Implementation of Nonlinear Model Predictive Growth Control for Soft Growth Robots

Lowell Lobo
School of Engineering
University of Maryland
College Park
Email: lorocks@umd.edu

***Abstract*—The method for controlling the growth of a soft-growth robot using a Nonlinear Model Predictive Control method in [1] is analyzed and results are compared. All results in [1] are derived, and using these findings a controller algorithm is designed in CasAdi to emulate a Nonlinear Model Predictive Control. The results obtained vary a lot compared to the simulations in [1] which can be explained due to the lack of information in model design and algorithm implemented for the Nonlinear Model Predictive Control.**

1. Introduction

The following article is the implementation and simulation of the work proposed in [1] for Growth Control of a Class of Plant-Inspired Soft Growing Robots. [1] proposes the use of a Nonlinear Model Predictive Control method to automate growth characteristics and apply feasible constraints to the robot. Constraints include irreversible growth and slow movement which will be discussed later.

1.1. Soft Growing Robots

The morphological adaptation capabilities shown by elephant trunks, octopus arms, squid tentacles, and snakes have heavily inspired the development of soft-growing robots or soft continuum robots [1][2]. Continuum robots can only influence a small workspace beyond which there can be a loss of control and immense actuator strain. As such, continuum robots are used in scenarios that require efficient maneuvering in tight or obstacle-ridden terrain. Compared to hyper-redundant robots, which have more than 6 kinematically actuated joints, continuum robots can influence the robot workspace with less directly actuated DOF and have far greater maneuverability [3]. But as compared to normal robots, continuum robots require stronger hardware material or tricky mechanical design to perform operations at the edges of the workspace, withstand weight-intensive applications, and execute complex path navigation. Continuum robots find applications in disaster situations like search and rescue and can be applied for minimally invasive surgery as well.

1.2. Model Predictive Control

Model Predictive Control (MPC) falls under the class of predictive controllers that provide control actions based on forward regression of system states from an initial condition, found through optimization of control input on the system plant model over a specified period. MPC can handle multi-input multi-output (MIMO) systems and takes into account all system variables and input interactions. MPC is also able to handle environmental constraints like joint activation limits and actuator efforts by incorporating them into the optimization problem. The goal of any controller is to provide control inputs such that the system follows a certain reference and MPC does this by using an optimizer to predict the future states of the provided system plant.

MPC parameters are listed below:-

- Plant - Kinematic or Dynamic model of the system that can be used to drive the system
- Sample Time - Rate at which the controller finds optimal control actions and applies them to the plant
- Prediction Horizon - Time period over which the optimal controls are to be found
- Control Horizon - Time period over which control actions will be varied
- Cost Function - optimization problem to be minimized used in finding control actions to drive the system to the reference
- Weights - Ratio of importance given to parameters in the cost function
- Constraints - Environmental constraints that will be included in the optimization algorithm

An MPC algorithm finds optimal control inputs for steps in the prediction horizon based on the control horizon. If the control horizon is equal to the prediction horizon, the MPC algorithm will compute the optimal control input for every step in the prediction horizon. If the control horizon is lesser, the MPC algorithm will compute optimal control inputs till

the control horizon and keep the remaining control inputs the same as the one previously computed. For example, in an algorithm with a prediction horizon of 10 and a control horizon of 2. The algorithm will run the iteration 10 times. The first optimal control will be calculated and applied to the system. At the second step a new optimal control is found and applied to the new system state found in the first step. From the third step onward, the optimal control from the second step is used, since the control horizon is 2, but the control is applied to updated system states. Finally, the MPC will find where the system will be in the future after optimization of control inputs. Even though computation is done over a range of steps, only the first optimal control action is applied to the system. This helps account for any external disturbances affecting the system. For the optimization problem, a cost function is formulated normally in the form,

$$J = \int X(t)^T Q X(t) dt + \int U(t)^T R U(t) dt$$

or

$$J = \sum_{i=1}^{k} X_i^T Q X_i + \sum_{i=1}^{k} U_i^T R U_i$$

where Q and R are diagonal weight matrices.

The objective of optimization is to minimize the cost function taking into account any constraints. The weights help prioritize the parameters that need to be minimized. If the weight of a specific parameter is higher than another, the one with the higher weight will affect the cost value more and therefore is minimized more.

MPC can be computationally heavy depending on prediction horizon, control horizon, and convergence. Finally, there are many types of MPC algorithms which include, Linear, Adaptive, Gain-Scheduled and Nonlinear MPCs.

In this article, the methodology and implementation described in [1] will be simulated, and results compared. The kinematic modeling, state space model description, and Nonlinear Model Predictive Control (NMPC) methodology for growth control in soft robotics will be analyzed in Section 2. The derived methodology will be simulated and results compared in Section 3. Limitations and constraints will be discussed in Section 4, followed by a final conclusion in Section 5.

2. Discussion

2.1. Plant Model Parameters

The objective is to build a NMPC method to actuate the continuum robot. A continuum robot can be defined in state space using six parameters. The position of the tip of the end effector can be defined using Cartesian coordinates (x, y, z). Normally most robots use roll (ψ), pitch (θ), and yaw (ϕ) to define the position in space and then correlate the

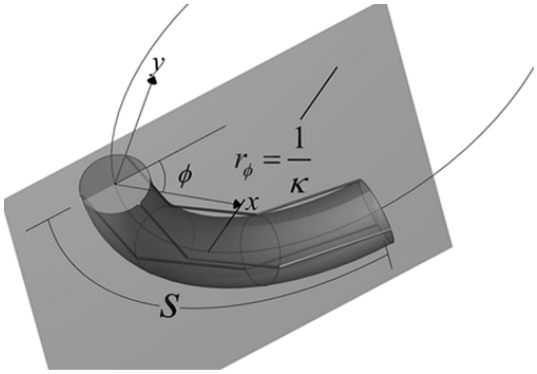

Figure 1. Robot parameters, s, k and ϕ [2]

[x, y, z, ψ, θ, ϕ] to the joint angles. But in the case of continuum robots, the length of the trunk (s), the curvature of the trunk (k) and angle of curvature (ϕ) can be used to fully define the robot. The ϕ value shows the angle of rotation of the tip in the XY plane [2].

The following parameters can be adapted and instead of k, θ can be used where θ is found using the formula,

$$s = r\theta$$

Since, $r = \frac{1}{k}$, we get

$$s = \frac{\theta}{k} \tag{1}$$

$$\theta = sk \tag{2}$$

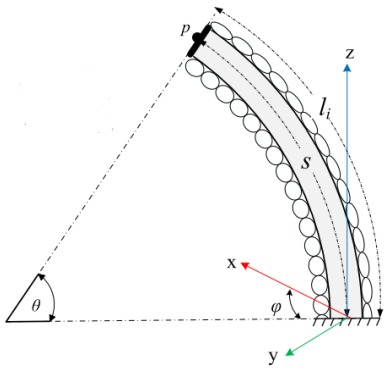

Figure 2. Edited parameters, s, θ, ϕ [1]

2.2. Modeling

There are two methods by which the model of the robot can be found. The first is by computing a transformation matrix using DH parameters and the other is through geometrical analysis. In this article, the geometrical analysis combined with the homogeneous transformations is used to

derive the kinematic model for continuum robots. The pure DH parameter modeling method will be mentioned later for cross-referencing.

2.2.1. Kinematic Model. First, we find the distance between the origin of the robot and its end effector. In Figure 3, s is the arc length of the robot, O is its origin and E is the end effector location. The value θ is placed based on Figure 2.

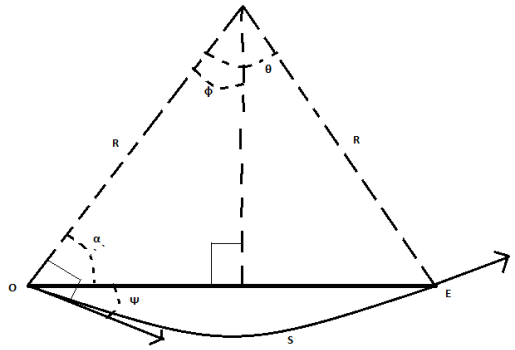

Figure 3. Planar Geometry of Continuum Robot [3]

From (2), we have,

$$\theta = sk$$

$$\phi = \frac{\theta}{2} = \frac{sk}{2} \tag{3}$$

From the triangle we get,

$$\alpha + \phi + 90 = 180$$

$$\psi = 90 - \alpha$$

Substituting the above equation gives,

$$90 - \psi + \phi + 90 = 180$$

$$\phi - \psi = 0$$

From (3), we get ψ to be,

$$\psi = \frac{ks}{2} \tag{4}$$

Also, from Figure 3, the angle ϕ gives,

$$sin\phi = \frac{d}{2R}$$

where d is the distance between the origin and end effector and $r = \frac{1}{k}$. Thus we have,

$$d = \frac{2sin\phi}{k}$$

Substituting the value of ϕ from equation (3) gives us,

$$d = \frac{2sin(\frac{ks}{2})}{k} \tag{5}$$

Proof: [3]

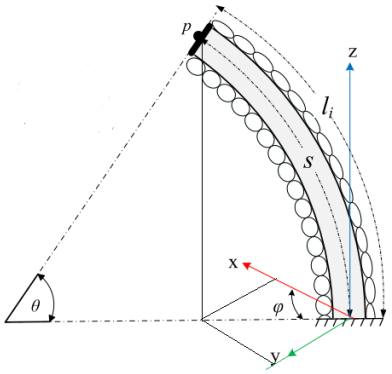

Figure 4. Radius of rotation used to find the values of x and y [1]

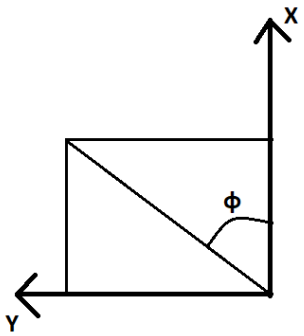

Figure 5. Vector components of radius with angle ϕ

In Figure 2. the robot is arced away from the z-axis with a rotation of ϕ. Using geometry as shown in Figure 4 and Figure 5, we can get the relation between the x and y distance as,

$$x = rcos\phi$$

$$y = rsin\phi \tag{6}$$

Where r is the radius of rotation on the XY plane.

To find the coordinates of the end-effector, the homogeneous transformation matrix needs to be computed. The steps in the transformation consist of rotation to align the z-axis along the vector $\vec{d}$ and translation of value d along the z-axis.

To simplify the calculation, the robot is first rotated by an angle $90 - \phi$ on the XY plane.
This results in the $\vec{d}$ lying on the YZ plane. Also, the y value of the robot is now equal to r, the radius of rotation on the XY plane, since rotation took place.

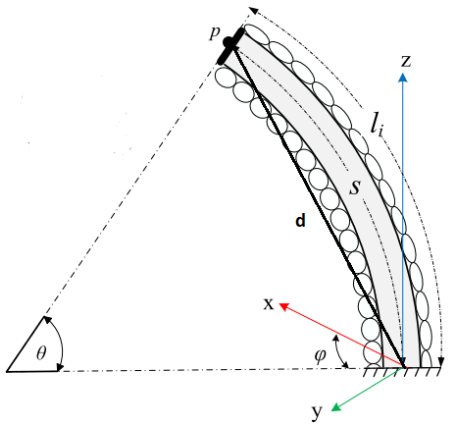

Figure 6. d, distance vector between origin and end effector location [1]

Total transformation is:-

1) Rotation by $90 - \phi$ along the z-axis
2) Rotation along the x-axis by ψ as in Figure 3
3) Translation by d along z-axis.

Since the first rotation can be logically explained, we will focus only on transformations 2 and 3. The expected outcome should be a y value and z value, and $x = 0$ since after transformation 1, x will be zero.

Any homogeneous transformation matrix is made of up of a rotational and translational component such that,

$$H = \begin{bmatrix} R_{xyz} & T_{xyz} \\ 0 & 1 \end{bmatrix}$$

Where, R_{xyz} is the rotation matrix to achieve the rotational configuration, and T_{xyz} is the translation matrix for the same.

The translation matrix also gives the position of the end effector defined on the Cartesian 3D plane.

$$T_{xyz} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$H = R_{Z_\psi} T_{Z_d}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\psi & -sin\psi & 0 \\ 0 & cos\psi & -sin\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\psi & -sin\psi & -dsin\psi \\ 0 & cos\psi & -sin\psi & dcos\psi \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From H, we see that, $x = 0$ as expected and,

$$y = -dsin\psi$$

$$z = dcos\psi$$

Substituting values of d and ψ from (4) and (5) gives,

$$y = -\frac{2sin(\frac{ks}{2})sin(\frac{ks}{2})}{k}$$

$$z = \frac{2sin(\frac{ks}{2})cos(\frac{ks}{2})}{k}$$

We know that,

$$sin(\frac{A}{2}) = \sqrt{\frac{1 - cosA}{2}}$$

$$cos(\frac{A}{2}) = \sqrt{\frac{1 + cosA}{2}}$$

Using the above equations give,

$$y = -\frac{2(1 - cos(ks))}{2k} \tag{7}$$

Thus,

$$y = \frac{(cos(ks) - 1)}{k}$$

and,

$$z = \frac{2\sqrt{\frac{1-cos(ks)}{2}}\sqrt{\frac{1+cos(ks)}{2}}}{k}$$

$$z = \frac{\sqrt{1 - cos^2(ks)}}{k}$$

$$z = \frac{sin(ks)}{k} \tag{8}$$

We know that the robot is on the YZ plane and thus, $\phi = 90$, from (6), we can infer that since,

$$y = rsin\phi$$

$$y = rsin(90)$$

$$y = r$$

and from (7), we get

$$r = \frac{(cos(ks) - 1)}{k}$$

From this, we get,

$$x = rcos\phi$$

$$x = \frac{cos(\phi)(cos(ks) - 1)}{k} \tag{9}$$

and,

$$y = rsin\phi$$

$$y = \frac{sin(\phi)(cos(ks) - 1)}{k} \tag{10}$$

Thus from (1), (2), (8), (9) and (10), we get,

$$x = \frac{s\ cos\phi(cos\theta - 1)}{\theta}$$
$$y = \frac{s\ sin\phi(cos\theta - 1)}{\theta}$$
$$z = \frac{s\ sin\theta}{\theta} \tag{11}$$

The above results are similar to the ones in [1], [2] and [3], where DH parameters were used in the derivation.
Thus, the direct kinematics will be,

$$p = Jq$$

Where, $p = \begin{bmatrix} x & y & z \end{bmatrix}^T$, $q = \begin{bmatrix} s & \theta & \phi \end{bmatrix}^T$ and J is the Jacobian matrix.

Thus as part of the modeling only the translation part of the homogeneous transform is required.

2.2.2. Velocity Kinematics. Velocity Kinematics can be calculated analytically using the position kinematics, such that,

$$\dot{p} = J\dot{q}$$

and,

$$J = \frac{\partial p}{\partial q}$$
$$J = \frac{\partial p}{\partial (s, \theta, \phi)}$$

Note: There is a slight mistake in [1] where the partial is written with respect to (s, k, ϕ) when implementation uses (s, θ, ϕ).

Thus with partial differentiation we get,

$J =$

$$\begin{bmatrix} \frac{cos\phi(cos\theta-1)}{\theta} & \frac{s\ cos\phi(1-cos\theta-\theta sin\theta)}{\theta^2} & \frac{-s\ sin\phi(cos\theta-1)}{\theta} \\ \frac{sin\phi(cos\theta-1)}{\theta} & \frac{s\ sin\phi(1-cos\theta-\theta sin\theta)}{\theta^2} & \frac{s\ cos\phi(cos\theta-1)}{\theta} \\ \frac{sin\theta}{\theta} & \frac{s\theta cos\theta - sin\theta}{\theta^2} & 0 \end{bmatrix}$$

Therefore,

$\dot{p} =$

$$\begin{bmatrix} \frac{cos\phi(cos\theta-1)}{\theta} & \frac{s\ cos\phi(1-cos\theta-\theta sin\theta)}{\theta^2} & \frac{-s\ sin\phi(cos\theta-1)}{\theta} \\ \frac{sin\phi(cos\theta-1)}{\theta} & \frac{s\ sin\phi(1-cos\theta-\theta sin\theta)}{\theta^2} & \frac{s\ cos\phi(cos\theta-1)}{\theta} \\ \frac{sin\theta}{\theta} & \frac{s\theta cos\theta - sin\theta}{\theta^2} & 0 \end{bmatrix} \dot{q}$$

Where, $\dot{p} = [\dot{x}, \dot{y}, \dot{z}]$ and $\dot{q} = [\dot{s}, \dot{\theta}, \dot{\phi}]$

2.3. NMPC Model Design

2.3.1. State Space Representation. The states are chosen based on two criteria in this implementation. One is to fully define the robot in the Cartesian coordinate system and the second is to account for irreversible growth. The following robot is to be designed such that the robot only grows but never shrinks. This can be done when the arc length of the robot or trunk curvature, s, will not decrease. The constraints applied for such conditions will be discussed later.

The states chosen are, $\vec{X} = [\ p\ \ q\ ]^T$ or,

$$\vec{X} = \begin{bmatrix} x \\ y \\ z \\ s \\ \theta \\ \phi \end{bmatrix} \tag{12}$$

We also have, $p = Jq$

Thus for computing $\dot{\vec{X}}$ the first three states are related to the velocity of s, θ and ϕ. Also, the last three states are directly related to q.

Therefore the state space equation can be written as,

$$\dot{\vec{X}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{s} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} J \\ I_3 \end{bmatrix} \dot{q} \tag{13}$$

Here $\dot{q}$ can be the control input to the system therefore,

$$\vec{U} = \begin{bmatrix} \dot{s} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} \tag{14}$$

and,

$$B = \begin{bmatrix} J \\ I_3 \end{bmatrix} \tag{15}$$

where B is a 6x3 matrix as in [1]

2.3.2. Cost Function Definition. The main aim of the NMPC is to have the robot follow a predefined trajectory, which is, $\vec{X_{ref}} = [x_{ref}, y_{ref}, z_{ref}, s_{ref}, \theta_{ref}, \phi_{ref}]^T$

The cost function chosen is as below,

$$J(k) = \sum_{j=1}^{N} e_{k+j}^T Q e_{k+j} + \sum_{i=1}^{k} \Delta U_{k+j-1}^T R \Delta U_{k+j-1}$$

Here, e is the error or difference between the reference and current position, calculated as, $e = \vec{X} - \vec{X_{ref}}$, and ΔU is the increment between control action. Q and R are weight matrices kept constant over the prediction horizon N [1].

2.3.3. Constraints. Constraints have been applied to all state variables and control actions to limit the robot workspace and limit growth. This is the main reason for considering states s, θ and ϕ as part of the state space design. Although redundant, this enables the addition of constraints to the robot parameters. If s, θ and ϕ were not included as states, applying constraints would be impossible and impractical simulation results would be displayed.

$$-4 \leq \begin{bmatrix} x\ (m) \\ y\ (m) \\ z\ (m) \end{bmatrix} \leq 4$$

$$\begin{bmatrix} 0 \\ -\pi \\ -\pi \end{bmatrix} \leq \begin{bmatrix} s\ (m) \\ \theta\ (rad) \\ \phi\ (rad) \end{bmatrix} \leq \begin{bmatrix} 10 \\ \pi \\ \pi \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ \\ -\frac{\pi}{10} \\ \\ -\frac{\pi}{10} \end{bmatrix} \leq \begin{bmatrix} \dot{s}\ (m/s) \\ \\ \dot{\theta}\ (rad/s) \\ \\ \dot{\phi}\ (rad/s) \end{bmatrix} \leq \begin{bmatrix} 0.1 \\ \\ \frac{\pi}{10} \\ \\ \frac{\pi}{10} \end{bmatrix}$$

The below constraint combination,

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \leq \begin{bmatrix} \dot{s}\ (m/s) \\ s\ (m) \end{bmatrix} \leq \begin{bmatrix} 0.1 \\ 10 \end{bmatrix}$$

ensures that the robot is unable to shrink, especially since the velocity of s never becomes negative [1].

2.3.4. Internal Controller Design. For an NMPC, the internal plant model of the system needs to be a discrete-time linear model. But the plant model in equation (13) is both continuous-time and nonlinear since the B matrix changes at every time step based on the robot parameters q. Therefore to combat this some measures are introduced. First, the continuous-time model is discretized using Euler discretization thus we get,

$$\vec{X}(k+1) = \vec{X}(k) + \Delta T \dot{\vec{X}}(k)$$

$$\vec{X}(k+1) = \vec{X}(k) + \Delta T \begin{bmatrix} J \\ I_3 \end{bmatrix} \dot{\vec{U}}(k) \tag{16}$$

Here ΔT is the sampling time.

To combat the changing dynamics of the plant model, an Adaptive MPC methodology is used where the B matrix is computed at every time step, thus equation (16) becomes,

$$\vec{X}(k+1) = \vec{X}(k) + \Delta T \begin{bmatrix} J(k) \\ I_3 \end{bmatrix} \dot{\vec{U}}(k) \tag{17}$$

2.3.5. Algorithm. The implemented algorithm uses a multi-shooting method where both the current state and the next control action are considered as parameters in the optimal control problem. The other way of designing a CasAdi MPC algorithm is by using a single-shooting method that only considers the next control action as a parameter for the optimal control problem. Both implementations can give varying results, but it is most often noticed that multi-shooting provides better results.

First, all variables are initialized and reference positions are created using them. Next, using CasAdi symbols, all the functions and equations that are required for the NMPC are derived and written in the symbolic form. Later these functions will be called by replacing the symbols with actual parameters during optimization. After the creation of symbolic functions, the constraints are input. There are two types of constraints, optimization constraints, and value constraints. Value constraints are simple inequality constraints for each and every system state and control input. The optimization constraints, on the other hand, are constraints given to any functions of states and optimization parameters. The optimization constraints are heavily relied on for accuracy in the multi-shooting method. For a single-shooting method, optimization constraints will be provided for control actions, while for multi-shooting constraints are provided for the function below,

$$\vec{X}_{(k+1)} - X_{P_{k+1}} = 0$$

This function gives a condition to the optimizer that, the next predicted state should be equal to the value of state when substituted in the plant model. Although the condition stated seems like it should be obvious, this condition brings the states and control actions as parameters into the optimal control loop. This works especially well with nonlinear systems.

3. Results

This section will show the comparison between the output graphs of the simulations. This is followed by a proposed explanation of the discrepancies and deviations from the design. In [1] CasAdi was used to implement the NMPC algorithm. The personal simulation implementation is also built using CasAdi. CasAdi is an open-source tool for optimization and optimal control which is available for C++, Python, and MATLAB.

The results will be shown in the format where, first the expected simulation result as in [1] is displayed, followed by the actual simulation results. Next, the design parameters is mentioned, followed by the discrepancies and the reason for their existence.

3.1. Point Stabilization

3.1.1. Results of Paper.

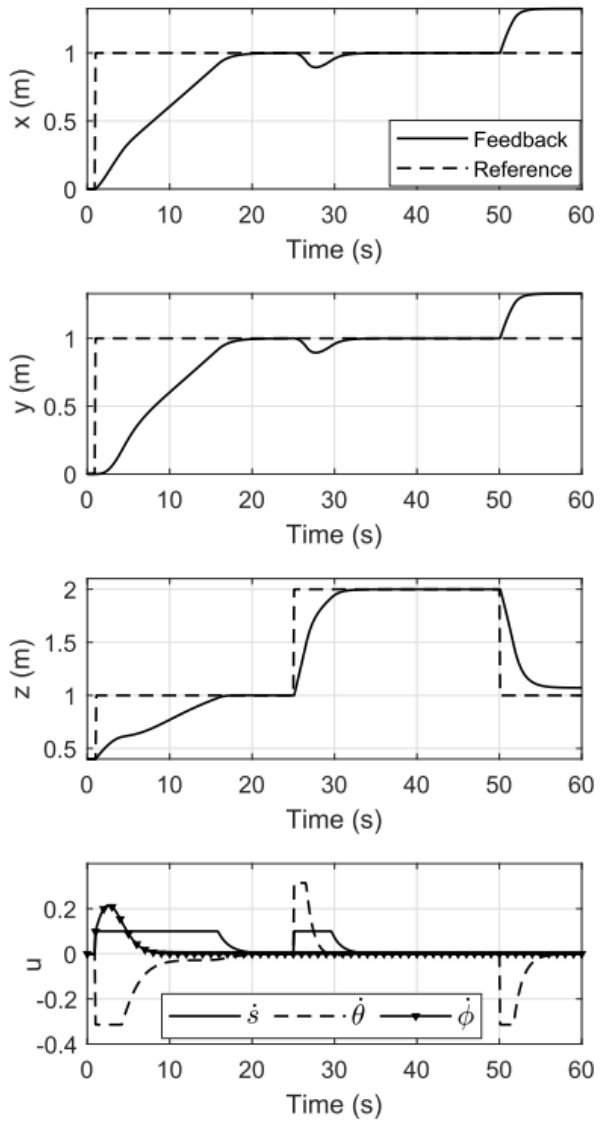

Figure 7. Results of point stabilization from [1]

3.1.2. Results of Personal Implementation.

As shown in Figure 8

3.1.3. Parameter Selection. The initial state, x0, is chosen as [0, 0, 0.4, 0.4, 0, 0]. Comparing the initial state to the direct kinematic equations (11), taking s, θ and ϕ values as 0.4, 0 and 0 gives undefined x, y and z values. This is especially because θ is in the denominator. To derive the values of x, y and z, we take limit θ tending to 0.

$$\lim_{\theta \longrightarrow 0} \frac{cos\theta - 1}{\theta} = 0$$

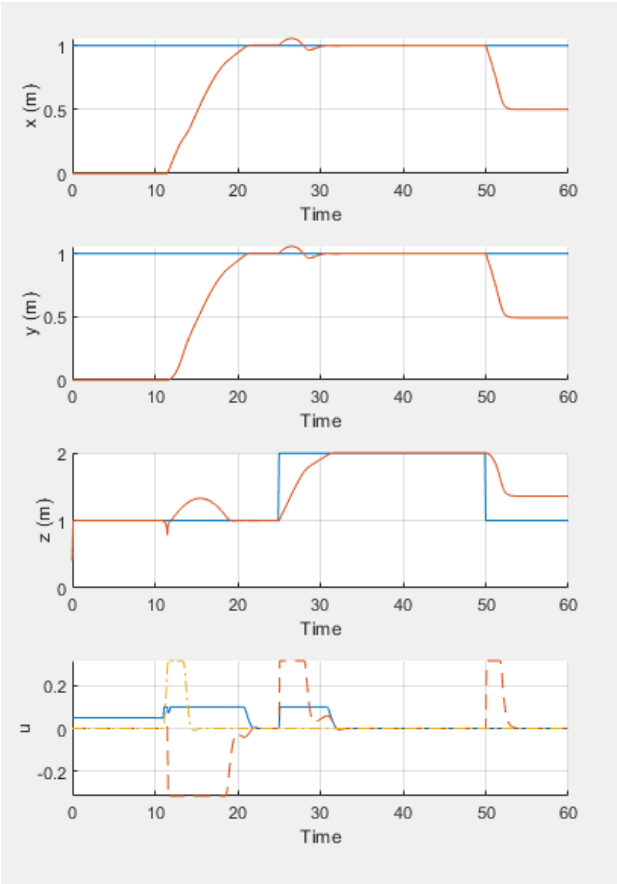

Figure 8. MATLAB simulation results using similar labels as [1]

$$\lim_{\theta \longrightarrow 0} \frac{sin\theta}{\theta} = 1$$

We get,

$$x = 0$$

$$y = 0$$

$$z = s$$

The sampling time is taken as, T_s = 0.1s

The simulation time is taken as, T = 60s

Prediction horizon is taken as, N = 10

The state weight matrix, Q, is taken as a 6x6 matrix with diagonal values, Q = (1, 1, 1, 0, 0, 0)

The state weight matrix is taken as a 3x3 matrix with diagonal values, R = (0.5, 0.5, 0.5)

3.1.4. Discrepancies. Comparing Figure 7 and Figure 8, there are major similarities between the images, yet the simulations do not match. The findings are listed below,

1) z value shoots up to 1 immediately, while the z value in [1] increments slowly
2) If the time frame between 0s - 10s and 10s - 20s is swapped the results from 0s - 50s will look similar
3) The x, y and θ values are opposite increments compared to [1], while the z value is similar
4) In between 10s and 20s, z value increments in a odd manner

Most of the discrepancies are attributed to the model design. The z value shoots up exponentially in the beginning because the value of z is

$$z = \frac{s\ sin\theta}{\theta}$$

In the initial condition, since θ is taken as 0, the value of z becomes indefinite, and when θ is a very low value, the discretized model gives a large increment for z value.

The exponential increment leads to a lagging in the NMPC model. The model takes the first 10 seconds to adjust s, θ and ϕ values and reduces the error between the reference and x and y. z deviates from its reference to perform the following.

At the end, the x and y values drop while simulation in [1] shows an increment. This can be explained logically. Assuming a continuum robot whose end effector is at position (1, 1, 2) similar to time 50s, for z to drop to 1 the robot's end effector can move either forward and bend down or move backward and bend down. In [1], the robot bends forward while in the personal implementation, the robot bends backward. This is also shown by the opposing θ values.

3.2. Obstacle Avoidance

3.2.1. Results of Paper.
As shown in Figure 9

3.2.2. Results of Personal Implementation.
As shown in Figures 10 and 11

3.2.3. Parameter Selection. The initial state according to the simulation in [1] is taken as, x0 = [0, 0, 0.4]. The end goal is set as [1, 1, 1]. The obstacle coordinates are taken [0.5, 0.5, 0.5] which is exactly the midpoint between the origin and point [1, 1, 1]. The obstacle radius is chosen as 0.1m and the end effector tip radius is 0.1m.

The chosen obstacle coordinates ensure the robot needs to evade away from the shortest path. For this, a new constraint is introduced where the Euclidean distance between the obstacle's edge and the end effector's edge must be greater than 0. The constraint is given as,

$$-\infty \leq d + (radius_{obstacle} + radius_{robot}) \leq 0$$

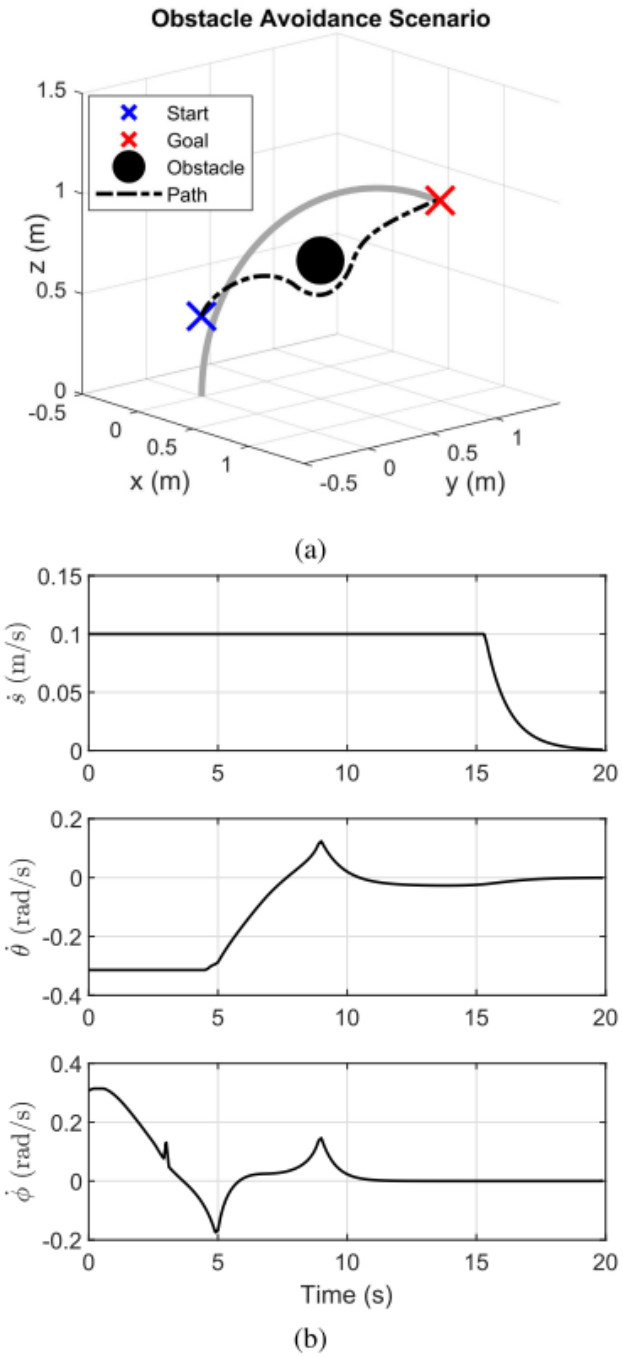

Figure 9. (a) Robot path taken by avoiding the obstacle, (b) Control inputs generated by NMPC model [1]

Here d is taken as,

$$d = \sqrt{x_{current} - x0}$$

The sampling time is taken, T_s = 0.1s

The simulation ends once the robot reaches the end point

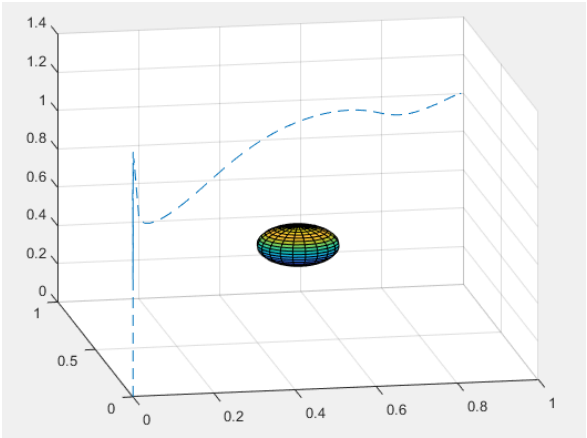

Figure 10. MATLAB simulation of obstacle avoidance

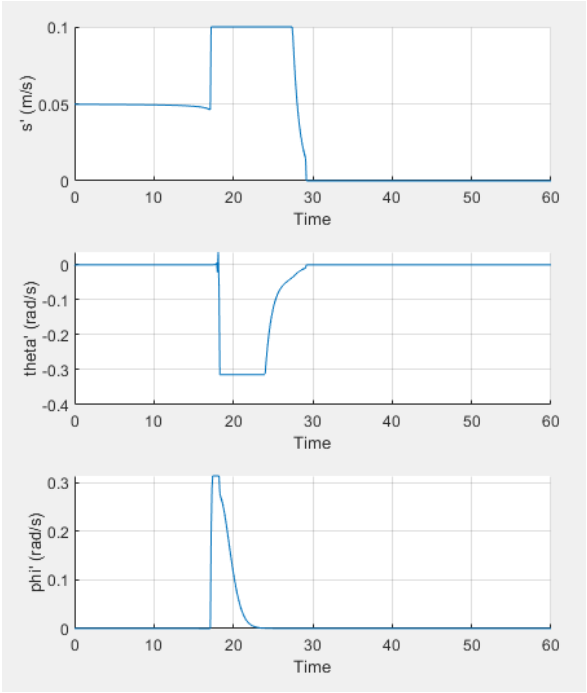

Figure 11. Control actions of MATLAB simulation

Prediction horizon is taken as, N = 10

Since the weight matrix parameters have not been defined. The state weight matrix, Q, is taken as a 6x6 matrix with diagonal values, Q = (1, 1, 1, 0, 0, 0)

The state weight matrix is taken as a 3x3 matrix with diagonal values, R = (0.5, 0.5, 0.5)

3.2.4. Discrepancies. Comparing results, the simulations are completely different.

1) z value shoots up to 1 immediately, while the z value in [1] increments slowly
2) The control actions are also different
3) The robot bends in the opposite direction compared to [1]

The z value shoots up immediately again due to the chosen model. Compared to the Point Stabilization simulation, the Obstacle Avoidance action was completed quickly. Since the NMPC is not given time to stabilize the abrupt z increment, the control actions vary by a lot.

Also, similar to the Point Stabilisation after 50s, the robot bends in the opposite direction. In [1], the robot bends and reaches the end point from below while the personal simulation goes upward. There must be a logical and algorithmic difference in implementation that makes it such that an opposing takes place on the z value. Either that or the model design contributes to this deviation.

3.2.5. Extra.

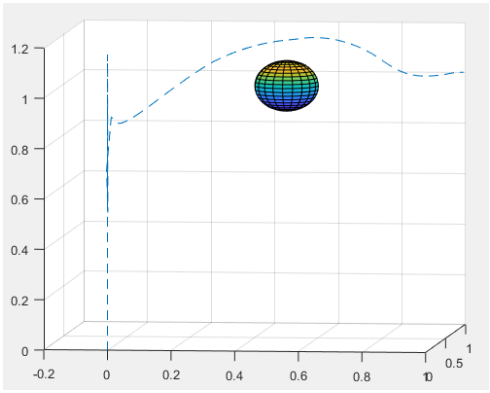

Figure 12. Extra testing of obstacle avoidance algorithm

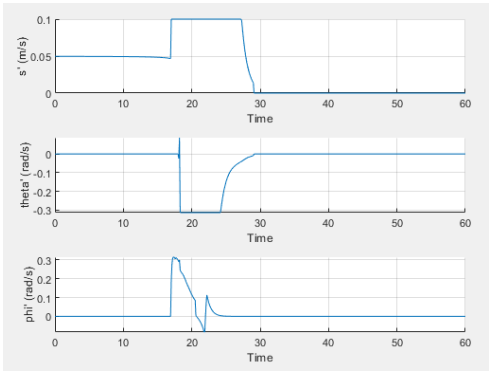

Figure 13. Control actions for obstacle avoidance algorithm testing

Changing the obstacle's z position to 1, makes it lie in between the trajectory path to the end point. From Figure 12, it can be seen that the obstacle avoidance aspect of the NMPC works as intended. It can also be seen that a bending away of the $\dot{\phi}$ is noticed as in [1].

3.3. Trajectory Tracking

3.3.1. Evaluation. There are many simulations performed in [1] for trajectory tracking. Although the reference path is the same in all simulations, the constraints and even methods displayed may vary. The NMPC results of [1] will be focused on and compared. Three major results are highlighted, NMPC tracking without constraints, NMPC reference tracking with constraints, and an explanation of the Jacobian controller. The reference to be tracked is as shown in Figure 14.

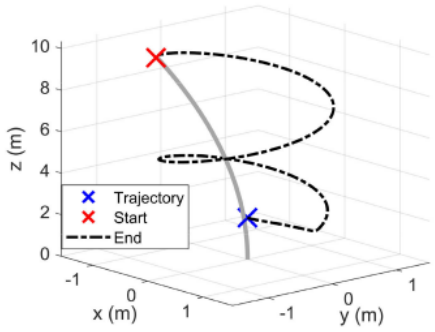

Figure 14. Trajectory tracking reference [1]

3.3.2. Results of Paper (1).

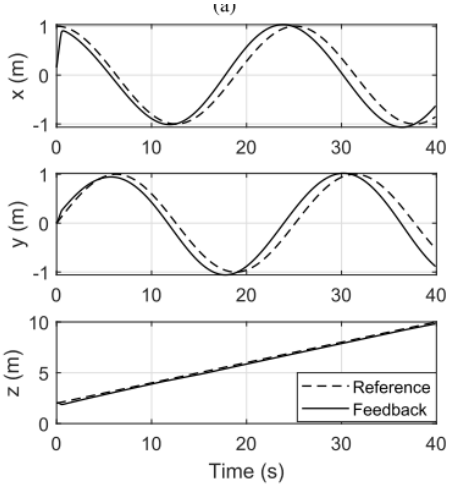

Figure 15. NMPC tracking without constraints [1]

3.3.3. Results of Personal Implementation (1).
As shown in Figure 16

3.3.4. Results of Paper (2).
As in Figure 17

3.3.5. Results of Personal Implementation (2).
As shown in Figure 18 and 19

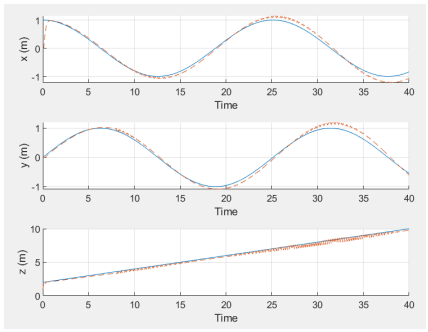

Figure 16. Matlab simulation without considering constraints

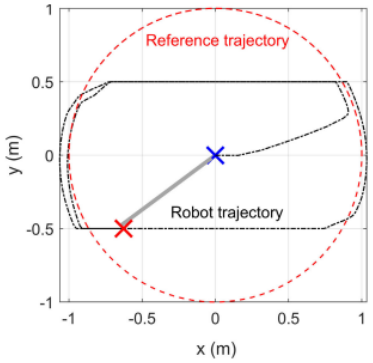

Figure 17. NMPC tracking with constraints [1]

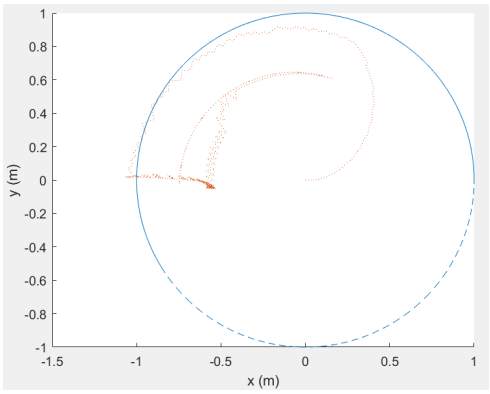

Figure 18. Enter Caption

3.3.6. Parameter Selection. The initial state, x0, is chosen as [0, 0, 0.9, 0.9, 0, 0]. Comparing the initial state to the direct kinematic equations (11), taking s, θ and ϕ values as 0.4, 0 and 0 gives undefined x, y and z values. This is especially because θ is in the denominator. To derive the values of x, y and z, we take limit θ tending to 0.

$$\lim_{\theta \longrightarrow 0} \frac{cos\theta - 1}{\theta} = 0$$

$$\lim_{\theta \longrightarrow 0} \frac{sin\theta}{\theta} = 1$$

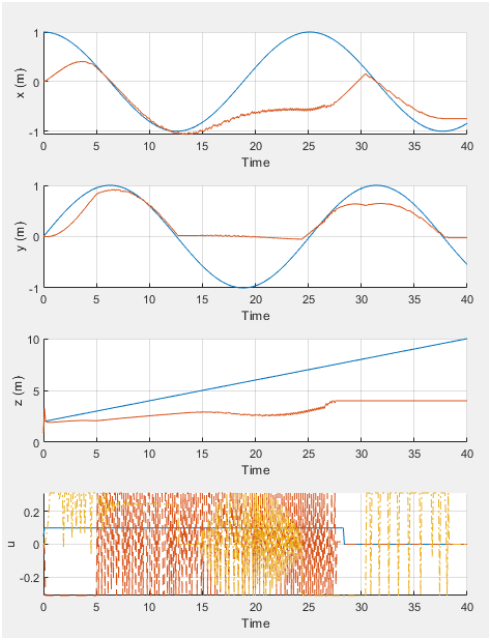

Figure 19. Matlab simulation with considering constraints

We get,

$$x = 0$$

$$y = 0$$

$$z = s$$

The reference trajectory is taken as a function of time, t such that,

$$x_{ref} = cos(0.25t)$$

$$y_{ref} = sin(0.25t)$$

$$z_{ref} = 2 + 0.2t$$

The sampling time is taken, T_s = 0.1s

The simulation time is stated as 20s but when looking at simulation graph it can be seen that the value 20s is a mistake and simulation time is taken as, T = 40s

Prediction horizon is taken as, N = 20

The state weight matrix, Q, is taken as a 6x6 matrix with diagonal values, Q = (1, 1, 1, 0, 0, 0)

The state weight matrix is taken as a 3x3 matrix with diagonal values, R = (0.5, 0.5, 0.5)

3.3.7. Discrepancies. The simulation outputs of the NMPC are roughly similar for the unconstrained implementation. But for the constrained implementation there are major differences. First and foremost, the simulation from [1] only provides the robot movement in the XY plane, therefore the results cannot be inferred properly. Logically, the robot should be able to trace the circle since tracing the circle will require minor increments to θ and sinusoidal change in ϕ, thus the result in Figure 17 is vague due to lack of information.

Yet, the simulation in Figure 18 shows that the personal NMPC design doesn't work as intended with constraints applied. Figure 19 show bad values of $\dot{\phi}$ throughout the simulation and bad values of $\dot{\theta}$ between 5s and 27s. Looking closely at the z value at 0s shows the same exponential increment that occurs due to model design.

$\dot{\theta}$ and $\dot{\phi}$ shows garbage optimal control inputs until z reaches the max limit. After which the optimal control actions stabilize, yet the x and y values are unable to reach the reference. It should also be stated that, the robot's growth mechanics are being tested and not its movement. Therefore whence z reaches the max range, the robot will have to grow by moving along a circular path in the XY plane, which is not possible. It is not possible since the robot's own body will block and hinder any further growth. One reason for bad results could be that the robot's trunk length limit majorly affects the optimal control decision. It might be that the implementation in [1] uses the single-shooting method, and thus the parameter s in the system state will not be considered as a parameter for the optimal control problem.

3.3.8. Jacobian Controller. For the trajectory tracking case, a Jacobian-based controller is also created to compare results with their NPMC model [1]. The formula is provided but its actual implementation is not. The Jacobian controller will be looked at theoretically since implementation for the controller is unknown. But, the implementation for the controller is straight forward and results attained don't need actual implementation to analyse. Also, the Jacobian controller heavily relies on a properly designed plant model to work, which has not been provided. Therefore any and all results will be sub-optimal if not outright wrong. The formula is given below,

$$u = J^T K e$$

Where e is the error between the reference and current state and K is a diagonal gain matrix. The formula is almost reminiscent to a P controller, where K will be the proportional gain. The K is taken as an identity matrix thus the equation reduces to ,

$$u = J^T e$$

$$u = J^T x_{ref} - J^T x$$

Another thing to note, is that the Jacobian is a 3x3 matrix, but there exists 6 states. It is unknown how matrix multi-

plication is performed in such a scenario to obtain control inputs.

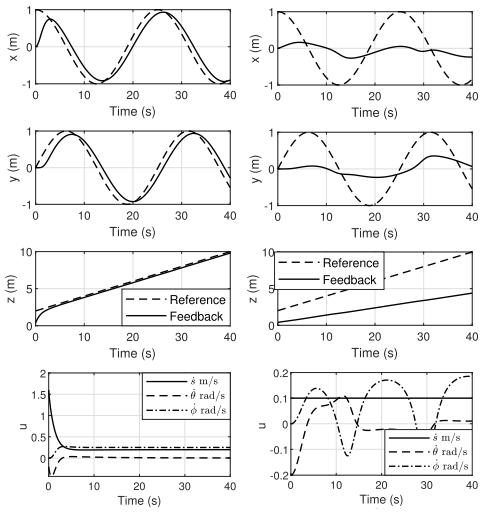

Figure 20. Jacobian controller simulation, unconstrained and constrained

Figure 20 contains simulation results form [1]. Since the states are a direct relation with the Jacobian, it is expected that the unconstrained simulation shows good results, especially due to no external gain values being chosen. On the other hand, for the constrained case, the results are lackluster, which is expected since the controller is focused on trying to eventually stabilize the system, but sinusoidal changes are hard to reference unless the controller design is accurate. The addition of constraints hampers the controller heavily.

3.4. Robustness Analysis

3.4.1. Evaluation. In [1], input disturbances are provided to the system to verify robustness of the algorithm. The variance of disturbance on the control actions, u, is chosen based on the constraints. Since the tests run in [1] have not been provided, a simple reference tracking simulation is performed.

3.4.2. Results of Paper. It is found that the Weight matrix combination,

$$Q = diag(10, 10, 1, 0, 0, 0)$$
$$R = diag(0.05, 0.05, 0.05)$$

shows good performance with input disturbances.

3.4.3. Results of Personal Implementation.
As shown in Figures 21, 22, 23 and 24

3.4.4. Simulation Results. It can be seen that when the x and y weight are considerable large compared to z weight, the simulation shows good outputs. This can also be attributed to the model design where a small increment in θ

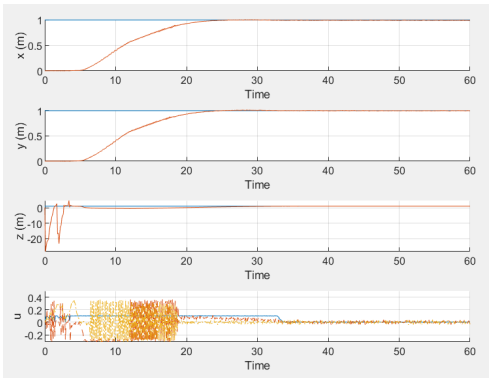

Figure 21. Q = diag(10, 10, 1, 0, 0, 0), R = diag(0.1, 0.1, 0.1)

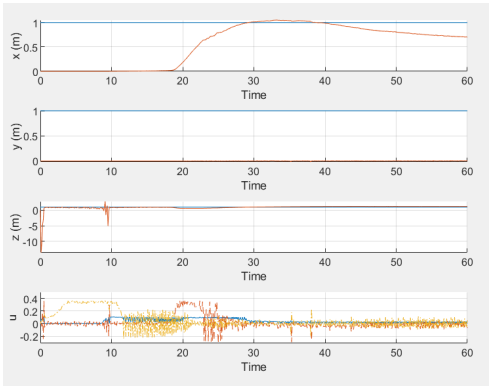

Figure 22. Q = diag(10, 10, 10, 0, 0, 0), R = diag(0.1, 0.1, 0.1)

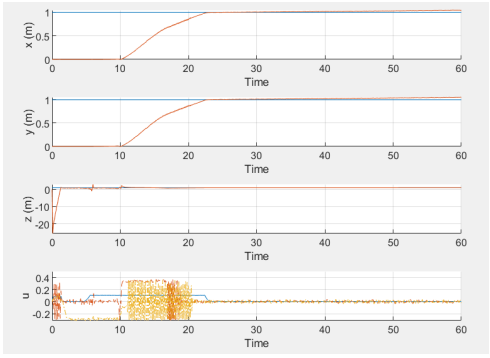

Figure 23. Q = diag(10, 10, 1, 0, 0, 0), R = diag(1, 0.1, 0.1)

highly affects the z value. It can be seen that Figure 23 shows best result, while for [1], best result is attained from parameters in Figure 24. This can also be attributed to the derived plant model. For Figure 24, the weights in control actions are equivalent, but the weights in Figure 23 try to minimize the value of $\dot{s}$ the most. Since any small $\dot{\theta}$ value shoots the z value to the reference, the optimizer chooses control actions where $\dot{\theta}$ is small because that drives the z error to 0.

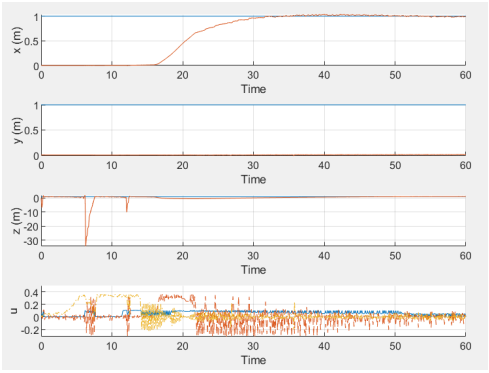

Figure 24. Q = diag(10, 10, 1, 0, 0, 0), R = diag(0.05, 0.05, 0.05)

4. Limitations in Design

The implementation in [1] directly takes the position kinematics of a continuum robot and uses it to derive its plant model. Only the analytical method of calculating the missing Jacobian term is provided where,

$$J = \frac{\partial p}{\partial(s, \theta, \phi)}$$

The following operations gives matrix components that can become undefined. The main one being, when $\theta = 0$. In [1], all initial conditions for simulation take θ as 0. Therefore the model attained from the above operation has either been limited or converted into a form where, such errors will not affect the system states. This information is not provided and can only be inferred, or the problem can be evaded by selecting very small θ values as the initial state.

Yet taking θ as a very small value leads to other errors. Looking at the equation of the z value we have,

$$z = \frac{sin\theta}{\theta}\dot{s} + \frac{s\theta cos\theta - sin\theta}{\theta^2}\dot{\theta}$$

There is a θ^2 term in the second z expression, which will lead to an exponential value of z, especially when $\lim_{\theta \longrightarrow 0^+}$ and $\lim_{\theta \longrightarrow 0^-}$ is not equal. There is no value for $\lim_{\theta \longrightarrow 0}$. Therefore the plant model used in [1] has been edited in a way that has not been mentioned.

CasAdi has an initial learning curve where the documentation is not easily accessible. When using the derived plant model, CasAdi might find optimal control actions that can lead to NaN error. This NaN error popped up many times during the algorithmic implementation.

The implementation in [1] doesn't provide the solver used for the optimal problem and even the algorithm is not provided.

It can be noticed from the comparison of the results that most, if not all, problems occur because of the model design or multi-shoot algorithm implemented.

5. Conclusion

In this paper, the findings and propositions in [1] have been analyzed, designed, implemented, simulated, and compared.

First, the plant model to be used is derived. Derivations are done using the minimal equations provided in [1], and by using findings from [2] and [3]. The velocity kinematic equation is taken as the plant model.

Next, the NMPC scheme is designed as in [1]. System states, [x, y, z, s, θ, ϕ] are chosen and the velocity kinematic equation is brought into the state space form. From the state space form, the control actions are taken as [$\dot{s}$, $\dot{\theta}$, $\dot{\phi}$]. The cost function and inequality constraints are then defined for the optimization problem to find the optimal control action. Also, the continuous-time model is discretized using the Euler discretization scheme to be used as the internal plant model for the NMPC.

Finally, all simulation results are compared and reasons are proposed for discrepancies in NMPC results. Four types of tests are run namely, point stabilization, obstacle avoidance, trajectory tracking, and robustness analysis. It can be seen that the simulation results vary a lot, which can be attributed to the lack of information regarding the plant model, which has been reduced to a simpler form in [1], and the algorithm employed.

References

[1] H. El-Hussieny, I. A. Hameed and J. -H. Ryu, "Nonlinear Model Predictive Growth Control of a Class of Plant-Inspired Soft Growing Robots," in *IEEE Access*, vol. 8, pp. 214495-214503, 2020, doi: 10.1109/ACCESS.2020.3041616.

[2] B. A. Jones and I. D. Walker, "Kinematics for multisection continuum robots," *IEEE Trans. Robot.*, vol. 22, no. 1, pp. 43–55, Feb. 2006.

[3] M. W. Hannan and I. D. Walker, "Kinematics and the implementation of an elephant's trunk manipulator and other continuum style robots," *J. Robot. Syst.*, vol. 20, no. 2, pp. 45–63, Feb. 2003.