frac{d}{ds}$ and time derivtives $\frac{d}{dt}$. Generally this is difficult to solve, however, with some simplification strategies, the equations are still solvable. Such a strategy is considered in the next section.

2.6 Applied forces and moments

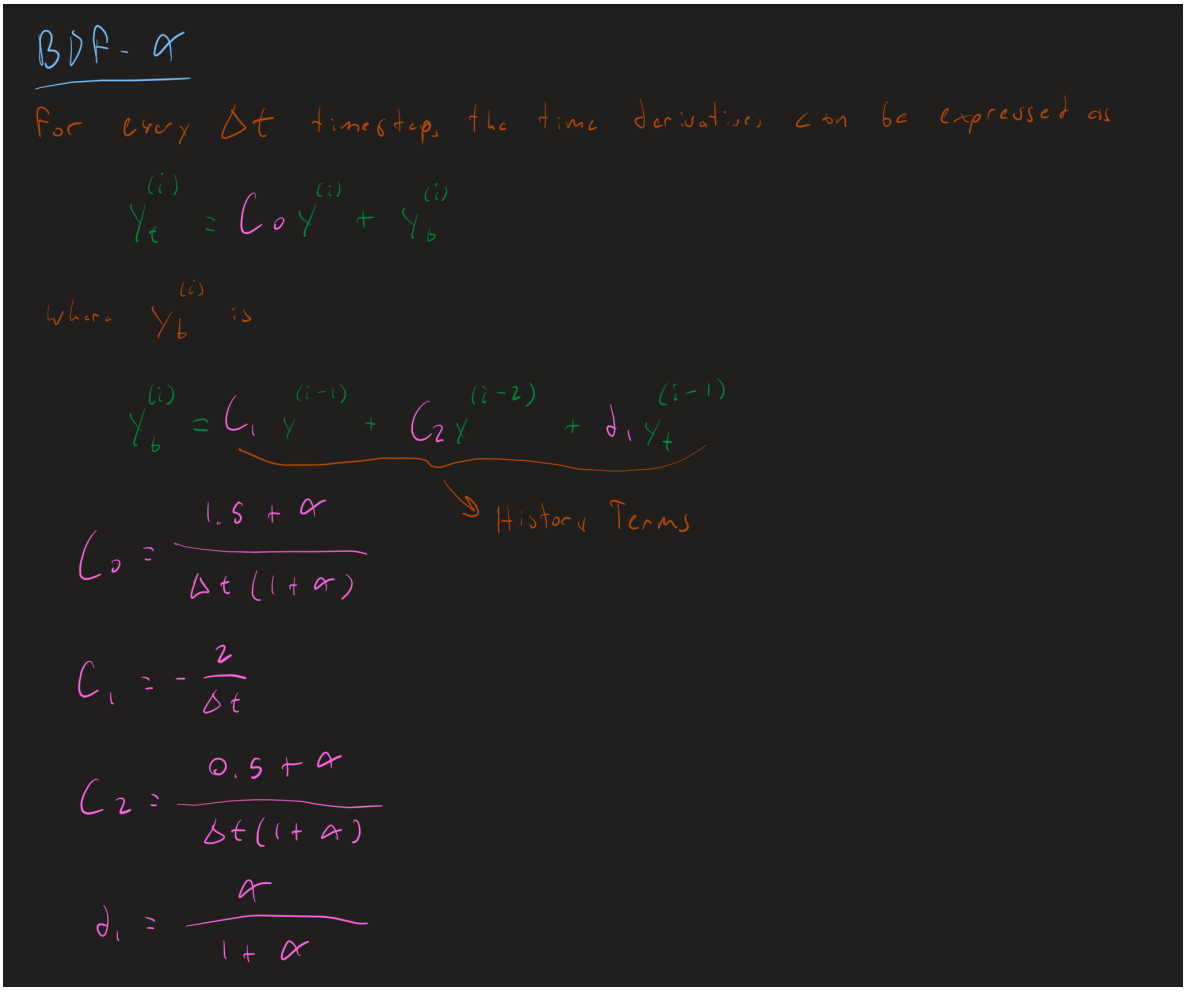
The following is some clarity on the extra f and l terms applied along the rod.



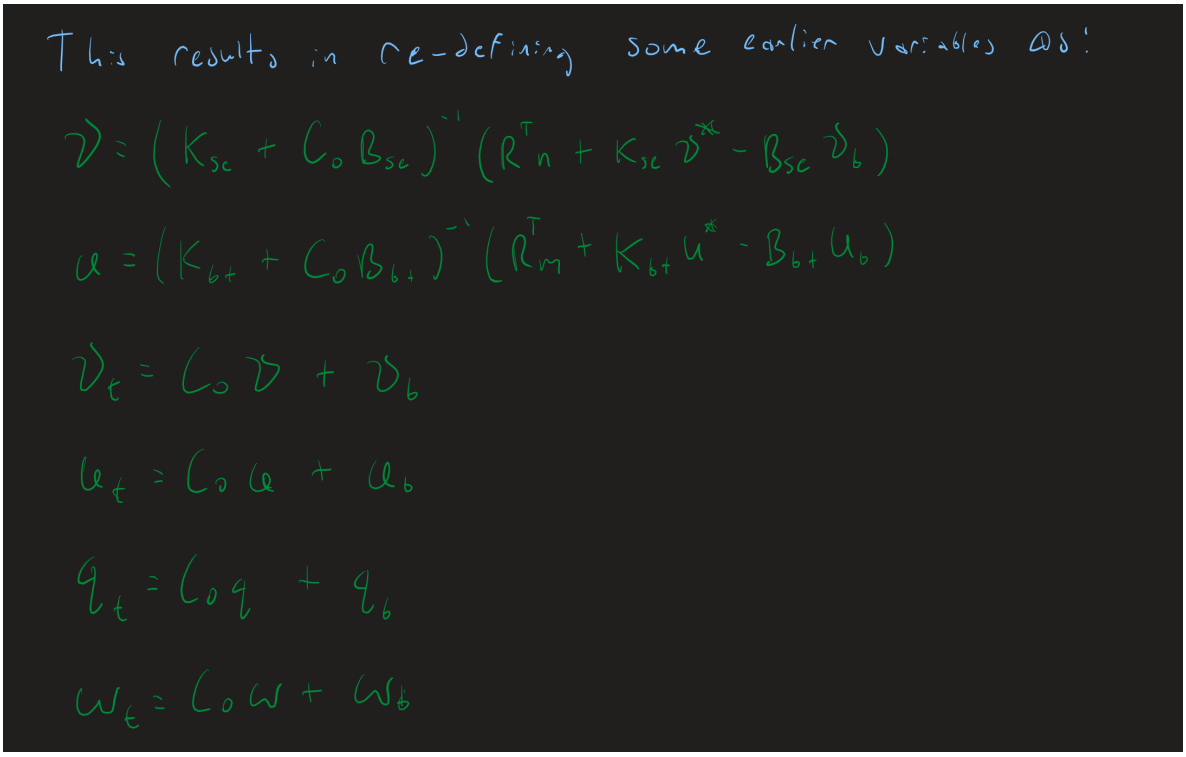
2.7 Semi-discretizations of the Time Derivatives

While PDEs can be very difficult to deal with, Ordinary Differential Equations (ODE) are much easier. By semi-discretizing the time derivatives, we can turn the PDEs into solvable ODEs.

The technique used in this project is known as Backwards-Difference-Alpha (BDF-α). The time derivatives are simply turned into averages over "history terms" stored in memory as the solution develops. the α term is used as a factor in the following set of equations in order to adjust stability.



This results in our ability to rewrite our time-derivatives as follows:



3 Simulation Results

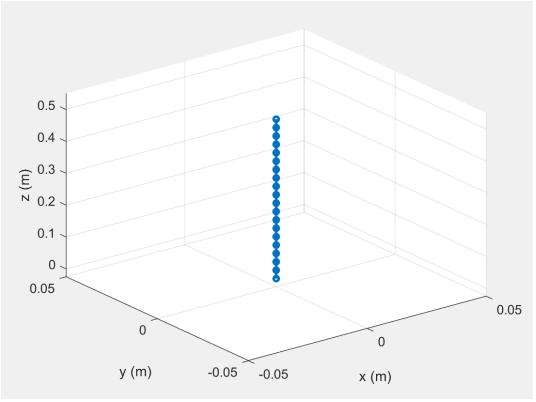
This section will demonstrate the forward kinematic simulation of the TDCR. The following are the relevant parameters that were arbitrarily chosen. Do note that these values can be easily calibrated for hardware experiments:

Parameter	Value
Length	0.5
Spatial Resolution	20
Young's Modulus	190e9
Cross-Section Radius	0.001
Tendon Distance from Spine	0.01
Density	17189
Shear/Bending Coefficient	0
Bending/Torsion Coefficient	0.008
Viscous Damping Coefficient	0.01
Timestep Size	0.005
BDF Parameter	-0.2
Number of Time Steps	100

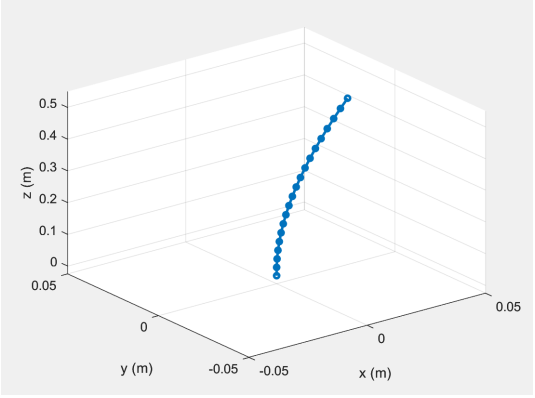
*Note: All in SI units

3.1 Forward Kinematics

The first figure showcases the rod's deflection in 3-D Space. The following figures will show the displacement due to a constant applied tension force of $3N$. Do note that all deflection is in the x-z direction as it is only being controlled by a single tendon.



(a) Starting configuration @ [0; 0; 0.5]



(b) Deflection @ [0.027; 0; 0.4727]

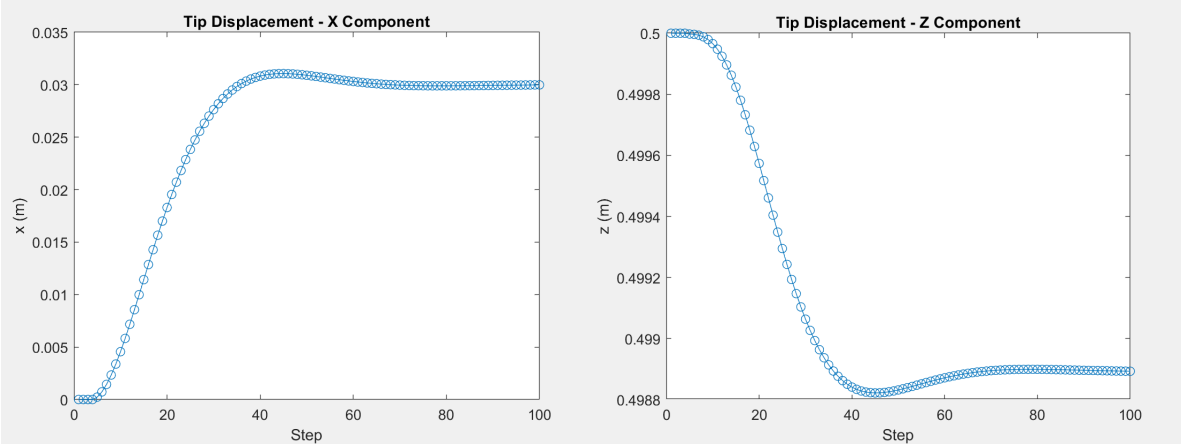


Figure 2: X-Z Tip Displacement

3.1.1 Singularities

As the TDCR has a theoretical infinite degrees of freedom, its singularities might not be immediately evident. The most obvious one is when it's at its upright configuration of [0; 0; L] due to robot losing its ability to travel in the upwards Z direction. Other singularities lie within the inversion of a matrix.

As per [1], the inversion of Φ is necessary in order to find the shear and transverse strains within the model.

$$\begin{bmatrix} \mathbf{v}_s \\ \mathbf{u}_s \end{bmatrix} = \mathbf{\Phi}^{-1} \begin{bmatrix} \mathbf{\Pi}_n - \mathbf{\Sigma}_n \\ \mathbf{\Pi}_m - \mathbf{\Sigma}_m \end{bmatrix},$$
$$\mathbf{\Phi} = \begin{bmatrix} \mathbf{K}_{se} + c_0\mathbf{B}_{se} + \mathbf{A} & \mathbf{G} \\ \mathbf{G}^T & \mathbf{K}_{bt} + c_0\mathbf{B}_{bt} + \mathbf{H} \end{bmatrix}.$$

During my implementation, this matrix inversion would suffer singularities at steady state, however, a simplification was offered in a code base from [1] which simplifies this inversion to that of a 2x2 block matrix which greatly increases the stability of the overall system.

3.2 Dynamics

The following is the tip Velocities and Accelerations throughout the entire simulation. As mentioned earlier, a single tendon in the positive X directon is subjected to a constant tension of $3N$.

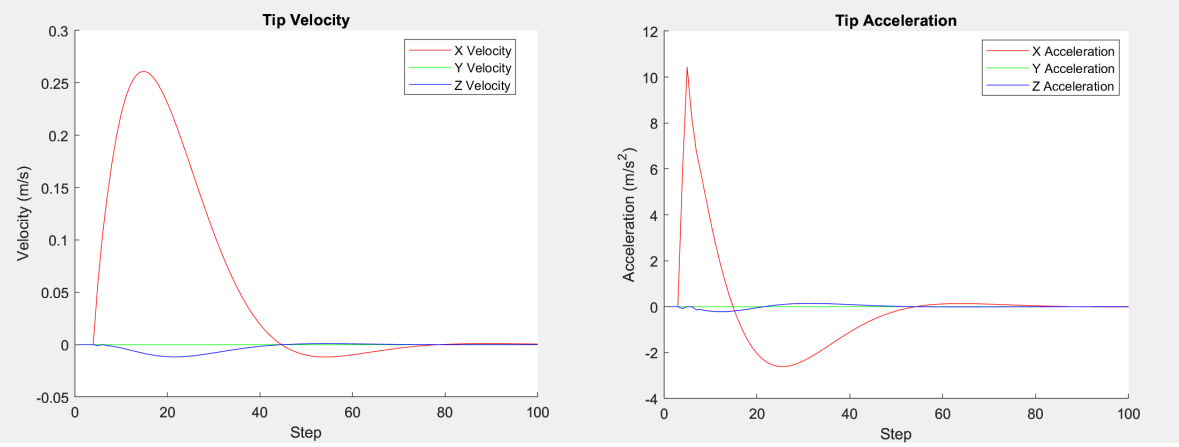
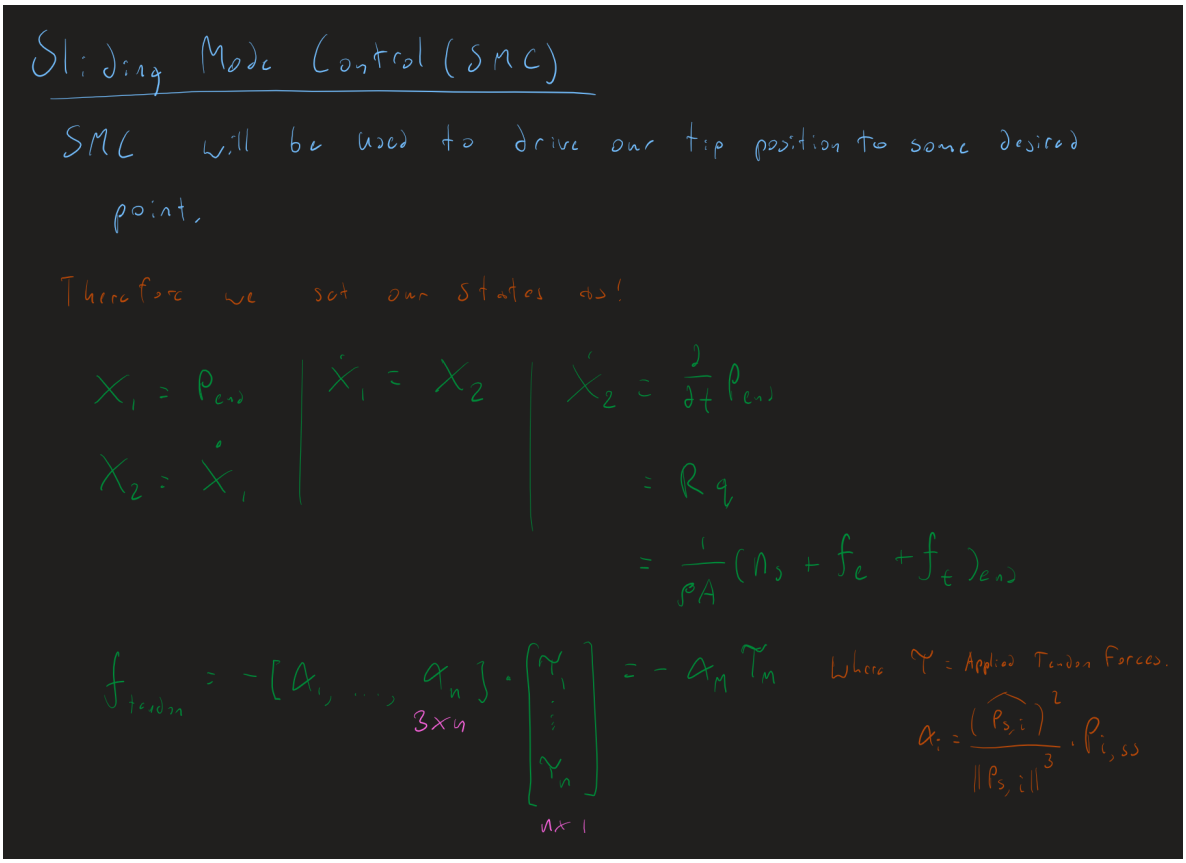


Figure 3: Tip Velocities and Accelerations

4 Controls

To control a Coserat-Rod based system, you will need methodologies from non-linear controls. As a result, I have chosen to use Sliding Mode Control (SMC). Additionally, there are resources available that suggest the effectiveness of SMC for controlling a TDCR.

4.1 Derivation



Then the state-space is:

$$X_1 = P_{cmd}$$
$$X_2 = \dot{X}_1 = (Rq)_{cmd}$$

$$\dot{X}_1 = X_2$$
$$\dot{X}_2 = \frac{1}{\rho A}(n_s + f_e)_{cmd} - \frac{1}{\rho A}(\alpha_M)\cdot \underline{T_M}$$

$T_M = U$ — As we want to control the tendon forces.

Now for the sliding-mode-control, we want want to drive some position and velocity error to zero.

$$e = X_d - X_1$$
$$\dot{e} = \dot{X}_d - \dot{X}_1$$

We identify our sliding surface as:

$$S = \dot{e} + Ce, \quad C > 0$$

We also use the exponential reaching law:

$$\dot{S} = -\varepsilon\, \mathrm{sgn}(S) - KS, \quad K > 0$$

Now to apply to our sliding surface:

$$\dot{S} = \ddot{e} + C\dot{e}$$
$$= \ddot{X}_d - \ddot{X}_1 + C(\dot{X}_d - \dot{X}_1)$$
$$= \ddot{X}_d - (a_c + b_c U) + C(\dot{X}_d - \dot{X}_1)$$

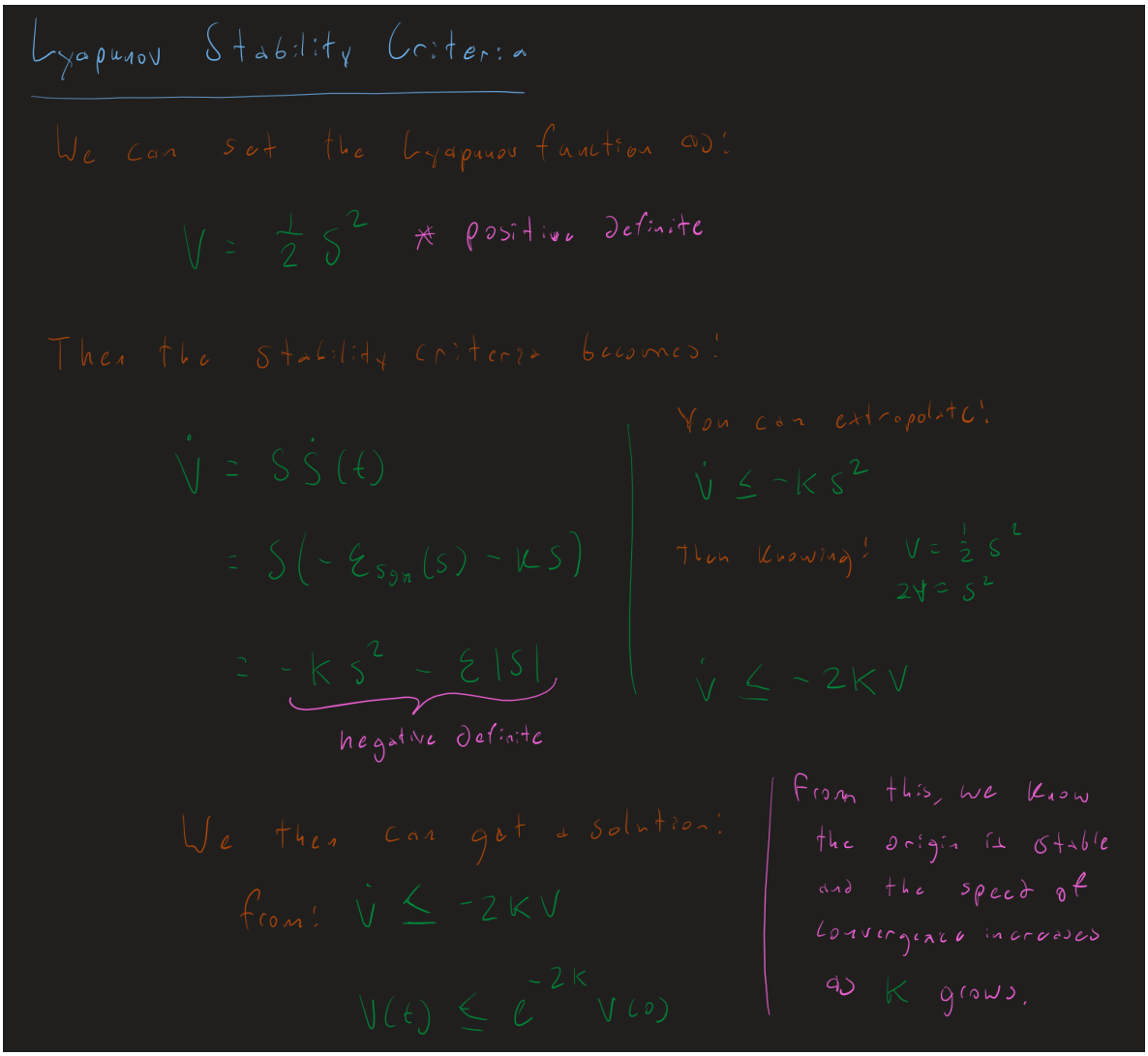
To equate with $\dot{S}$ from earlier & isolate for U gives:

$$U = \frac{1}{b_c}\left(C(\dot{X}_d - \dot{X}_1) + \ddot{X}_d - a_c + \varepsilon\, \mathrm{sgn}(S) + KS\right), \quad \varepsilon, K > 0$$

where:

$$a_c = \frac{1}{\rho A}(n_s + f_e), \quad b_c = -\frac{1}{\rho A}(\alpha_M), \quad U = T_M$$

4.2 Lyapunov Stability



4.3 Control Gains and Tuning

Through experimentation, the gains were selected as follows:

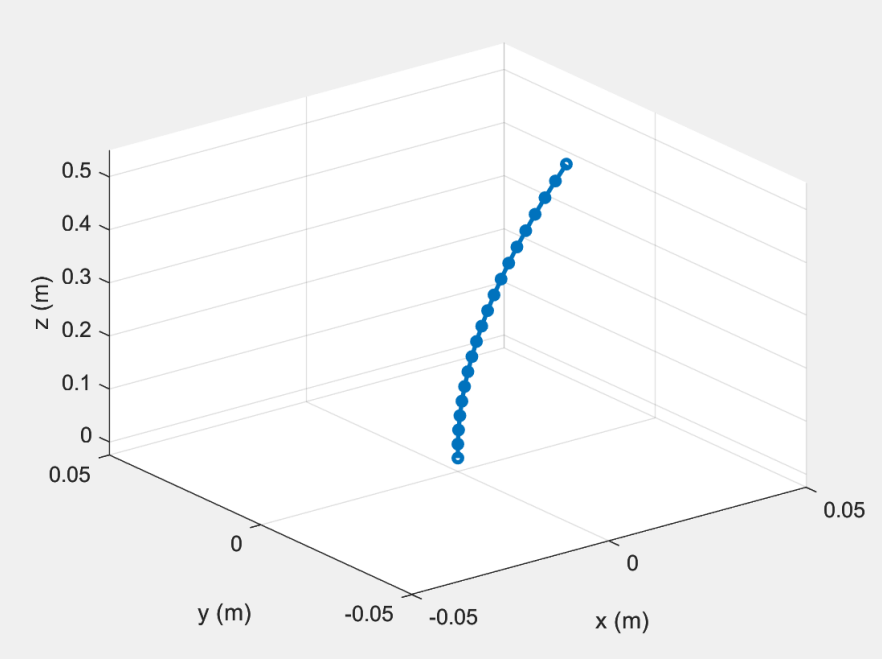
$$\mathbf{C} = \begin{bmatrix} 15000 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 500 \end{bmatrix} \quad \mathbf{k} = \begin{bmatrix} 30 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

Notably, to ensure convergence of our X error, the gains needed to be rather large. the Gain for the Y error is 1 in this scenario as there was no deflection into the Y axis. For multi-tendon systems, this gain would need to be considered. The gain for the Z error is significantly less due to the fact that the rod deflection was small enough that its Z location changed very little.

The strategy for tuning gains C versus k were notably different. Gain C helps to drive our robot to the sliding plane while k affects the speed at which it arrives there (as per our Lyapunov analysis). Due to the stiffness of the model, gains C had to be much larger in order to converge while k could be kept relatively small in order to avoid overshoot and oscillations.

4.4 Simulation Results

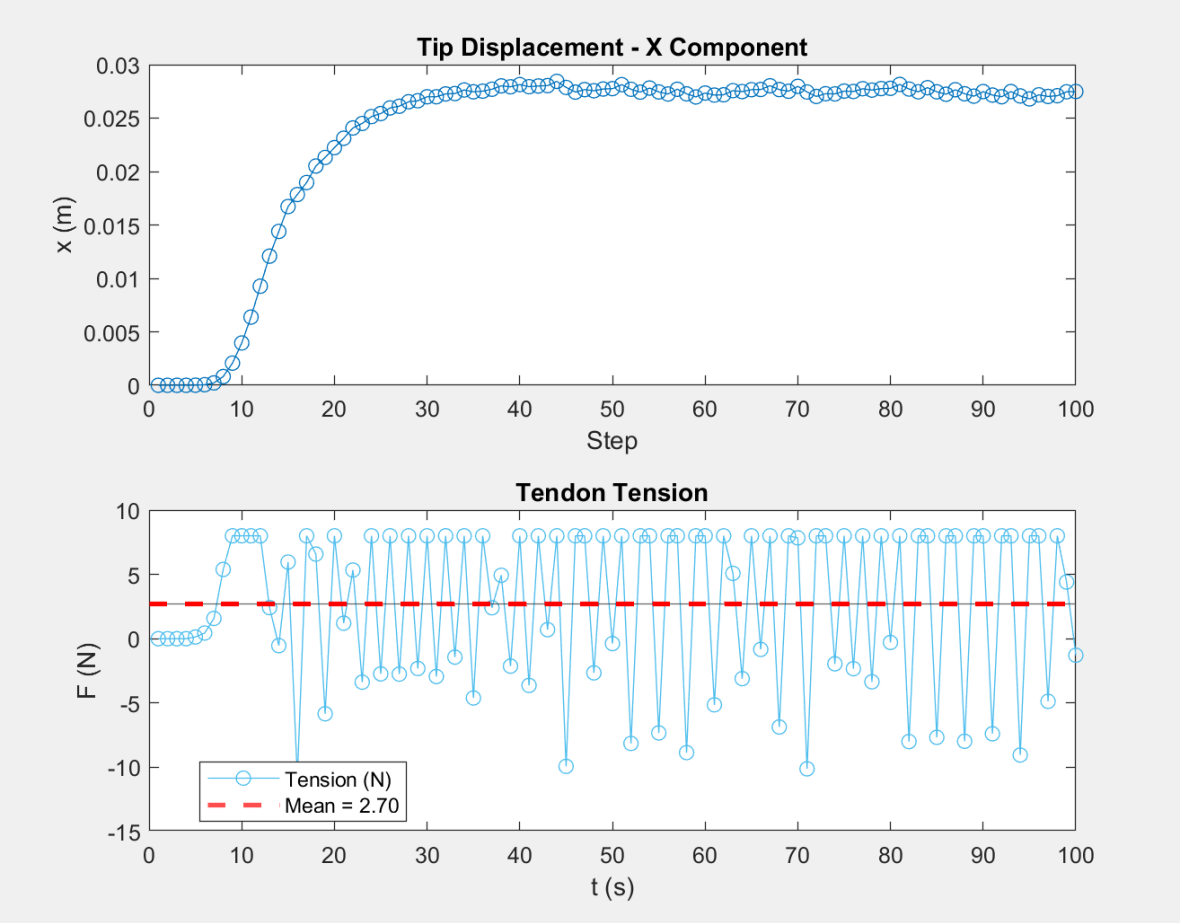
The following image is the resulting location of the TDCR:



In order to verify my control system, I input the desired location to be the same as the forward kinematic experiment.

While the figure looks very similar to our forward kinematic solution prior, its tip position is $[0.0247; 0; 0.4729]$ which is very close.

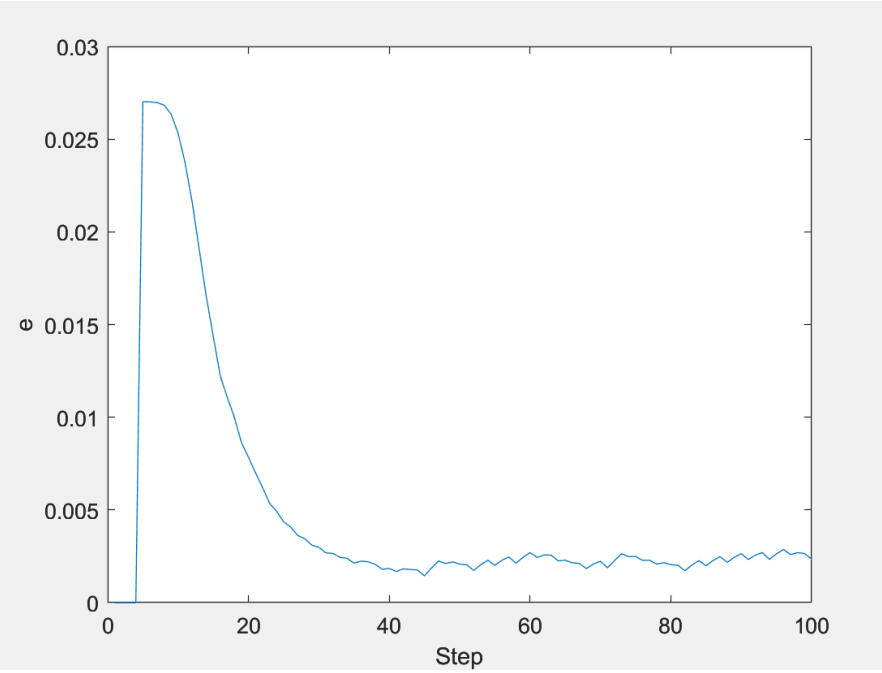
Furthermore this resulted in nearly $3N$ of force. As seen below, there is an immense amount of chattering in the system, causing not only the tip location to be a bit unstable, but for the tendon force to also be wildly unstable.



This is one of the problems with SMC, however, it can be mediated. A simple mediation is to adaptively shrink the gains near the sliding surface, or to change the behaviour of the sliding surface as the error approaches a certain threshold. For now, that is out of the scope of this project.

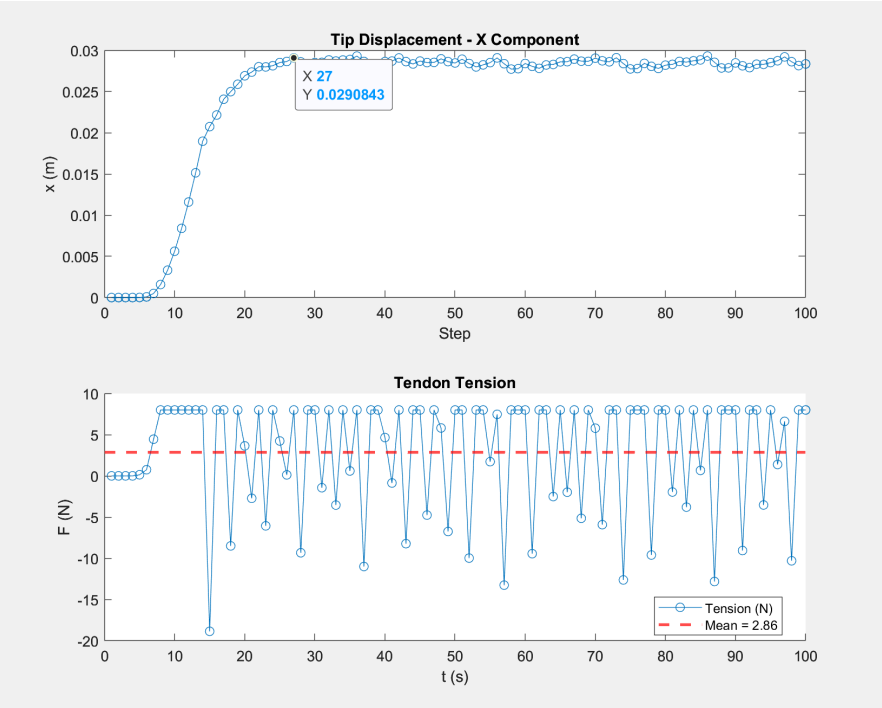
4.5 Error Analysis

As you can see, the controller does a relatively good job at reducing the positional error. The ”buzziness” observed at steady state is due to the ”chattering” behaviour of the SMC.



4.6 Faster Trajectories

As discussed earlier, increasing the gain k can possibly lead to faster convergence, however, not without overshoot or oscillations.



Evidently, increasing $k = 30 \rightarrow 60$ somewhat reduces the convergence to roughly 30 time-steps, but the overshoot and unstable behaviour at steady-state is much more rampant.

5 Conclusion

To reiterate, the goal of this project was to model and control a Tendon-Driven Cosserat-Rod Continuum Robot. The model proved to be difficult to handle, however, quite accurate. The solving method is not the most efficient, but for the sake of a simulator it is good enough.

The Sliding Mode Controller proved to work decently well, however, with the effects of chattering, there is much left to be desired. There are ways to reduce chattering, and should be investigated in the future. The gains could also have been tuned more effectively and efficiently if more time was permitted.

6 Sources

References

[1] Rana Danesh and Farrokh Janabi-Sharifi. Backstepping control of tendon-driven continuum robots in large deflections using the cosserat rod model. *Mechanism and Machine Theory*, 208:105953, 2025.

[2] Farrokh Janabi-Sharifi, Amir Jalali, and Ian D. Walker. Cosserat rod-based dynamic modeling of tendon-driven continuum robots: A tutorial. *IEEE Access*, 9:68703–68719, 2021.

[3] D. Caleb Rucker and Robert J. Webster III. Statics and dynamics of continuum robots with general tendon routing and external loading. *IEEE Transactions on Robotics*, 27(6):1033–1044, 2011.

[4] John Till, Vincent Aloi, and D. Rucker. Real-time dynamics of soft and continuum robots based on cosserat-rod models. *The International Journal of Robotics Research*, 38:723–746, 05 2019.

[2] [3] [4] [1]

7 Appendix - Matlab Code

```
% This program implements the Cosserat Rod Theory for a
   Tendon Driven CR
% with simple Control inputs using Sliding Model Control and
   Lyapunov
% Stability
% Benjamin Chung
% 11/25/2025

function SingleTendonCosseratRod
clear all;
clc;

global i p R j n m v u q w ns vs us vt ut qt wt vst ust vh uh
    vsh ush qh wh nLL mLL x y z X Y Z  %Make vars available
   in whole program
global Tension_Hist X1_Hist Xdot1_Hist e_Hist edot_Hist
   f_Hist norm_e_Hist Control_c
%Hat operator
hat = @(y)[0, -y(3), y(2);
    y(3), 0, -y(1);
    -y(2), y(1), 0];

%Declare variables
L = 0.5;                        %Length in m
N = 20;                         %Spatial resolution
E = 190e9;                      %Young's modulus
r = 1/1000;                      %Cross-section radius
rt = {[0.01;0;0]                %Location of Tendons - We'll
   see if 4 works
    [0;0.01;0]
    [-0.01;0;0]
    [0;-0.01;0]};
rho = 17189;                     %Density
g = [0;0;-9.81];                %Gravity vector
Bse = zeros(3);                 %Shear/Bending Coefficient
Bbt = 0.008*eye(3);               %Bending/Torsion Coefficient
C = 0.01*eye(3);                 %Viscous Damping Coefficient
Tt = {0 0 0 0};                 %Assume 0 Tension during
   initial Static solve
dt = 0.005;                     %Size of Time step
alpha = -0.2;                   %BDF-alpha parameter
STEPS = 100;                     %Number of timesteps to
   completion
STEPCOUNT = 1;
```

```
%Initial Pre-bending variables
vstar = @(s)[0;0;1];
ustar = @(s)[0;0;0];
vsstar = @(s)[0;0;1];
usstar = @(s)[0;0;0];

%Boundary Conditions
%Clamped base to free end
for i = 1 : STEPS
    p{i,1} = [0;0;0];
    R{i,1} = eye(3);
    q{i,1} = [0;0;0];
    w{i,1} = [0;0;0];
end

nL = [0;0;0];                       %Start with no forces
mL = [0;0;0];

%Dependent Parameter Calculations
A = pi*r^2;                                     %Cross-sectional
   area
J = diag([pi*r^4/4  pi*r^4/4  pi*r^4/2]);   %Inertia Matrix
G = E / (2 * (1 + 0.3));                        %Shear modulus
Kse = diag([G*A, G*A, E*A]);                    %SE Stiffness
   Matrix
Kbt = diag([E*J(1,1), E*J(2,2), G*J(3,3)]); %BT Stiffness
   Matrix
ds = L/(N-1);                                   %Finite change in
    S over total steps

%BDF-alpha coefficients
C0 = (1.5 + alpha) / ( dt*(1+alpha));
C1 = -2/dt;
C2 = (0.5 + alpha) / ( dt*(1+alpha));
d1 = alpha / (1+alpha);

% -- START SIMULATION --
% Steps to consider:
% 1) Solve with initial static conditions
% 2) Apply Tension Forces
% 3) Semi-discretize the PDE into an ODE using BDF-Alpha
% 4) Solve PDE with shooting

%Initial Static Solve in order to get IVP problem going
i = 1;
disp("Solving Initial Static Model")
```

```
fsolve(@initialSolve, zeros(6,1)); %Solve static BVP w/
   shooting method
visualize();

%Setup initial history terms
for j = 1 : N-1
    vh{i+1,j} = (C1+C2)*v{i,j};
    uh{i+1,j} = (C1+C2)*u{i,j};
    vsh{i+1,j} = (C1+C2)*vs{i,j};
    ush{i+1,j} = (C1+C2)*us{i,j};
    qh{i+1,j} = [0;0;0];
    wh{i+1,j} = [0;0;0];
    q{i,j} = [0;0;0];
    w{i,j} = [0;0;0];
end

%Set Control Variables
CONTROL = true;                         %SET TRUE OR FALSE IF YOU
    WANT TO USE THE SMC
P_d = [0.027; 0.0000; 0.4727];          %Desired end point
V_d = [0; 0; 0];                        %Desired tip velocity
Acc_d = [0; 0; 0];                      %Desired tip acceleration
Control_c = diag([15000, 1, 500]);      %Control gain 1
k = diag([60, 1, 5]);                   %Control gain 2 (for S
   convergence)
epsilon = 0.005;                        %Reaching epsilon
clamp = 8;

%DEBUG
warning('off', 'all');
options = optimoptions('fsolve', 'Display', 'none');


% -- START SIMULATION LOOP --
for i = 2 : STEPS
    STEPCOUNT = STEPCOUNT + 1;
    %Set the Tendon Forces at each time step
    if i < 5
        Tt{1} = 0; %Apply 0 for initial history terms
        Tt{2} = 0;
        Tt{3} = 0;
        Tt{4} = 0;
    else
        if CONTROL
            Tt{1} = SMCCosserat(P_d, V_d, Control_c, k,
               epsilon, Acc_d);
            Tt{2} = 0;
```

```
            Tt{3} = 0;
            Tt{4} = 0;
        else
            Tt{1} = 0; %MANUAL INPUT TENDON FORCES
            Tt{2} = 0;
            Tt{3} = 0;
            Tt{4} = 0;
        end
    end

    %Now to actually solve, given the tendon forces
    disp(["Solving Dynamic Model, Step = ", i])
    fsolve(@dynamicSolve, [n{i-1,1}; m{i-1,1}], options); %
       Solve semi-discretized PDE w/ shooting
    updateBDFalpha();
    visualize();
end

disp('Final Pos'); p{i,N-1}

% -- FUNCTIONS --
    function E = initialSolve(G)
        n{i,1} = G(1:3);
        m{i,1} = G(4:6);

        %Euler's method for IVP problem
        for j = 1 : N-1
            [ps, Rs, ns, ms, us{i,j}, vs{i,j} ,v{i,j}, u{i,j
               }] = formStaticCosseratODE(p{i,j},R{i,j},n{i,j
               },m{i,j});
            p{i,j+1} = p{i,j} + ds*ps;
            R{i,j+1} = R{i,j} + ds*Rs;
            n{i,j+1} = n{i,j} + ds*ns;
            m{i,j+1} = m{i,j} + ds*ms;
        end

        E = [ n{i,N} - nL;  m{i,N} - mL ];
    end

    function E = dynamicSolve(G)
        n{i,1} = G(1:3);
        m{i,1} = G(4:6);

        %Euler's method
        for j = 1 : N-1
            [ps, Rs, ns, ms, qs, ws, vs{i,j}, us{i,j},...
                v{i,j}, u{i,j}, vt{i,j}, ut{i,j},...
```

```
                qt{i,j}, wt{i,j},vst{i,j}, ust{i,j}] =
                   formCosseratODE(p{i,j},R{i,j},n{i,j},m{i,j
                   },q{i,j},w{i,j});

            p{i,j+1} = p{i,j} + ds*ps;
            R{i,j+1} = R{i,j} + ds*Rs;
            n{i,j+1} = n{i,j} + ds*ns;
            m{i,j+1} = m{i,j} + ds*ms;
            q{i,j+1} = q{i,j} + ds*qs;
            w{i,j+1} = w{i,j} + ds*ws;

        end

        E = [n{i,N} - nLL ;  m{i,N} - mLL];
    end


%This function will essentially build the ODE that will be
   put through
%fsolve dynamically over time
    function [ps,Rs,ns,ms,qs,ws,vs,us,v,u,vt,ut,qt,wt,vst,ust
       ] = formCosseratODE(p, R, n, m, q, w, Tt)
        v = (Kse + C0*Bse)\(R'*n + Kse*vstar(ds*(j-1)) - Bse*
           vh{i,j}); %Use \ for inverse for better stability
        u = (Kbt + C0*Bbt)\(R'*m + Kbt*ustar(ds*(j-1)) - Bbt*
           uh{i,j});

        %Time derivatives - Replaced by BDFa terms
        vt = C0*v + vh{i,j};
        ut = C0*u + uh{i,j};
        qt = C0*q + qh{i,j};
        wt = C0*w + wh{i,j};

        %Calculate Forces
        %Using _ for subscript t, for tension forces
        [a_t, b_t, A_t, B_t, G_t, H_t] = getTendonForces(v, u
           );

        phi = [Kse + C0*Bse + A_t, G_t;
            G_t', Kbt + C0*Bbt + H_t];

        LamdaN = -a_t + rho*A*(hat(w)*q + qt)+ C*q.*abs(q)- R
           '*rho*A*g;
        LamdaM = -b_t + rho*(hat(w)*J*w + J*wt) -hat (v)*(Kse
           *(v-vstar(ds*(j-1)))+Bse*vt);
        GammaV = hat(u)*(Kse*(v-vstar(ds*(j-1)))+Bse*vt)-Kse*
           vsstar(ds*(j-1))+Bse*vsh{i,j};
```

```
        GammaU = hat(u)*(Kbt*(u-ustar(ds*(j-1)))+Bbt*ut)-Kbt*
           usstar(ds*(j-1))+Bbt*ush{i,j};

        %Matrix inverse formula as all matrices are
           orthogonal
        us = 1/det(phi)*(-G_t'*(-GammaV+LamdaN)+(Kse+C0*Bse+
           A_t)*(-GammaU+LamdaM));
        vs = 1/det(phi)*((Kbt+C0*Bbt+H_t)*(-GammaV+LamdaN)-
           G_t*(-GammaU+LamdaM));

        vst = C0*vs + vsh{i,j};
        ust = C0*us + ush{i,j};

        f = rho*A*g + R*(a_t + A_t*vs + G_t*us) - (R*C*q.*abs
           (q));
        f_Hist{i,j} = f;
        l = R*(b_t + B_t*vt + H_t*ut);

        %Find boundary forces and moments from tendons for
           residual calculation
        [nLL, mLL] = getTendonForcesAtL(R, u, v);

        %Spatial derivatives
        ps = R * v;
        Rs = R * hat(u);
        qs = vt - hat(u)*q + hat(w)*v;
        ws = ut - hat(u)*w;
        ns = rho*A*R*(hat(w)*q + qt) - f;
        ms = R*rho*(hat(w)*J*w + J*wt) - hat(ps)*n - l;
    end

%STATIC
%This function is very explicit as I simly set the time terms
    to 0
%Could be optimized to OMIT time terms later on
    function [ps, Rs, ns, ms, us, vs, v, u] =
       formStaticCosseratODE(p, R, n, m)
        v = (Kse)\(R'*n + Kse*vstar(ds*(j-1))); %Use \ for
           inverse for better stability
        u = (Kbt)\(R'*m + Kbt*ustar(ds*(j-1)));

        %Time derivatives - all 0 at static

        %Calculate Forces
        %Using _ for subscript t, for tension forces
        [a_t, b_t, A_t, B_t, G_t, H_t] = getTendonForces(v, u
           );
```

```
        %Following Sharifi et al for vs and us
        %Their code seems to assume that all block matrices
           are diagonal
        %The following works in all cases, but is a bit less
           efficient
        %I didn't time to verify the matrix sizes to know if
           Gt is diagonal
        %WARNING: If singular, the whole solution explodes
        phi = [Kse + A_t, G_t;
               G_t', Kbt + H_t];

        LamdaN = hat(u) * (Kse * (v - vstar(ds*(j-1)))) - Kse
           *vsstar(ds*(j-1));
        LamdaM = hat(u) * (Kbt * (u - ustar(ds*(j-1)))) - Kbt
           *usstar(ds*(j-1));
        GammaN = R'*rho*A_t*g - a_t; %**R'R = 1, so can be
           omitted
        GammaM = hat(v) * Kse * (v - vstar(ds*(j-1))) - b_t;
           %**hat(ps) = Rv, also le = 0

        rhs = [-GammaN + LamdaN;
            -GammaM + LamdaM];
        temp = phi \ rhs;

        %Extract the result into vs and us
        vs = temp(1:3);
        us = temp(4:6);

        f = rho*A*g;
        f_Hist{i,j} = f;
        l = 0;

        %Spatial derivatives
        ps = R * v;
        Rs = R * hat(u);
        ns = -f;
        ms = -hat(ps)*n - l;
    end

%This function calculates the variables required to find the
   tendon forces
%Honestly, I could make this modular to have n tendons
%Probably a future thing
    function [a, b, At, Bt, Gt, Ht] = getTendonForces(v, u)

        %Tendon 1
```

```
ptsb1 = hat(u)*rt{1}+v;
At1 = -Tt{1} * (hat(ptsb1) * hat(ptsb1)) / (norm(
   ptsb1)^3);
Gt1 = At1*hat(rt{1});
Bt1 = hat(rt{1}) * At1;
H1 = Bt1 * hat(rt{1});
a1 = At1 * (hat(u) * ptsb1); %Assume the derivatives
   of r to be 0
b1 = hat(rt{1}) * a1;

%Tendon 2
ptsb2 = hat(u)*rt{2}+v;
At2 = -Tt{2} * (hat(ptsb2) * hat(ptsb2)) / (norm(
   ptsb2)^3);
Gt2 = At2*hat(rt{2});
Bt2 = hat(rt{2}) * At2;
H2 = Bt2 * hat(rt{2});
a2 = At2 * (hat(u) * ptsb2);
b2 = hat(rt{2}) * a2;

%Tendon 3
ptsb3 = hat(u)*rt{3}+v;
At3 = -Tt{3} * (hat(ptsb3) * hat(ptsb3)) / (norm(
   ptsb3)^3);
Gt3 = At3*hat(rt{3});
Bt3 = hat(rt{3}) * At3;
H3 = Bt3 * hat(rt{3});
a3 = At3 * (hat(u) * ptsb3);
b3 = hat(rt{3}) * a3;

%Tendon 4
ptsb4 = hat(u)*rt{4}+v;
At4 = -Tt{4} * (hat(ptsb4) * hat(ptsb4)) / (norm(
   ptsb4)^3);
Gt4 = At4*hat(rt{4});
Bt4 = hat(rt{4}) * At4;
H4 = Bt4 * hat(rt{4});
a4 = At4 * (hat(u) * ptsb4);
b4 = hat(rt{4}) * a4;

%Sum the values
a = a1 + a2 + a3 +a4;
b = b1 + b2 + b3 +b4;
At = At1 + At2 + At3 + At4;
Bt = Bt1 + Bt2 + Bt3 + Bt4;
Gt = Gt1 + Gt2 + Gt3 + Gt4;
Ht = H1 + H2 + H3 + H4;
```

```
    end

%This function find the tendon forces at the end point L
%Later then used in fsolve for residual calculation to solve
   the IVP
    function [nLL, mLL] = getTendonForcesAtL(R, u, v)
        pts1 = R*hat(u)*rt{1}+R*v;
        pts2 = R*hat(u)*rt{2}+R*v;
        pts3 = R*hat(u)*rt{3}+R*v;
        pts4 = R*hat(u)*rt{4}+R*v;

        nLL = -Tt{1}*pts1/norm(pts1) - Tt{2}*pts2/norm(pts2)
           - Tt{3}*pts3/norm(pts3); - Tt{4}*pts4/norm(pts4);
        mLL = -Tt{1}*hat(R*rt{1})*pts1/norm(pts1) - Tt{2}*hat
           (R*rt{2})*pts2/norm(pts2) - Tt{3}*hat(R*rt{3})*
           pts3/norm(pts3) - Tt{4}*hat(R*rt{4})*pts4/norm(
           pts4);
    end

    function updateBDFalpha()
        %h for history, these are history term updates
        for j = 1 : N-1
            vh{i+1,j} = C1*v{i,j} + C2*v{i-1,j} + d1*vt{i,j};
            uh{i+1,j} = C1*u{i,j} + C2*u{i-1,j} + d1*ut{i,j};
            vsh{i+1,j} = C1*vs{i,j} + C2*vs{i-1,j} + d1*vst{i
               ,j};
            ush{i+1,j} = C1*us{i,j} + C2*us{i-1,j} + d1*ust{i
               ,j};
            qh{i+1,j} = C1*q{i,j} + C2*q{i-1,j} + d1*qt{i,j};
            wh{i+1,j} = C1*w{i,j} + C2*w{i-1,j} + d1*wt{i,j};
        end
    end

    function Tt = SMCCosserat(X1d, Xdot1d, Control_c, k,
       epsilon, Xddotd)

        X1 = p{i-1,N-1};
        Xdot1 = R{i-1,N-1}*q{i-1,N-1};
        e = (X1d - X1);                 %Error between
           positions
        edot = Xdot1d - Xdot1;          %Derivative of error,
            which is velocity
        S = edot + Control_c*e;         %Sliding surface

        %Calculating the Alpha
        pts1 = R{i-1, N-1}*hat(u{i-1,N-1})*rt{1} + v{i-1,N
           -1};
```

```
        ptss1 = R{i-1, N-1}*hat(u{i-1,N-1})*hat(u{i-1,N-1})*
           rt{1} + hat(u{i-1,N-1})*v{i-1,N-1} + hat(us{i-1, N
           -1})*rt{1} + vs{i-1,N-1};

        a1 = ( (hat(pts1)*hat(pts1)*ptss1) / (norm(pts1)^3) )
            + (pts1 / norm(pts1));
        alpha = a1;

        ac = (ns + rho*A*g - (R{i-1, N-1}*C*q{i-1, N-1}.*abs(
           q{i-1, N-1}))) / (rho*A);
        bc = (alpha) / (-rho*A);

        Tt = bc\(Control_c*edot + Xddotd - ac + epsilon*sign(
           S) + k*(edot + Control_c*e));

        %Apply clamping saturation
        if(Tt > clamp)
            Tt = clamp;
        end

        norm_e_Hist(i) = norm(e);
        Tension_Hist(i) = Tt;
        X1_Hist(i) = X1(1);
        Xdot1_Hist(i) = Xdot1(1);
        e_Hist(i) = e(1);
        edot_Hist(i) = edot(1);
    end


    function visualize()
        % p: cell array p{i,j} = 3x1 positions
        % N: number of points along rod
        % L: rod length for axis limits
        % i: current time step

        % Extract x,y,z positions
        x = zeros(1,N);
        y = zeros(1,N);
        z = zeros(1,N);
        for j = 1:N
            x(j) = p{i,j}(1);
            y(j) = p{i,j}(2);
            z(j) = p{i,j}(3);
        end

        % Persistent line handle to update instead of
           replotting
```

```
        persistent hLine
        if isempty(hLine) || ~isvalid(hLine)
            figure(1); clf;
            hLine = plot3(x, y, z, '-o', 'LineWidth', 2, '
               MarkerSize', 4);
            axis([-0.1*L, 0.1*L, -0.1*L, 0.1*L, -0.05*L, 1.1*
               L]);
            xlabel('x (m)'); ylabel('y (m)'); zlabel('z (m)')
               ;
            grid on; view(3);
        else
            % Update existing line
            set(hLine, 'XData', x, 'YData', y, 'ZData', z);
        end
        drawnow;
        pause(0.05);
    end

for time = 1:STEPCOUNT
    x(time) = p{time,N}(1);
    z(time) = p{time,N}(3);
    velocity{time} = R{time, N-1}*q{time,N-1};
end

velocity_x = cellfun(@(v) v(1), velocity);
velocity_y = cellfun(@(v) v(2), velocity);
velocity_z = cellfun(@(v) v(3), velocity);

accel_x = gradient(velocity_x, dt);
accel_y = gradient(velocity_y, dt);
accel_z = gradient(velocity_z, dt);



%All of the plotting fun!
if(CONTROL)
    x_axis = 1:size(x,2);
    figure (2)
    subplot(2,1,1)
    plot(x_axis,x, '-o');
    xlabel('Step');  ylabel('x (m)'); title('Tip Displacement
        - X Component');

    subplot(2,1,2)
    t_mean = mean(Tension_Hist);
    hold on
    plot_t = plot(x_axis, Tension_Hist, '-o');
```

```
    mean_t = yline(t_mean, 'r--', 'LineWidth', 2);
    xlabel('t (s)');  ylabel('F (N)'); title('Tendon Tension'
       );
    legend([plot_t, mean_t], {'Tension (N)', ['Mean = ',
       num2str(t_mean,'%.2f')]}, 'Location','best')
    hold off

    figure(3)
    plot(X1_Hist, Xdot1_Hist);
    xlabel('X1'); ylabel('Xdot1');

    figure(4)
    plot(e_Hist, edot_Hist);
    xlabel('e'); ylabel('e_dot');

    figure(5)
    plot(x_axis, norm_e_Hist);
    xlabel('Step'); ylabel('e');

else
    x_axis = 1:size(x,2);
    figure (2)
    plot(x_axis, x, '-o');
    xlabel('Step');  ylabel('x (m)'); title('Tip Displacement
        - X Component');

    figure (3)
    plot(x_axis, z, '-o');
    xlabel('Step');  ylabel('z (m)'); title('Tip Displacement
        - Z Component');

    figure (4)
    hold on
    vx = plot(x_axis, velocity_x, 'r');
    vy = plot(x_axis, velocity_y, 'g');
    vz = plot(x_axis, velocity_z, 'b');
    xlabel('Step');  ylabel('Velocity (m/s)'); title('Tip
       Velocity');
    legend([vx vy vz], {'X Velocity','Y Velocity','Z Velocity
       '}, 'Location','best')
    hold off

    figure (5)
    hold on
    ax = plot(x_axis, accel_x, 'r');
    ay = plot(x_axis, accel_y, 'g');
    az = plot(x_axis, accel_z, 'b');
```

```
    xlabel('Step');  ylabel('Acceleration (m/s^2)'); title('
       Tip Acceleration');
    legend([ax ay az], {'X Acceleration','Y Acceleration','Z
       Acceleration'}, 'Location','best')
    hold off
end

end
```