California Institute of Technology

Pasadena, California

ME/CS/EE 133A : Robotics

Fall 2022

Final Project Report

for

Self Untangling Robotic Snake Arm with Dynamic Obstacle Avoidance

Submitted by

Team Name : Rattlestar

Team Members : Sri Aditya Deevi & Jeff Chen

9 December, 2022

I Problem Description

1 Problem Statement

For this problem, we are considering a simulated environment where there are obstacles (e.g. rocks) falling vertically. The robot being used is a redundant snake arm. The robotic snake arm has to perform the following tasks :

- Avoid the obstacles hitting its body.
- Touch a random target (e.g. a colored cube) in the workspace, with the correct gripper orientation.
- If initially entangled in a loop, it has to untangle itself and then repeat, the above mentioned tasks.

2 System Description

2.1 Robot Types

For this project, RVIZ is being utilized. We built and used the following types of snake robots for analyzing and comparing the performance (see Section IV).

XY pattern snake:

We refer to a snake robot URDF with 7 DOFs from the GitHub and extend it to 41 DOFs using a URDF generating package [*].

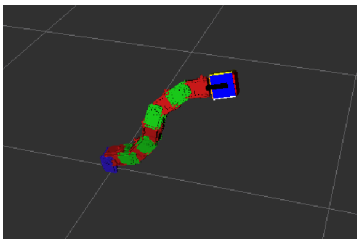


Figure 1: XY pattern 9 DoFs

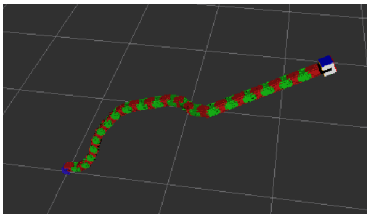


Figure 2: XY pattern 41 DoFs

YZ pattern snake:

For this, we consider the `snakebot` URDF provided by the professor. Here we have revolute joints with z and y as axes of rotation.

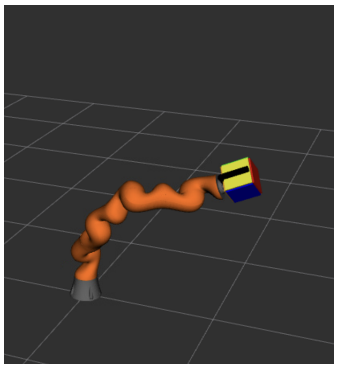


Figure 3: YZ pattern 9 DoFs

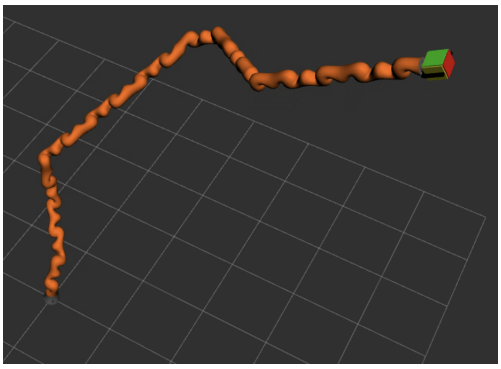


Figure 4: YZ pattern 41 DoFs

Also, note that a gripper with a cube is attached to the tip of the robot to illustrate that the tip orientation is matching the target's orientation.

3D Meshes

We use the following 3D meshes of a rock and a rubik's cube (target) to make the scene more vivid and realistic.

[*] `https://github.com/hauptmech/odio_urdf`

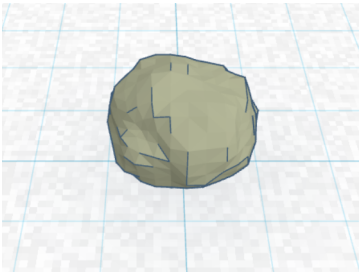


Figure 5: Mesh of Rock Obstacle

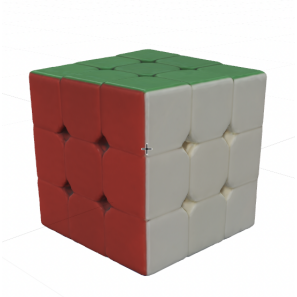


Figure 6: Mesh of Target Cube

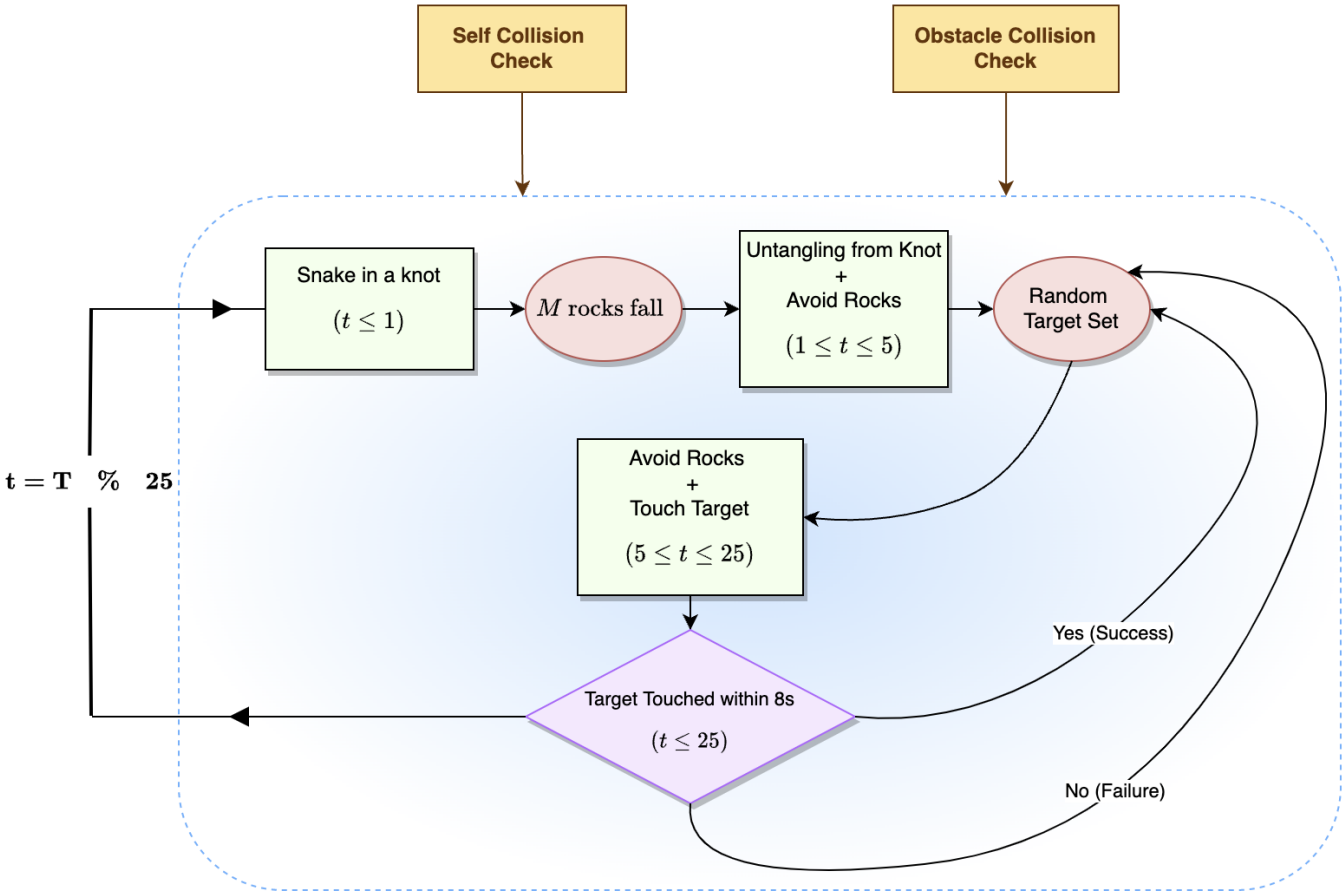
3 Applications

A snake robot is useful for exploring some dangerous environments. For example, in a Mars exploration mission, a snake robot can easily pass through a narrow cave or a thin trench and use its gripper to collect samples. Or in a rescue mission after the earthquake, the snake robot can sneak into a collapsed building while avoiding falling rocks using the algorithm implemented in this project.

II Methodology

High Level Block Diagram

The following block diagram illustrates the different phases of a typical experiment:



Now we will focus and zoom into different aspects of the methodology.

1 Dynamic Obstacle Avoidance

1.1 Formulation

We assume that we know the position of the falling obstacles*. Here, we consider that at a given point of time, there are M rocks or obstacles and use the following strategy:

(i) Find the joint that is closest to an obstacle in the snake's body and call it a "tip". Define multiple tips (# Tips =# Obstacles = M) in the snake's body, where each tip corresponds to an obstacle :

$$\text{tipdof}, j = \underset{i}{\text{argmin}} \quad \left\| {}^0 \mid \vec{p}_{\text{obs},j} - {}^0 \mid \vec{p}_i \right\| \tag{1}$$

(ii) Compute all the M corresponding tip Jacobians ${J_{v,\text{tip},j}}^{\dagger}$ (i^{th} column is computed here):

$$(J_{v,\text{tip},j})_i = \begin{bmatrix} {}^0 \mid \vec{e}_i \times \left({}^0 \left| \vec{p}_{\text{tipdof},j} - {}^0 \right| \vec{p}_i \right) \end{bmatrix} \tag{2}$$

(iii) Concatenate all the primary task quantities as follows:

$$J_{v,\text{avoid, eff}} = \begin{bmatrix} J_{v,\text{tip},1} \\ J_{v,\text{tip},2} \\ \vdots \\ J_{v,\text{tip},M} \end{bmatrix} \quad ; \quad {}^0 \mid \vec{p}_{\text{obs, eff}} = \begin{bmatrix} {}^0 \mid \vec{p}_{\text{obs},1} \\ {}^0 \mid \vec{p}_{\text{obs},2} \\ \vdots \\ {}^0 \mid \vec{p}_{\text{obs},M} \end{bmatrix} \quad ; \quad {}^0 \mid \vec{p}_{\text{tipdof, eff}} = \begin{bmatrix} {}^0 \mid \vec{p}_{\text{tipdof},1} \\ {}^0 \mid \vec{p}_{\text{tipdof},2} \\ \vdots \\ {}^0 \mid \vec{p}_{\text{tipdof},M} \end{bmatrix}$$

It should be mentioned that we are only considering the XY movements (repulsions) when we do this because the obstacles are falling in the Z-direction. This means that all positions in this equation are of the form $\vec{p}_r = \begin{bmatrix} x_r \\ y_r \end{bmatrix} \in \mathbb{R}^2$. Also, extracting first two rows, $J_{v,\text{tip},j}$ is a $(2 \times \#\text{ DoFs })$ matrix.

(iv) Use the following equation to the effective joint velocities:

$$\dot{q}_{\text{avoid}} = \lambda J^{+}_{v,\text{avoid, eff}} \frac{\left({}^0 \mid p_{\text{tipdof, eff}} - {}^0 \mid p_{\text{obs, eff}} \right)}{\left\| {}^0 \mid p_{\text{tipdof, eff}} - {}^0 \mid p_{\text{obs, eff}} \right\|^2} \tag{3}$$

where $J^{+}_{v,\text{avoid, eff}}$ is the weighted pseudo inverse. Intuition is that the robot's movement (repulsion) should be away from the obstacle and its magnitude should be inversely proportional to the obstacle's distance from its body (in the XY plane).

1.2 Calculation of Jacobian

Equation (2) describes the calculation of i^{th} column of the tip jacobian. We have explored two different formulations as described:

Robotic Jacobian –

$$\begin{aligned} (J_{v,\text{tip},j})_i &= \begin{bmatrix} {}^0 \mid \vec{e}_i \times \left({}^0 \left| \vec{p}_{\text{tipdof},j} - {}^0 \right| \vec{p}_i \right) \end{bmatrix} \quad ; \quad i \leq \text{tipdof}, j \\ (J_{v,\text{tip},j})_i &= \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad ; \quad i > \text{tipdof}, j \end{aligned} \tag{4}$$

*In real life, this can be hopefully estimated using object pose estimation AI algorithms

†Some details about Jacobian calculation are described later

Biological Jacobian –

$$(J_{v,\text{tip},j})_i = \begin{bmatrix} {}^0 \mid \vec{e}_i \times \left({}^0\left|\vec{p}_{\text{tipdof},j} - {}^0\right|\vec{p}_i\right) \end{bmatrix} \quad ; \quad \forall i = 1,2,\ldots,\#DoFs \tag{5}$$

We analyze how the snake's movement is affected by using these two different formulations in Section IV. But for the majority of the experiments we utilize the robotic jacobian as it provides more "freedom" to the snake (to dodge obstacles).

2 Touching Target

In this section, we discuss schemes we used for the target touching. The objective in this task is that the gripper cube should match that of the target cube[‡]. Let the position and orientation of the target be p_{goal} and R_{goal}, and the position and orientation of the end tip (at an instant) be p_0 and R_0. We determine the movement to approach the goal in the following two steps:

2.1 Finding Axis and Angle of Rotation

We want to find a single rotation (axis $\vec{e}$ and angle θ_{goal}) that takes from the current end tip pose to the goal pose. In other words, we can find a rotation matrix $Rot(\vec{e},\theta_{\text{goal}})$ such that $R_{goal} = Rot(\vec{e},\theta_{\text{goal}}) \cdot R_0$. Note that $\vec{e}$ is the eigenvector for $Rot(\vec{e},\theta_{\text{goal}}) = R_{\text{goal}} \cdot R_0^T$, so we derive $\vec{e}$ and θ_{goal} through the following calculation:

(i) For the general case, we know that:

$$\text{Rot}(\vec{e},\theta_{\text{goal}}) = \begin{bmatrix} e_x e_x(1-c_\theta)+ & c_\theta & e_x e_y(1-c_\theta) - e_z s_\theta & e_x e_z(1-c_\theta) + e_y s_\theta \\ e_y e_x(1-c_\theta) + e_z s_\theta & e_y e_y(1-c_\theta)+ & c_\theta & e_y e_z(1-c_\theta) - e_x s_\theta \\ e_z e_x(1-c_\theta) - e_y s_\theta & e_z e_y(1-c_\theta) + e_x s_\theta & e_z e_z(1-c_\theta)+ & c_\theta \end{bmatrix}$$

where $s_\theta = \sin(\theta_{\text{goal}}), c_\theta = \cos(\theta_{\text{goal}})$, and $\vec{e} = \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix}$.

So, since we know $Rot(\vec{e},\theta_{\text{goal}})$, say :

$$Rot(\vec{e},\theta_{\text{goal}}) = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Then, when $|sin(\theta_{goal})| >> 0$, we can easily get $\vec{e}$ and θ_{goal} using the following relations:

$$\vec{e} = \begin{bmatrix} h-f \\ c-g \\ d-b \end{bmatrix} \quad ; \quad ||\vec{e}|| = 2sin(\theta_{\text{goal}})$$

(ii) When $sin(\theta_{\text{goal}}) \approx 0$ we can to obtain three eigenvectors and eigenvalues for $Rot(\vec{e},\theta_{\text{goal}})$ {say using `numpy.eig`}:

$$\vec{v} = [\vec{v}_1, \vec{v}_2, \vec{v}_3]$$

$$\lambda = [1, e^{i\theta_{\text{goal}}}, e^{-i\theta_{\text{goal}}}]$$

$\vec{e}$ would be $\vec{v}_1$, which has a corresponding eigenvalue of 1, and using the remaining eigenvalues $e^{\pm i\theta_{\text{goal}}}$ we can obtain θ_{goal}.

[‡]Both of them are colored for visual ease

2.2 Calculating Trajectories

Given $Rot(\vec{e},\theta_{\text{goal}})$, we then explore three different trajectory generation and execution schemes and define $\dot{q}_{\text{touch}}$ in three ways:

(i) **spline** Scheme –

First, we can define a cubic spline for x and R :

$$x(t),\dot{x}(t) = \text{spline}(x_0, x_{\text{goal}}, 6sec)$$

$$R(t) = Rot(\vec{e},\theta)\cdot R_0, \quad \theta(t),\dot{\theta}(t) = \text{spline}(0,\theta_{\text{goal}}, 6sec)$$

Then $\dot{q}_{\text{touch}}$ would be:

$$\dot{q}_{\text{touch}} = J^{+}_{\text{touch}} \begin{bmatrix} x + \lambda_v e_P(x, x(q)) \\ \hat{e}\dot{\theta} + \lambda_w e_R(R, R(q)) \end{bmatrix} \tag{6}$$

where $\hat{e}$ is the unit eigenvector derived in the previous section and $J_{\text{touch}} = \begin{bmatrix} J_v \\ J_w \end{bmatrix}$.

(ii) **attract** Scheme –

Here we make the tip approach the goal directly without using velocities derived from splines :

$$\dot{q}_{\text{touch}} = J^{+}_{\text{touch}} \begin{bmatrix} \lambda_v e_P(x_{\text{goal}}, x(q)) \\ \lambda_w e_R(R_{\text{goal}}, R(q)) \end{bmatrix} \tag{7}$$

where $J_{\text{touch}} = \begin{bmatrix} J_v \\ J_w \end{bmatrix}$.

(iii) **spline+attract** Scheme –

We first make the tip to approach the goal by spline movement, after the spline is complete, if the tip hasn't reached the goal (because touching targets is not always the primary task), then we use attraction movement to approach the goal.

$$\dot{q}_{\text{touch}} = \begin{cases} J^{+}_{\text{touch}} \begin{bmatrix} x + \lambda_v e_P(x, x(q)) \\ \hat{e}\theta + \lambda_w e_R(R, R(q)) \end{bmatrix}, & t \le 6 \text{ sec} \\ \\ J^{+}_{\text{touch}} \begin{bmatrix} \lambda_v e_P(x_{\text{goal}}, x(q)) \\ \lambda_w e_R(R_{\text{goal}}, R(q)) \end{bmatrix}, & 6 < t \le 8 \text{ sec} \end{cases} \tag{8}$$

where $\hat{e}$ is the unit eigenvector derived in the previous section and $J_{\text{touch}} = \begin{bmatrix} J_v \\ J_w \end{bmatrix}$.

Checkbox for target

We also created a checkbox GUI (based on a Boolean Publisher) to determine whether the snake should approach the target (while also avoiding rocks) or just avoid the obstacles. This just changes the priorities of the snake in a certain way and can help us focus on the performance of obstacle avoidance.

3 Self Knots in the Snake

3.1 Starting in a Tangled Pose

Since we are studying techniques to untangle a snake robot when it is stuck in a knot, the first step we took was to formalize the notion of a knot and initialize the robot in a knotted state. For this purpose, we are considering the trefoil knot, which is the simplest non-trivial knot (as shown in Figure. 7).

Figure 7: Illustration of a Trefoil Knot

This knot can be represented by the following parametric equations, $t \in [0, 2\pi)$:

$$x = \cos t + 2\cos 2t$$
$$y = \sin t - 2\sin 2t$$
$$z = -\sin 3t$$

We want the snake's body[§] be in the shape described by this knot. We can see that $t = 0$ and $t = 2\pi$ corresponds to the same point in the 3D space. This would mean that if we consider the entire interval $[0, 2\pi)$, the snake's tip would be touching its base, which is not desired. So, we use (2/3) of this interval, i.e. $t \in [\frac{\pi}{3}, \frac{5\pi}{3}]$ (See Figure 8).

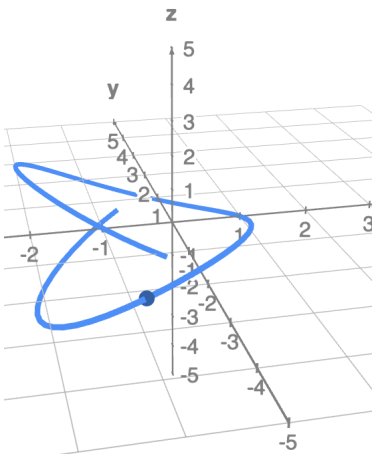


Figure 8: Partial Trefoil Knot, $t \in [\frac{\pi}{3}, \frac{5\pi}{3}]$

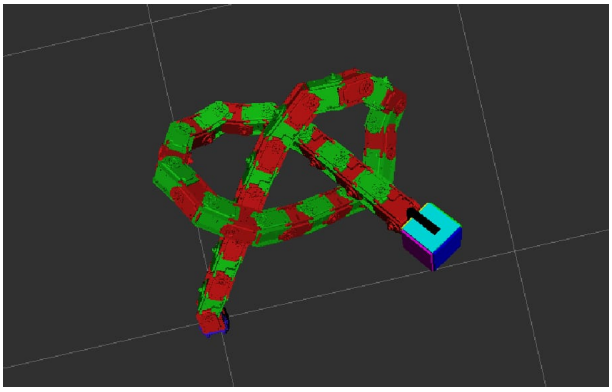


Figure 9: XY snake initialized in a partial trefoil knot

We use the following strategy to initialize the snake in a random trefoil knot (randomization of knots is discussed later) as shown in Figure 9:

(i) We consider Q equally (and sufficiently) spaced points (in 3D space) from the parametric curve, where each point maps to a joint in a snake's body. Note that we only consider a portion of the snake's body as mentioned earlier.

(ii) These points in 3D form the ${}^0 \mid p_{\text{goal},j}$ for the corresponding snake's joints.

(iii) The joints corresponding to these points are considered as "tips" and we compute all the tip jacobians $J_{v,\text{tip},j}$, $j = 1, 2, \ldots, Q$, (i^{th} column is computed here):

$$(J_{v,\text{tip},j})_i = \begin{bmatrix} {}^0 \mid \vec{e}_i \times \left({}^0 \left| \vec{p}_{\text{tipdof},j} - {}^0 \right| \vec{p}_i\right) \end{bmatrix}$$

(iv) Concatenate all the quantities as follows:

[§]While implementing we considered only a portion of the robot (leaving the few links) to be part of the knot, because that seemed to form a more visually pleasing knot!

$$J_{v,\text{tip,eff}} = \begin{bmatrix} J_{v,\text{tip},1} \\ J_{v,\text{tip},2} \\ \vdots \\ J_{v,\text{tip},Q} \end{bmatrix} \quad ; \quad {}^0 \mid \vec{p}_{\text{goal, eff}} = \begin{bmatrix} {}^0 \mid \vec{p}_{\text{goal},1} \\ {}^0 \mid \vec{p}_{\text{goal},2} \\ \vdots \\ {}^0 \mid \vec{p}_{\text{goal},Q} \end{bmatrix} \quad ; \quad {}^0 \mid \vec{p}_{\text{tipdof, eff}} = \begin{bmatrix} {}^0 \mid \vec{p}_{\text{tipdof},1} \\ {}^0 \mid \vec{p}_{\text{tipdof},2} \\ \vdots \\ {}^0 \mid \vec{p}_{\text{tipdof},Q} \end{bmatrix}$$

(v) Then we use the Newton Raphson's Algorithm iteratively, to get the final joint configuration so that the snake is placed in a trefoil knot:

$$q(k) = q(k-1) + J^{-1}_{v,\text{tip,eff}}(q(k-1)) \begin{bmatrix} \underbrace{{}^0 \mid \vec{p}_{\text{goal, eff}}}_{\text{Constant for a knot}} & -{}^0 \mid \vec{p}_{\text{tipdof, eff}} \end{bmatrix}$$

3.2 Untangling : Self-Repulsion Formulation

Now that the snake is initialized in a knot, we formulated a self repulsion mechanism so that the snake untangles itself from the knot. For this, we use the following strategy:

(i) For all the links in the kinematic chain, we obtain the position vector along the link which can be obtained by walking up the chain (shown for link i here, $i = 1, 2, \ldots, \#\text{DoFs} - 1$) as follows:

$${}^0 \mid \vec{p}_{\text{L},\,i} = {}^0 \mid \vec{p}_{i+1} - {}^0 \mid \vec{p}_i$$

(ii) Now, for all the links j succeeding this link i ($\forall j > i$), we check if it is too close to link i (and this is again done for all links $i = 1, 2, \ldots, \#\text{DoFs} - 1$)):

$$\underbrace{\left\| {}^0 \mid \vec{p}_{\text{L},\,j} - \frac{\langle {}^0 \mid \vec{p}_{\text{L},\,j}, {}^0 \mid \vec{p}_{\text{L},\,i} \rangle}{\left\| {}^0 \mid \vec{p}_{\text{L},\,i} \right\|^2} \right\|}_{\text{Perpendicular Distance between Link } i \text{ and } j} < 2.8 L_{link} \quad \underline{\text{and}} \quad 0 < \langle {}^0 \mid \vec{p}_{\text{L},\,j}, {}^0 \mid \vec{p}_{\text{L},\,i} \rangle < \left\| {}^0 \mid \vec{p}_{\text{L},\,i} \right\|^2$$

where L_{link} is the length of each link in the snake's body.

(iii) If the condition mentioned in (ii) is satisfied (say for joint z), we set a desired repulsive vector for the link z as follows (this is done for all links j that satisfy the condition mentioned in (ii)) :

$${}^0 \mid \vec{p}_{\text{repel},z} = \frac{{}^0 \mid \vec{p}_{\text{L},\,z} - \left(\frac{{}^0 \mid \vec{p}_{\text{L},\,i} + {}^0 \mid \vec{p}_{\text{L},\,i+1}}{2} \right)}{\left\| {}^0 \mid \vec{p}_{\text{L},\,z} - \frac{\langle {}^0 \mid \vec{p}_{\text{L},\,z}, {}^0 \mid \vec{p}_{\text{L},\,i} \rangle}{\left\| {}^0 \mid \vec{p}_{\text{L},\,i} \right\|^2} \right\|^2}$$

Intuition is that link z, which is too close to link i should move away from the midpoint of link i. We concatenate all such vectors to form ${}^0 \mid \vec{p}_{\text{repel, eff}}$.

(iv) We also compute the "tip" jacobians for all such links z as follows (r^{th} column is computed here):

$$(J_{v,\text{tip},z})_r = \begin{bmatrix} {}^0 \mid \vec{e}_r \times \left({}^0 \left| \vec{p}_{\text{tipdof},j} - {}^0 \right| \vec{p}_r \right) \end{bmatrix} \quad ; \quad r > \text{tipdof}, i$$

$$(J_{v,\text{tip},z})_r = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad ; \quad r \leq \text{tipdof}, i$$

Intuition is that joints 1 to i need not move for the repulsion of link z away from the midpoint of link i. We concatenate all such jacobians to form $J_{v,\text{repel, eff}}$.

(v) Use the following equation to the effective joint velocities:

$$\dot{q}_{\text{repel}} = \lambda J^{+}_{v,\text{repel, eff}} \, {}^0 \mid \vec{p}_{\text{repel, eff}} \tag{9}$$

where $J^{+}_{v,\text{repel, eff}}$ is the weighted pseudo inverse.

A demonstration of the snake robot coming out of the loop is provided with the corresponding project video.

4 Task Specifications

In this section, we talk about how we compute the $\dot{q}$ values in different phases of the experiments:

4.1 Joint Velocity : Different Tasks

In conclusion to this section, the $\dot{q}_r$ and Jacobian J_r for different tasks can be summarized as follows (Note that for calculating “tip" jacobians we use Robotic Jacobian definition as mentioned in (4)) :

(i) Dynamic Obstacle Avoidance –

This equation can be obtained from (3) :

$$\dot{q}_{\text{avoid}} = \lambda J^{+}_{v,\text{avoid, eff}} \frac{\left({}^{0}\mid p_{\text{tipdof, eff}} - {}^{0}\mid p_{\text{obs, eff}}\right)}{\left\| {}^{0}\mid p_{\text{tipdof, eff}} - {}^{0}\mid p_{\text{obs, eff}} \right\|^{2}}$$

(ii) Touching Target –

This equation can be obtained from (6), (7) and (8) depending upon the trajectory execution scheme :

$$\dot{q}_{touch} = J^{+}_{touch} A$$

where

$$A = \begin{cases} \begin{bmatrix} x + \lambda_v e_P(x, x(q)) \\ \hat{e}\theta + \lambda_w e_R(R, R(q)) \end{bmatrix} & \forall t, & \textbf{spline} \text{ Scheme} \\ \begin{bmatrix} \lambda_v e_P(x_{\text{goal}}, x(q)) \\ \lambda_w e_R(R_{\text{goal}}, R(q)) \end{bmatrix} & \forall t, & \textbf{attract} \text{ Scheme} \\ \begin{bmatrix} x + \lambda_v e_P(x, x(q)) \\ \hat{e}\theta + \lambda_w e_R(R, R(q)) \end{bmatrix} & t \in [0,6]; \quad \begin{bmatrix} \lambda_v e_P(x_{\text{goal}}, x(q)) \\ \lambda_w e_R(R_{\text{goal}}, R(q)) \end{bmatrix} \quad t \in (6,8] & \textbf{spline+attract} \text{ Scheme} \end{cases}$$

(iii) Self Untangling –

This equation can be obtained from equation (9) :

$$\dot{q}_{\text{repel}} = \lambda J^{+}_{v,\text{repel, eff}} {}^{0}\mid \vec{p}_{\text{repel, eff}} \tag{9}$$

(iv) Maintain Nominal Joint Positions –

This task is added so that the snake robot stays close to a nominal joint configuration (can be the optimal in the real world). In the case of untangling phase, this term provides the extra directional pull for the snake to come out of the loop :

$$\dot{q}_{\text{nom}} = \lambda(q_{\text{nom}} - q)$$

4.2 Joint Velocity : Different Phases

For different phases of our typical experiments, there are different priorities for tasks. We describe our formulation in this section. Note that in this section, the null space for a task is given by the expression $\Phi_{\text{gen}} = (I - J^{+}_{\text{gen}} J_{\text{gen}})$, where J_{gen} is the Jacobian for the task.

(i) Stuck in a knot (Stays Still) –

$$\dot{q} = 0$$

(ii) Self Untangling + Dynamic Obstacle Avoidance –

Primary Task → Avoid Rocks ; Secondary Task → Untangle ; Tertiary Task → Nominal Joint Position

$$\dot{q} = \dot{q}_{\text{avoid}} + \Phi_{\text{avoid}}(\dot{q}_{\text{repel}} + \Phi_{\text{repel}} \dot{q}_{\text{nom}})$$

(iii) Target Touch + Dynamic Obstacle Avoidance –

Primary Task → Avoid Rocks ; Secondary Task → Touch Target ; Tertiary Task → Nominal Joint Position

$$\dot{q} = \dot{q}_{\text{avoid}} + \Phi_{\text{avoid}}(\dot{q}_{\text{target}} + \Phi_{\text{target}} \dot{q}_{\text{nom}})$$

(iv) Pure Dynamic Obstacle Avoidance –

This happens when the target checkbox is unchecked.

Primary Task → Avoid Rocks ; Secondary Task → Nominal Joint Position

$$\dot{q} = \dot{q}_{\text{avoid}} + \Phi_{\text{avoid}} \dot{q}_{nom}$$

III Implementation Details

1 Randomization

In this section, we discuss some of the implementation details with the respect to the randomization of obstacles, targets, and knots so that we can incorporate more diversity and understand if the robot is able to handle any general, *random* situation.

1.1 Rock Obstacles

We mainly randomize the (x, y, z) position of where the obstacle starts to fall from. To make sure that the path of some obstacles definitely intersects with any of the snake robot's links, we use the following strategy:

$$\begin{aligned} x &= {}^{0}|\,\vec{x}_{i+1} u_1 + {}^{0}|\,\vec{x}_i (1 - u_1) + \epsilon_1 \\ y &= {}^{0}|\,\vec{y}_{i+1} u_2 + {}^{0}|\,\vec{y}_i (1 - u_2) + \epsilon_2 \\ z &= 4 + \epsilon_3 \end{aligned}$$

where $\epsilon_1, \epsilon_2 \sim \mathcal{N}(0, 0.2)$; $u_1, u_2 \sim \mathcal{U}(0,1)$; $\epsilon_3 \sim \mathcal{N}(0,1)$. Also ${}^{0}|\,\vec{p}_{i+1} = \begin{bmatrix} {}^{0}|\,\vec{x}_{i+1} \\ {}^{0}|\,\vec{y}_{i+1} \\ {}^{0}|\,\vec{z}_{i+1} \end{bmatrix}$ and ${}^{0}|\,\vec{p}_{i} = \begin{bmatrix} {}^{0}|\,\vec{x}_{i} \\ {}^{0}|\,\vec{y}_{i} \\ {}^{0}|\,\vec{z}_{i} \end{bmatrix}$ are the position vectors of the i^{th} and $(i+1)^{th}$ joints respectively. Note that i is randomly chosen from the top portion of the snake, leaving a few joints from the base.

This is essentially an adversarial strategy, where we are targetedly "attacking" the snake and if it proves to dodge these obstacles effectively, it is highly likely that this method would work well in the real world, where there is no targeted bombarding of obstacles at the snake robot.

1.2 Target Cube

We also randomize the position (x_T, y_T, z_T) and orientation quaternion $(q_{x,T}, q_{y,T}, q_{z,T})$ of the target cube as follows:

$$\begin{bmatrix} x_T \\ y_T \\ z_T \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \quad ; \quad \begin{bmatrix} q_{x,T} \\ q_{y,T} \\ q_{w,T} \\ q_{z,T} \end{bmatrix} = \begin{bmatrix} \sqrt{1-m}\sin(2\pi n) \\ \sqrt{1-m}\cos(2\pi n) \\ \sqrt{m}\sin(2\pi o) \\ \sqrt{m}\cos(2\pi o) \end{bmatrix}$$

where m, n, $o \sim \mathcal{U}(0,1)$.

1.3 Knots

We also randomize the self knots in which the robot initializes in the beginning of each cycle by randomizing the following aspects:

- The number of joints that constitute a knot are randomized by randomly choosing a startpoint (or a starting joint).
- We also randomize the relative orientation of the knot:

$$K_{\text{rot}} = \text{Rot}_x(\theta_1)\text{Rot}_z(\theta_2)K$$

where K is a $3 \times Q$ matrix consisting of Q sampled 3D points from the parametric knot equations; θ_1, $\theta_2 \sim \mathcal{U}(0, 2\pi)$.

2 Performance Monitoring Functions

We check how well the robot fulfills the desired task by monitoring the following aspects as follows:

2.1 Obstacle Collision Check

We check whether every joint is far enough from every marker:

$$\text{No Obstacle Collision} \equiv \{\, ||^0 \mid \vec{p}_j - {}^0 \mid \vec{p}_m|| > \epsilon_{\text{threshold}} \quad \forall \text{ joints } j \text{ and obstacles } m\}$$

In case this condition is violated at any point, then we note this as an *obstacle collision*.

2.2 Self Collision Check

We check whether every joint is far enough from every link as follows:

$$\underbrace{\left\| {}^0 \mid \vec{p}_{\text{L},\,j} - \frac{\langle {}^0 \mid \vec{p}_{\text{L},\,j}, {}^0 \mid \vec{p}_{\text{L},\,i} \rangle}{\left\| {}^0 \mid \vec{p}_{\text{L},\,i} \right\|^2} \right\|}_{\text{Perpendicular Distance between Link } i \text{ and } j} > 2.8L_{link} \quad \underline{\text{or}} \quad 0 > \langle {}^0 \mid \vec{p}_{\text{L},\,j}, {}^0 \mid \vec{p}_{\text{L},\,i} \rangle \quad \underline{\text{or}} \quad \langle {}^0 \mid \vec{p}_{\text{L},\,j}, {}^0 \mid \vec{p}_{\text{L},\,i} \rangle > \left\| {}^0 \mid \vec{p}_{\text{L},\,i} \right\|^2$$

In case this condition is violated at any point, then we note this as a *self collision*.

2.3 Touch Target Check

We check whether the position and orientation of the gripper cube are close to the target cube as follows:

$$\text{Target Touched} \equiv \{\, \left\| {}^0 \mid \vec{p}_{\text{target}} - {}^0 \mid \vec{p}_{\text{tip}} \right\| < \epsilon_{\text{threshold}} \;\&\; \left\| R_{\text{target}} - R_{\text{tip}} \right\| < \epsilon_{\text{threshold}}\}$$

Whenever this condition is satisfied, we note it as a target touch.

IV Results, Analysis and Inferences

We have conducted a number of experiments in order to collect observations and analyze the various aspects of the project.

1 Experimental Configuration

Here, we talk about the configuration for different types of experiments we conducted:

1.1 Experiment Dimensions

We perform experiments differing in the following factors:

(i) **Robot types**. XY with 41 DOFs, XY with 9 DOFs, YZ with 41 DOFs, and YZ with 9 DOFs.

(ii) **Trajectory Execution Scheme**. **spline**, **spline+attract**, **attract** schemes

(iii) **# Rocks (Instantaneous)**. The number of rocks falling on the snake robot at a given instant of time is varied from 0 to 5.

For each experiment, we implement four periods and record data in ROS bag to evaluate the performance. Each period lasts for 25 seconds with the same tasks order (as mentioned earlier) and time intervals:

(i) *Stuck in a knot (Stays Still)* for 1 second

(ii) *Self Untangling + Dynamic Obstacle Avoidance* for 4 seconds

(iii) *Target Touch + Dynamic Obstacle Avoidance* for 20 seconds. When a target is touched, a new random target would be generated.

1.2 Types of Graphical Plots

We plot the following data from `rqt_plot` and MATLAB:

(i) The number of total targets and touched targets (cumulative) to evaluate the success rate.

(ii) The number of robot self-collisions (non-cumulative)

(iii) The number of rocks that collide with the robot (cumulative)

(iv) Joint States

1.3 Quantitative Metric Definitions

We record the following data for further analysis:

(i) The number of total targets and touched targets (cumulative) to evaluate the target touch accuracy.

(ii) The number of rocks that collide with the robot (cumulative) and the total number of rocks falling to evaluate the rock avoidance accuracy.

(iii) Joint Velocities and Joint Positions.

2 Results of Notable Experiments

In this section, we will first present notable quantitative and qualitative results obtained from the experiments we conducted according to the experimental configuration mentioned in the previous section and then we try to make interesting inferences from the results after analysis.

2.1 Quantitative Results

Note

In all the tables shown in this subsection, for each number of rocks (instantaneous):

- 1^{st} sub-row corresponds to the experiments using **spline** based target trajectory execution.
- 2^{nd} sub-row corresponds to the experiments using **spline+attract** based target trajectory execution.
- 3^{rd} sub-row corresponds to the experiments using **attract** based target trajectory execution.

The following table provides the results for XY snake robot with 41 DoFs, with respect to various metrics.

Metrics / # Rocks (Instantaneous)	Time Elapsed	Total # Rocks	# Rock Collisions	Rock Avoidance Accuracy (in %)	Total # Targets	# Touched Targets	Target Touch Accuracy (in %)
0	100	0	0	-	16	12	100
			0	-	16	12	100
			0	-	658	654	100
1	100	80	0	100	14	5	50
			0	100	16	12	100
			1	98.75	237	233	100
2	100	160	1	99.375	12	2	25
			0	100	15	10	90.909
			2	98.75	132	127	99.2188
3	100	240	1	99.583	12	1	12.5
			0	100	13	6	66.6667
			2	99.1667	39	28	80
4	100	320	2	99.375	12	1	12.5
			1	99.6875	14	7	70
			5	98.4375	28	18	75
5	100	400	3	99.25	12	0	0
			2	99.5	14	8	80
			8	98	28	19	79.1667

Table 1: Quantitative Results for XY snake with 41 DoFs

The following table provides the results for XY snake robot with 9 DoFs, with respect to various metrics.

Metrics / # Rocks (Instantaneous)	Time Elapsed	Total # Rocks	# Rock Collisions	Rock Avoidance Accuracy (in %)	Total # Targets	# Touched Targets	Target Touch Accuracy (in %)
0	100	0	0	-	16	11	91.667
			0	-	16	12	100
			0	-	187	180	98.361
1	100	80	1	98.75	14	6	60
			2	97.5	16	11	91.667
			2	97.5	144	137	97.857
2	100	160	3	98.125	14	7	70
			1	99.375	16	9	75
			4	97.5	99	93	97.895
3	100	240	4	98.333	12	2	25
			6	97.5	14	8	80
			6	97.5	71	64	95.522
4	100	320	8	97.5	12	1	12.5
			7	97.8125	13	9	100
			15	95.3125	41	31	83.7837
5	100	400	7	98.25	14	4	40
			12	97	13	4	44.444
			13	96.75	39	27	77.1429

Table 2: Quantitative Results for XY snake with 9 DoFs

The following table provides the results for YZ snake robot with 41 DoFs, with respect to various metrics.

Metrics / # Rocks (Instantaneous)	Time Elapsed	Total # Rocks	# Rock Collisions	Rock Avoidance Accuracy (in %)	Total # Targets	# Touched Targets	Target Touch Accuracy (in %)
0	100	0	0	-	16	12	100
			0	-	16	12	100
			0	-	681	677	100
1	100	80	1	98.75	12	2	25
			1	98.75	16	12	100
			1	98.75	258	254	100
2	100	160	2	98.75	12	1	12.5
			4	97.5	15	11	100
			5	96.875	98	93	98.9362
3	100	240	3	98.75	12	0	0
			2	99.1667	12	4	50
			4	98.333	49	42	93.333
4	100	320	7	97.8125	12	0	0
			6	98.125	12	4	50
			3	99.0625	25	17	80.9524
5	100	400	9	97.75	12	0	0
			9	97.75	13	6	66.6667
			12	97	25	15	71.4286

Table 3: Quantitative Results for YZ snake with 41 DoFs

The following table provides the results for YZ snake robot with 9 DoFs, with respect to various metrics.

Metrics / # Rocks (Instantaneous)	Time Elapsed	Total # Rocks	# Rock Collisions	Rock Avoidance Accuracy (in %)	Total # Targets	# Touched Targets	Target Touch Accuracy (in %)
0	100	0	0	-	16	12	100
			0	-	16	12	100
			0	-	652	648	100
1	100	80	0	100	16	9	75
			0	100	16	12	100
			1	98.75	497	493	100
2	100	160	0	100	15	7	63.6363
			3	98.125	16	12	100
			2	98.75	349	343	99.4203
3	100	240	4	98.333	14	7	70
			5	97.9167	15	11	100
			3	98.75	304	300	100
4	100	320	1	99.6875	13	4	44.4444
			3	99.0625	15	10	90.909
			5	98.4375	156	147	96.7105
5	100	400	6	98.5	12	2	25
			5	98.75	16	12	100
			6	98.5	254	249	99.60

Table 4: Quantitative Results for YZ snake with 9 DoFs

2.2 Interesting Inferences

Comparison based on Trajectory Schemes :

Now, let us compare the three different schemes for trajectory execution : **spline**, **spline+attract** and **attract**.

1. By observing the quantitative results mentioned in the previous section and also the plots below, we can say that **spline+attract** scheme works better than the other two, as it almost perfectly trades off between having a good target touch accuracy also relatively low joint velocities.

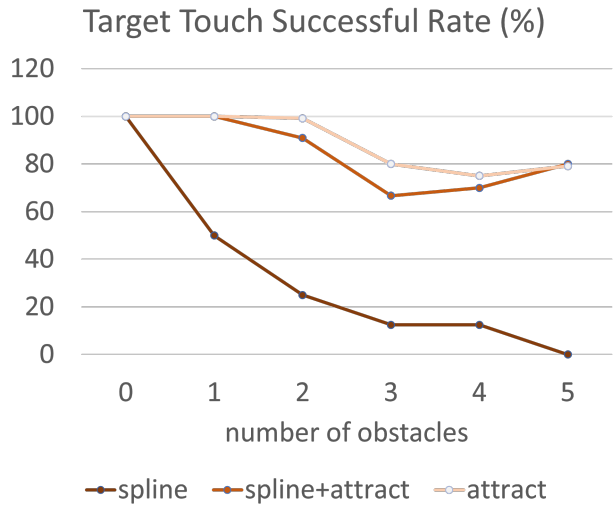


Figure 10: For XY robot with 41 DoFs

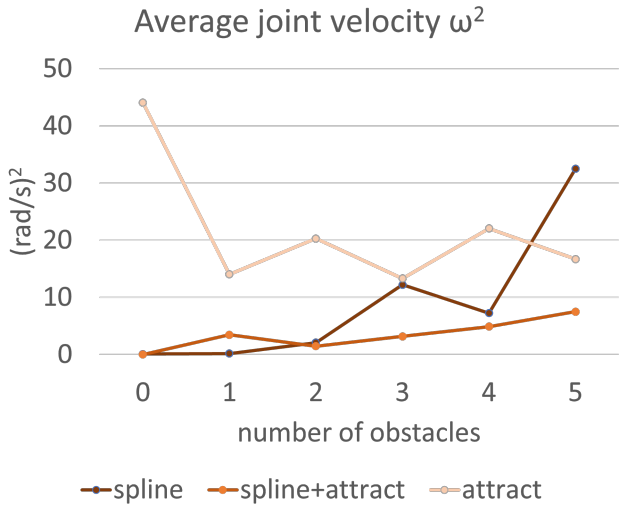


Figure 11: For XY robot with 41 DoFs

2. We see that the **attract** scheme touches the targets really fast but has high joint velocities, which can see in the abrupt joint position changes in Figure 12

3. Also, generally speaking, we observed high joint velocities while untying and lower joint velocities for touching the target, in case of **spline** trajectory scheme.

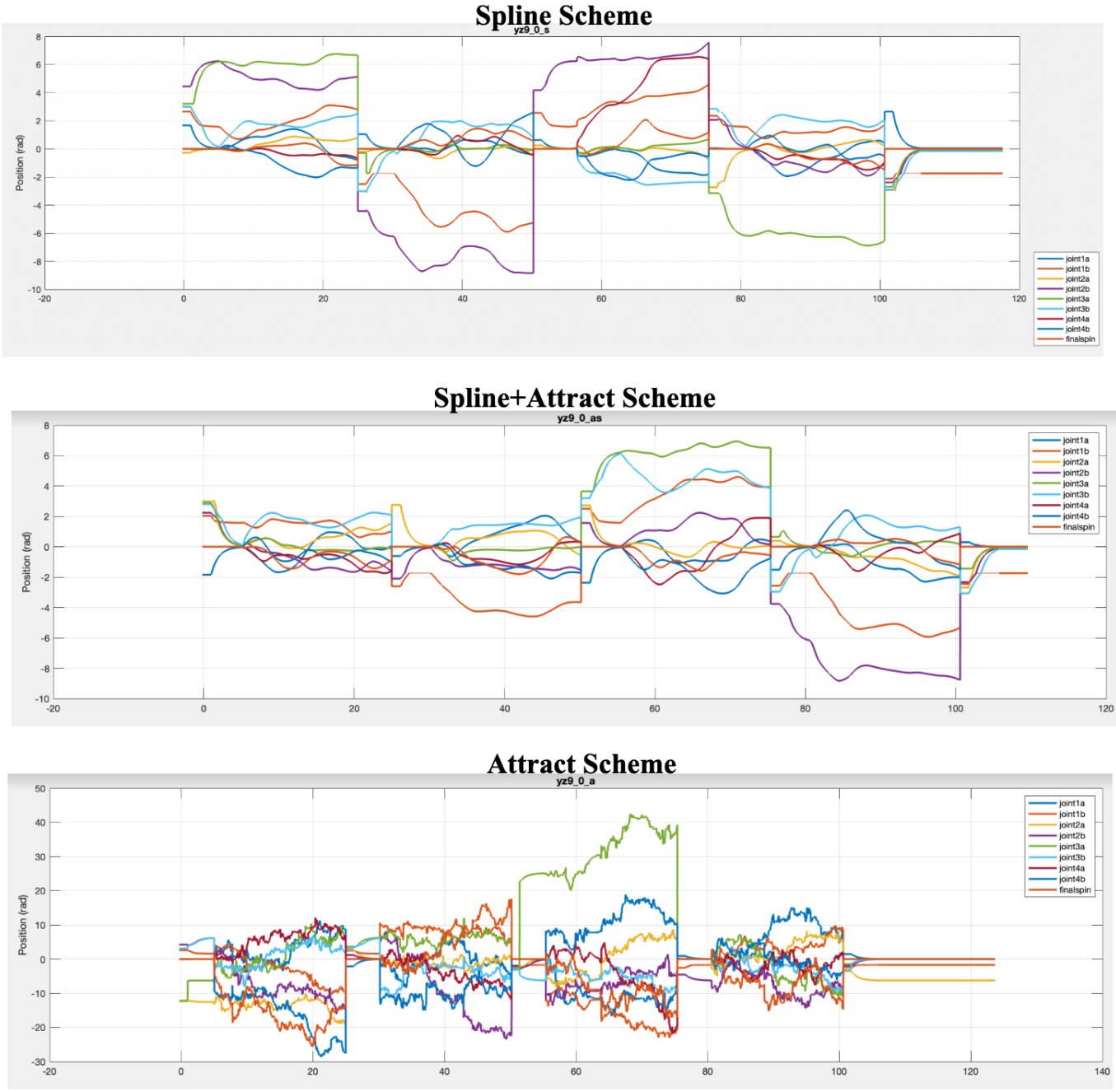


Figure 12: Comparison of Joint Position plots for different schemes - YZ snake with 9 DoFs

Comparison based on # Rocks (Instantaneous) :

In this regard, we have the following observations/inferences:

1. From Figure 13, we can see that the joint plots become more noisy as the number of rocks falling on the robot at a given instant increase. Note the high velocity peaks in the plots occur due to the self collisions that happen during Phase-1 when the robot goes into a knot.

2. From the quantitative results, Figure 10 and Figure 14, we can observe that as the number of rocks falling on the robot at a given instant increase, the target touch accuracy decreases, which is reasonable as we know that touching target is modeled as the secondary task.

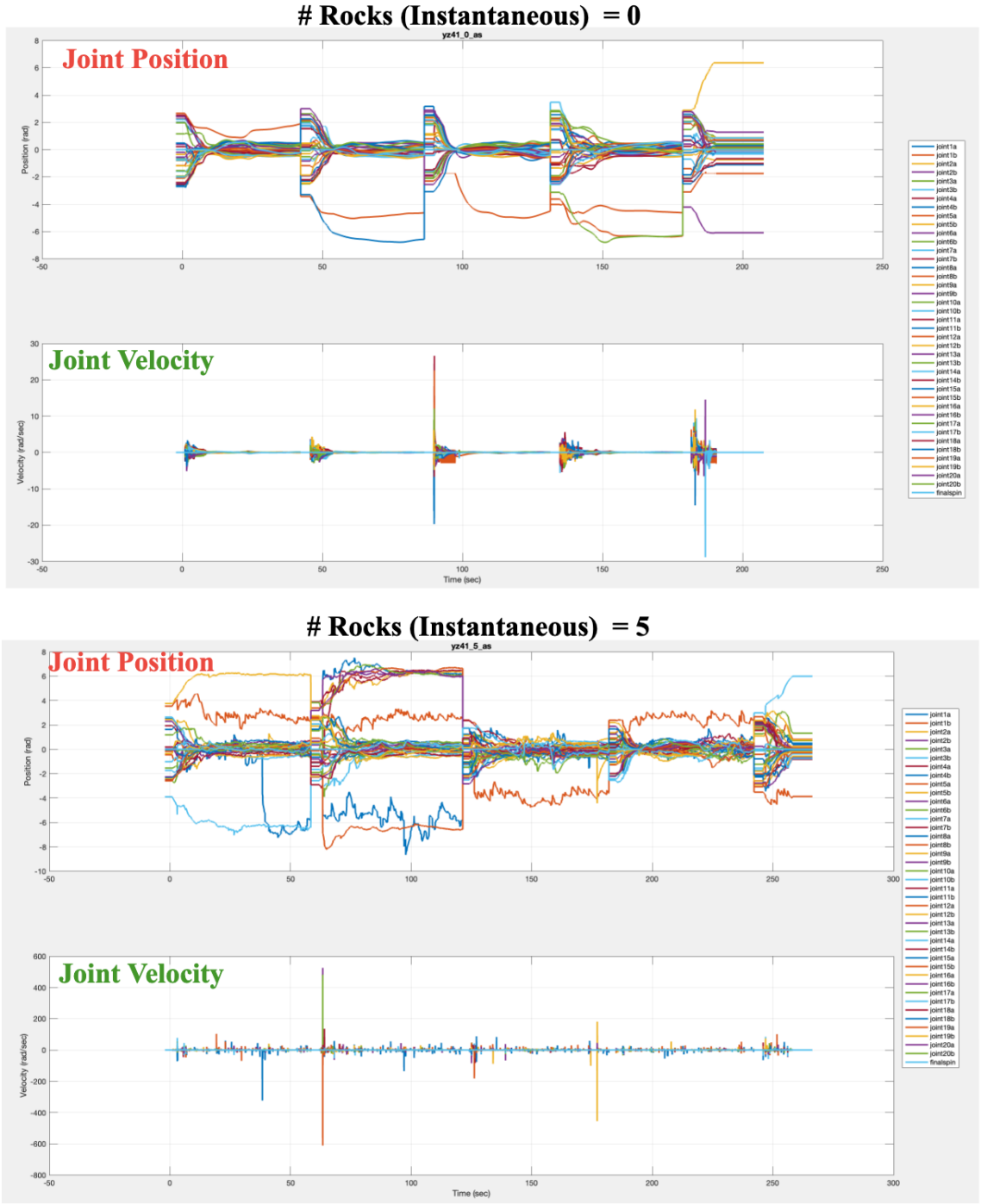


Figure 13: Comparison of Joint (position & velocity) plots for different number of obstacles - YZ Snake with 41 DoFs

3. Also, we can see that in almost all cases the rock avoidance accuracy is extremely high, even as the number of obstacles increase. So this makes sense because in Phase-3, we modeled obstacle avoidance as the primary task.

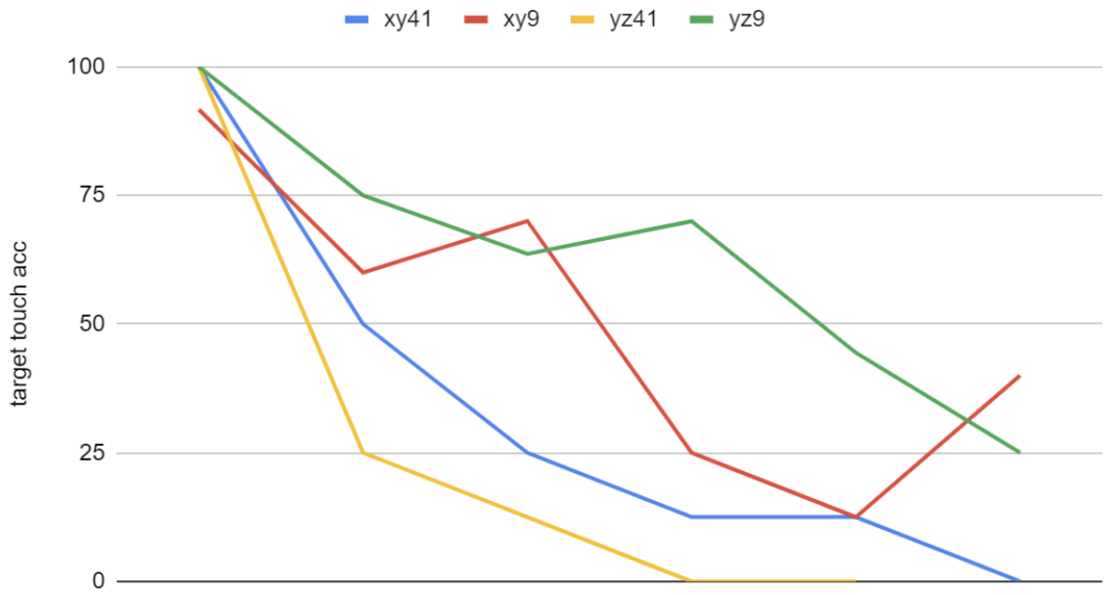


Figure 14: Comparison of Target Touch Accuracies for robot types - **spline** Trajectory Scheme

Comparison based on Robot Types :

For making any generic statements about snake robots with which axis pattern (either XY or YZ) performed better, we feel that a lot more experiments are necessary (as the sample size is too small to generalize as of now). But qualitatively, we observed that in the case of YZ robot there are more self collisions as compared to the XY robot.

Comparison based on Definition of "Tip" Jacobian :

Based on the type of definition of "tip" jacobian (either biological/robotic) jacobian we observe differences in behavior (and hence the names):

- In the case of biological jacobian definition, we consider the "tip" to affect both sides of its body. This leads to a more symmetric and biological movement. But it adds an additional constraint to the robot's movement. We tried using this definition in our experiments but it gave poor results because of this additional constraint.
- In the case of robotic jacobian definition, we consider the tip to affect the joints below it. This leads to asymmetric and more sharp robotic movements. But it has more freedom and hence has a better chance to juggle between multiple tasks more effectively. So, we decided to go with this definition of the jacobian.

A demonstration of this difference in the behaviors is shown in the accompanying project video.

V Conclusion

We successfully built the snake robots and designed their movement to complete multiple tasks simultaneously. The robot can stay in a knot, untie itself while avoiding falling rocks, and touch targets while avoiding rocks. For each task, we define trajectory execution schemes in different ways. To avoid obstacles, the robot moves the joint that is closest to an obstacle away from the obstacle. To touch a target, the robot tip first follows the **spline** trajectory to approach the target, and if the robot hasn't touched the target when the spline is complete, the robot switch to **attract** movement. To untie itself, we define a self repulsive formulation to make the joints move away from the links if they are too close.

We performed experiments and analyses for different robots and scenarios. We noticed that as the number of obstacles increases, the rock avoidance accuracy remains extremely high because we set avoiding obstacles to the primary task, and

meanwhile the target touch accuracy decreases because we set it to the secondary task. We analyzed the velocity and the target touch accuracy for three different schemes for trajectory execution and turns out the **spline+attract** has the best behavior.

We mentioned at the beginning that we want to apply this robot to ambitious applications like Mars space missions and rescue missions. Based on the high obstacle avoidance accuracy of the robot, we are confident that the snake robot can dodge the falling rock in the mission and make great contributions to these applications. Yet, we need to perform more experiments on the robot to fine-tune the target touch accuracy. One other avenue to explore in the future is to change the gamma of the weighted inverse Jacobian and the time interval for each task to see if we can improve the robot's performance.

We would like to acknowledge Professor Günter Niemeyer for the advice on the project topic, the formula of inverse Jacobian and his support in general. We would also like to thank all TAs that gave us feedback and advice.

The code repository for our project can be found at:

`https://github.com/dsriaditya999/robotics_final_v2`

The drive link for the accompanying final video is as follows:

`https://drive.google.com/file/d/1mdCJE-xgja2CTbyLOZJY8AU8m7SZORJ9/view?usp=share_link`