# B.Tech Project (BTP) Report: Control of Continuum Robots Using Deep Reinforcement Learning

Sai Aneesh Suryadevara (190100125), Department of Mechanical Engineering, IIT Bombay

***Abstract*— This report deals with the underlying motivation and methodologies utilised in the successful actuation of a simulated soft robotic arm. Vega FEM C++ middleware library [1] was used to simulate the dynamics of a mass-spring model of a beam under the application of five torques. The amount of torques to be applied were found by formulating the problem as a reinforcement learning task. We have tested the control using Reinforcement Learning in four tasks - Path Following (Family of arcs), Trajectory Following, Trajectory Following with Energy minimization and lastly, Elliptical Trajectory following. The code for the project can be found at : Github Project Link**

***Index Terms* – Reinforcement Learning, Continuum Robots, Control Theory, Soft-Actor Critic (SAC)**

## I. INTRODUCTION

Continuum robots are a class of robots that have flexible and continuous structures, which enable them to perform tasks in constrained and challenging environments. However, the complex kinematics and dynamics of continuum robots make their control a challenging problem. Even when all the information about the environment and the soft robot itself is provided, building an effective model for a particular task is still a substantial challenge. It may involve very complex physical mechanics analysis that can only be established for a specific task. With small changes to the task or environment, the model may be significantly different. To date, there is no principled way for modeling soft robots due to the diversity of material and structure (e.g., pneumatic, hydraulic, cables, electro-active polymers).

To address these challenges, we wish to explore a new approach to the control of soft robots, primarily working on two aspects: designing an abstract representation of the state of soft robots, and developing a reinforcement learning method to derive effective control policies. The reinforcement learning process can be trained quickly by ignoring the specific materials and structural properties of the soft robot.

Reinforcement learning (RL) is a subfield of machine learning that has shown great potential in controlling continuum robots. RL algorithms enable robots to learn control policies through interactions with the environment, without requiring an accurate model of the system's dynamics. Recent research has demonstrated the effectiveness of RL in various tasks, such as obstacle avoidance, path planning, and manipulation.

One of the main advantages of RL-based approaches for controlling continuum robots is their ability to handle the complex and nonlinear behavior of these systems. RL algorithms can learn control policies that optimize task performance without requiring an accurate model of the system's dynamics.

Moreover, RL-based approaches are adaptive and can handle changes in the robot's environment and task requirements. Traditional model-based control methods require an accurate model of the system's dynamics, which may not be possible for complex and uncertain systems such as continuum robots. Furthermore, model-based approaches are often computationally expensive, limiting their use in real-time applications. In contrast, RL-based approaches can learn control policies in a data-driven manner without requiring an accurate model of the system's dynamics.

## II. RELATED WORKS

To study the performance of data-driven control approach of continuum robots using Reinforcement Learning, we take inspiration from several works in this domain. When implementing RL algorithms to robots and other real equipment, an efficient simulator is often needed for training. Previously, researchers mainly choose an ABAQUS subroutine to simulate deformations of the robots. Models from the computer graphics community like discrete differential geometry and Cosserat rod give rise to new simulation methods for soft robots, which are quicker to compute with fair precision and provide possibilities for the implementation of modern AI techniques like RL.

In the work [2] for the actuation of soft magnetic robots, the authors have used the Cosserat rod model for modelling the soft robot and used PyElastica as the simulation environment. They have used the TD3 algorithm for training. In work [3], they have used piece-wise constant curvature (PCC) for modelling the soft robot, used V-REP for the simulation and Soft-Actor Critic as the RL algorithm. In the works [4], [5] the authors have used the MuJoCo simulation environment and used the Soft Actor Critic algorithm for training the soft robot. For our work, we primarily took inspiration from the works [6] and [7] where they have used OpenAI Gym [8] for training the RL policies. This is useful for our work as we can integrate our physics simulator VEGA FEM, with a custom OpenAI Gym environment for RL training.

## III. METHODOLOGY

In this work, the goal is to control the continuum robot and achieve path following and trajectory following tasks. In the earlier work (BTP - I) [9], we have discussed how we created the RL environment by developing a custom OpenAI Gym environment and using ROS framework to integrate it with

the Vega FEM C++ middleware library, which simulates the dynamics of the continuum robot. Now we will discuss how we have implemented Soft Actor Critic algorithm to achieve our tasks.

### A. The Continuum Robot

We chose to utilize the VEGA FEM library to simulate the dynamics of the Mass Spring Model. Vega is a computationally efficient and stable C/C++ physics library for three-dimensional deformable object simulation. It is designed to model large deformations, including geometric and material nonlinearities.

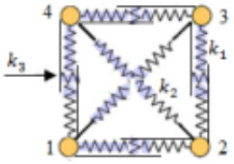



Fig. 1: Mass-Spring Model (MSS)

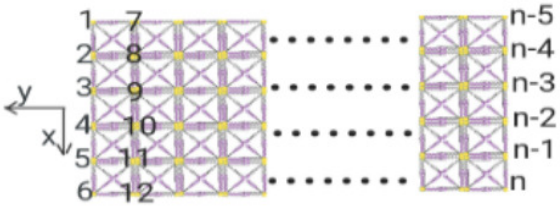



Fig. 2: The soft robot modelled using MSS

The mechanical properties and dimensions of the continuum robot we would using for our experiments is given below:

| | |
|---|---|
| **Number of nodes** | 3 in each row<br>31 columns<br>Total = 93 fixed vertices |
| **Number of Edges** | 360 |
| **Dimesions of the Beam** | Width = 0.05 m(x-direction)<br>Length = 0.75 m(y -direction) |
| **Surface Density** | 60 |
| **Tensile Stiffness** | 18750 N/m |
| **Shear Stiffness** | 18750 N/m |
| **Bending Stiffness** | 0 N/m |
| **Application of Torques** | Between Nodes 16-18, 34-36, 52-54, 70-72 and 88-90 |

### B. Defining the MDP

A Markov decision process (MDP) refers to a stochastic decision-making process that uses a mathematical framework to model the decision-making of a dynamic system. It is used in scenarios where the results are either random or controlled by a decision maker, which makes sequential decisions over time. MDPs evaluate which actions the decision maker should take considering the current state and environment of the system. We formulate RL problems as a Markov Decision Process (MDP). In our case,

**MDP**: $(S, A, T, R, \gamma)$

State Space: $S = [x_r, y_r, x_t, y_t]$

Continuous space in the range (-1, 1)

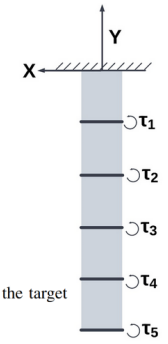
Action Space: $A = [\tau_1, \tau_2, \tau_3, \tau_4, \tau_5]$

Continuous space (-1, 1) for each of the five torques

Transition Probabilities: Handled by the VEGA simulation

Reward function: $R = -20 * |\sqrt{(x_r - x_t)^2 + (y_r - y_t)^2}|$

Discount Factor: 0.9 $(x_r, y_r)$ is the tip of the robot, and $(x_t, y_t)$ is the target




### C. Deep Reinforcement Learning

Firstly, we realize that our observation space (the position of the soft robot) and the action space (the torques to be applied) are continuous (not discrete). So, in this case we have three well known algorithms - Twin Delayed DDPG (TD3), Proximal Policy Optimization (PPO) and Soft Actor Critic. After our literature survey and trying out PPO and TD3, we have proceeded with the SAC algorithm as it gives the best results for this problem setting.

### D. Soft Actor Critic

Soft Actor Critic (SAC) Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. A key feature of SAC, and a major difference with common RL algorithms, is that it is trained to maximize a trade-off between expected return and entropy, a measure of randomness in the policy.

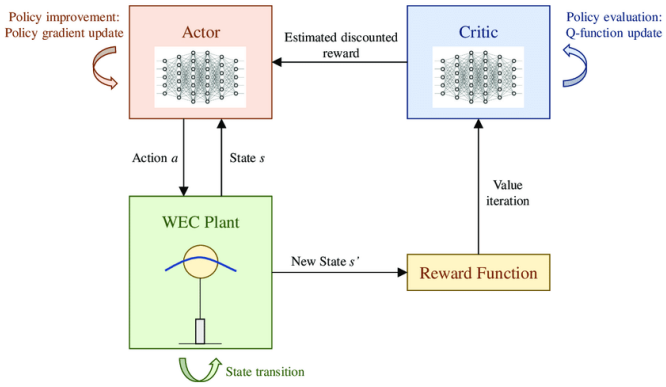



Fig. 3: Here we use the MlpPolicy which implements the Actor-Critic neural networks with 2 layers of 64 neurons.

### E. Hyperparamter Tuning

**1. Learning Rate:** corresponds to the strength of each gradient descent update step. This should typically be decreased if training is unstable, and the reward does not consistently increase. We have varied the learning rate within these values $[1e-3, 3e-4, 1e-5, 1e-6]$. The default value is $0.0003$ Here, I have compared only upto 70K timesteps because some of the training went unstable afterwards. We see that

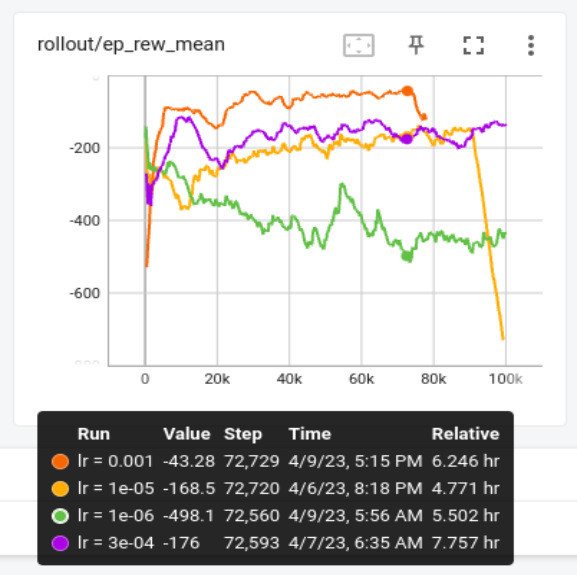



Fig. 4: Best result with learning rate = 0.001

the lowest lr = 1e-06 is not able to learn. As we keep increasing the learning rate, we find the best performance with lr = 0.001. This is the most important hyperparameter to be tuned as it significantly affects the performance.

**2. Tau:** corresponds to the magnitude of the target Q update during the SAC model update. In SAC, there are two neural networks: the target and the policy. The target network is used to bootstrap the policy's estimate of the future rewards at a given state, and is fixed while the policy is being updated. This target is then slowly updated according to tau. Typically, this value should be left at 0.005. For simple problems, increasing tau to 0.01 might reduce the time it takes to learn, at the cost of stability. We have varied tau in these values $[0.01, 0.005, 0.001]$


Fig. 5: Best result with tau = 0.01

**3. Gamma:** The discount factor is a hyperparameter that controls how much the agent values future rewards over immediate ones. It is usually denoted by gamma, and ranges from 0 to 1. A low gamma means the agent is short-sighted and only cares about the current reward, while a high gamma means the agent is far-sighted and considers the long-term consequences of its actions. For actor-critic algorithms, the discount factor affects the critic network, as it is used to calculate the expected return. We have varied the gamma between $[0.9, 0.95, 0.99]$


Fig. 6: Best result with gamma = 0.9

**4. Batch Size:** is the number of experiences used for one iteration of a gradient descent update. The default value is 64. However, when we are working with continuous action spaces, it is advisable to go with larger values, typically in the range $[256, 512, 1024]$.

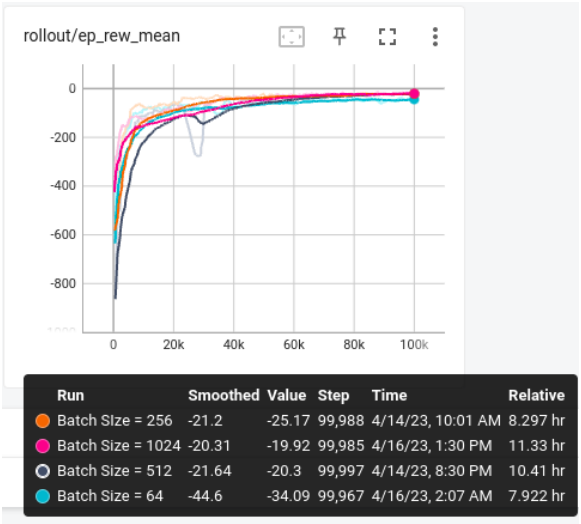



Fig. 7: Best result with batch size = 512

**Tuned Hyperparamters**

| Hyperparameters | Optimal Value |
|---|---|
| Learning Rate | 0.001 |
| Tau | 0.01 |
| Gamma | 0.9 |
| Batch Size | 512 |
| generalized State Dependent Exploration | True |

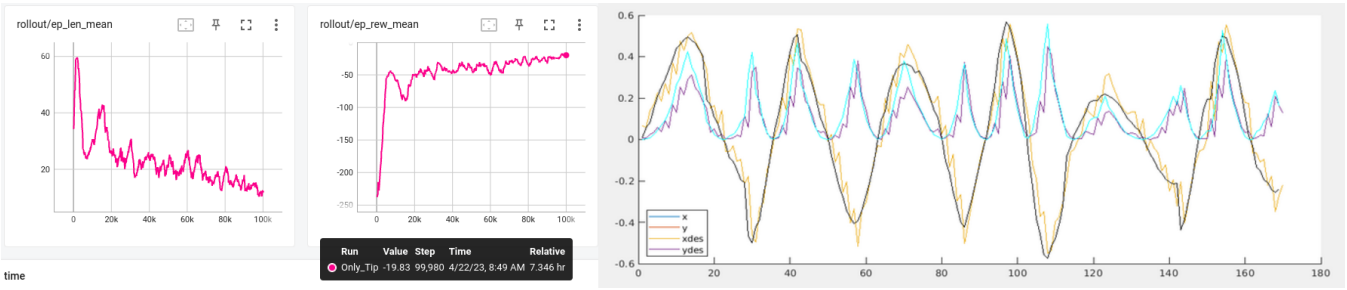



Fig. 8: **MODEL 1 :** a) Mean Episode length (**11.8 timesteps**) and Mean Episode reward (**-19.83**) for Task 1 with only tip as state information. b) We can see the performance of the model, it is following the arcs **even though the arc radius is changing**. However, there is a lot of jitter.

## IV. Task 1: Path Following Task

For the first task, we aim follow an arc of angular width $160°$ with the radius ranging between $[0.2, 0.6]$. The Target Angular velocity is fixed at $0.5\, rad/sec$. **Note:** The direction of the target can change suddenly during the arc motion, but the agent will still follow the target.

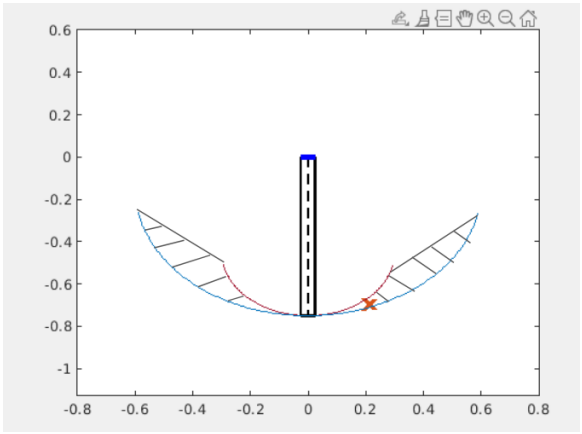


Fig. 9: The shaded region indicates the robot workspace

### A. State has only tip of the robot

We begin with the most simple case. The state information only contains the coordinates of the tip of the robot and reward is the euclidean distance between the tip and the target.

$$S = [x_r, x_{tar}]_{(4\times1)} \qquad R = w_d r_d(S)$$

$$x_r = [x_{tip}, y_{tip}]_{(2\times1)} \qquad r_d(S) = |\sqrt{(x_{tip} - x_t)^2 + (y_{tip} - y_t)^2}|$$

$$x_{tar} = [x_t, y_t]_{(2\times1)} \qquad w_d = -20$$

### B. Limiting the action space (the torques applied) to avoid jitter while following the path

Here, in the Task 1 result, we see a lot of jitter while the robot is trying to follow the path. So we will try to limit the torques and test whether it will improve performance and reduce the jitter.

As we know the action space is $A = [\tau_1, \tau_2, \tau_3, \tau_4, \tau_5]$, where each $\tau_i \in [-1, 1]$. However, the actual torques sent to the VEGA FEM, for calculating the deformations of the soft robot, are scaled to $[-4.5, 4.5]$. So, here we verify if this is the optimal range of torques for the task or if we can achieve a smoother performance with lower torque values. We have compared the performance with torque in the ranges: $[-0.5, 0.5], [-1.5, 1.5], [-3, 3], [-4.5, 4, 5], [-6, 6]$.

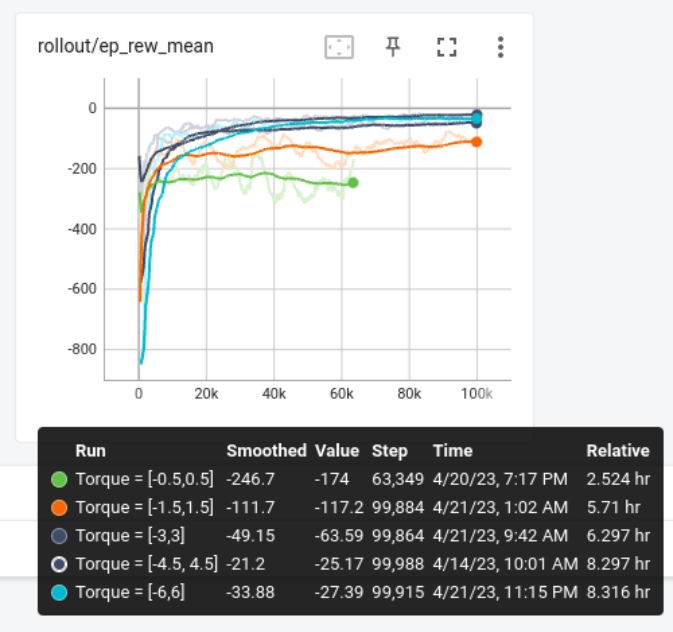


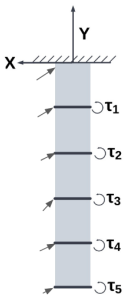

Fig. 10: The best performance is with $\tau = [-4.5, 4.5]$

### C. Complete State information of 6 points on the robot

$$S = [x_r, x_{tar}]_{(14\times1)} \qquad R = w_d r_d(S)$$

$$x_r = [x_1, y_1, ..., x_{tip}, y_{tip}]_{(12\times1)} \qquad r_d(S) = |\sqrt{(x_{tip} - x_t)^2 + (y_{tip} - y_t)^2}|$$

$$x_{tar} = [x_t, y_t]_{(2\times1)} \qquad w_d = -20$$



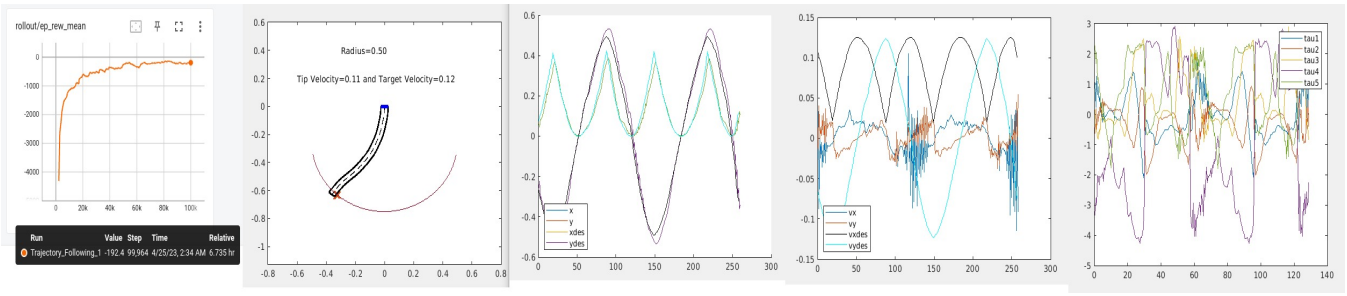



Fig. 11: **MODEL 3**, with (**case 2**) reward function:(a) Mean Episode reward (**-192.4**) for Task 2 with **six point state** information. (b) The Matlab visualisation of the soft robot (c) Following an arc of radius = 0.5 with velocity = 0.12 (**fixed radius** and **fixed velocity**) (d)The velocity $v_x$ and $v_y$ plot (e)The torques being applied

The state information contains the coordinates of 6 equidistant points on the robot and these points are the points of application of torque. This ensures that the agent has more precise information of how the soft robot actually deforms with the application of torques.

| Model | Mean Reward (2000 timesteps) |
|---|---|
| **Model 1:** Only Tip as state | -1.60 |
| **Model 2:** 6 point state information | -1.52 |

**Conclusion:** Model 2 performs better as expected, and it has higher mean reward, which implies it has better performance in path following.

## V. Task 2: Trajectory Following

For the second task, we aim follow an arc of angular width 160° with the radius ranging between $[0.2, 0.6]$. However, now the target Angular velocity is varying between $[0.1, 0.5]$ $rad/sec$. Hence the MDP will now change as we include the velocity term in the reward function. **Note:** Here also, the direction of the target can change suddenly during the arc motion, but the agent will still follow the target.

$$S = [x_r, v_r, x_{tar}, v_{tar}]_{(18\times1)}$$

$$x_r = [x_1, y_1, ..., x_{tip}, y_{tip}]_{(12\times1)}$$

$$v_r = [v_{x,tip}, v_{y,tip}]_{(2\times1)}$$

$$x_{tar} = [x_t, y_t]_{(2\times1)}$$

$$v_{tar} = [v_{x,tar}, v_{y,tar}]_{(2\times1)}$$

$$R = w_d r_d(S) + w_v r_v(S)$$

$$r_d(S) = |\sqrt{(x_{tip} - x_t)^2 + (y_{tip} - y_t)^2}|$$

$$r_v(S) = |\sqrt{(v_{x,tip} - v_{x,tar})^2 + (v_{y,tip} - v_{y,tar})^2}|$$

$$Case\ 1 : w_d = -20, w_v = -10$$
$$Case\ 2 : w_d = -40, w_v = -10$$
$$Case\ 3 : w_d = -400, w_v = -200$$

### A. Reward Function Design

Now, since we have two objectives, firstly to minimize the error between the tip of the robot and the target, and secondly to follow the path with any given velocity. (minimize the difference between the velocity of the robot tip and the velocity of the target). We now test three cases, where we first compare the performance with respect to the importance of each term in the reward function. (Case 1 and Case 2). Next we look at the effect of scaling the reward term. (Case 2 and Case 3). Hence, there are three cases of **Model 3**.

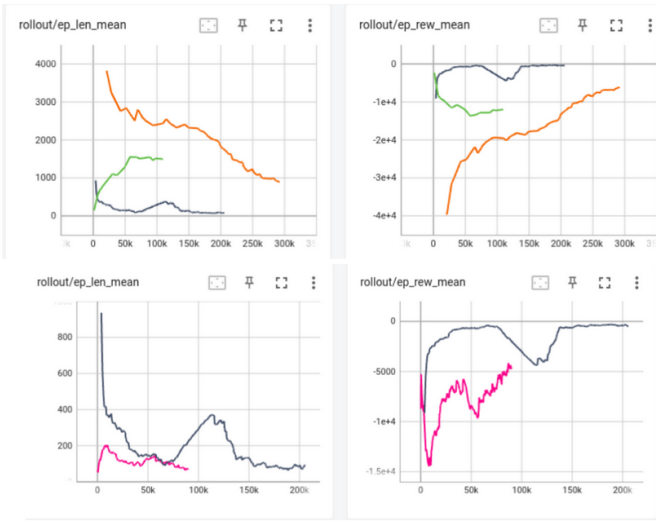



Fig. 12: **a) Case 1**: Green is training from scratch. (100k timesteps, 5 hours, Reward -12,000) Orange is training from pretrained model. (300k timesteps, 15 hours, Reward -6000) **b) Case 2:** Blue shows case 2, training from scratch. (200k timesteps, 15 hours, Reward -400) **c) Case 3:** Pink shows model 3, training from case 2 initialization. (90k timesteps, 6 hours, Reward -4760)

| 3000 Timesteps | Case 1 | Case 2 | Case 3 |
|---|---|---|---|
| **Mean Distance Error** | 0.078 | 0.075 | 0.08 |
| **Mean Velocity Error** | 0.12 | 0.11 | 0.13 |

Fig. 13: The best performance is with **Case 2**. Hence, this is best handcrafted reward function for this task.

### B. Effect of the velocity term in the reward

Though, we had expected the velocity term to be useful in the trajectory following, the soft robot was not following the desired velocity very closely. It has oscillations where it peaks to the max value and settles down again. However, it had an interesting effect on the path following. We can see that the robot now follows the path with less jitter. This can be explained as the velocity term penalizes the robot for sudden and erratic movements.

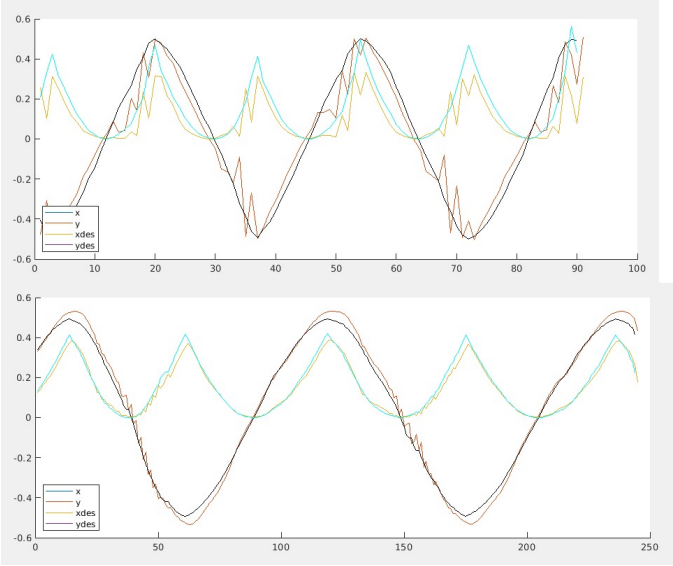



Fig. 14: Following an arc of constant radius $r = 0.5$ **Above:** This is the path following output from **MODEL 2**. **Below:** We can see the improved performance of **MODEL 3** and smooth tracking when we have added velocity term in the reward.

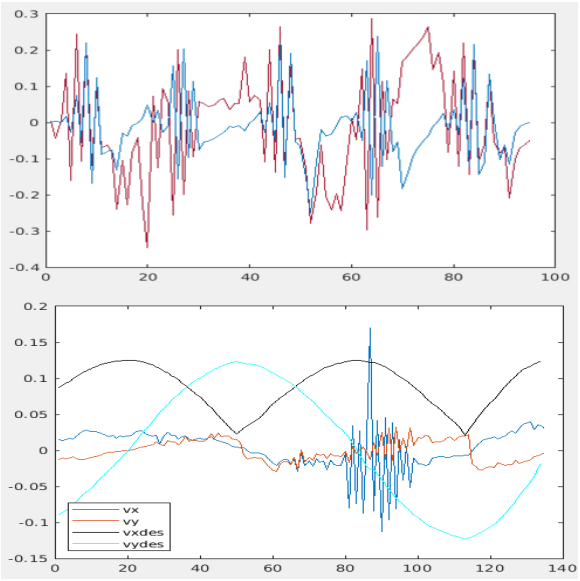



Fig. 15: Comparing the Velocity terms $v_x, v_y$ for both models **Above: MODEL 2**, We can see the high values of $v_x, v_y$ in the range [-0.3, 0.3] **Below: MODEL 3** Here the actual velocities $v_x, v_y$ are smaller, in the range [-0.05, 0.05]

Hence, as explained above, adding the velocity term in the reward function helped to regulate the high velocities and thus ensured a better path following task performance.

| | **MODEL 2 (Fixed radius = 0.5)** | **MODEL 2 (radius ∈ [0.2, 0.6])** |
|---|---|---|
| **Distance Error** | 0.075 | 0.088 |
| | **MODEL 3 (Fixed radius = 0.5)** | **MODEL 3 (radius ∈ [0.2, 0.6])** |
| **Distance Error** | 0.045 | 0.064 |

## Important Conclusions:

- Firstly, by randomly changing the direction of the target from clockwise to counter-clockwise direction (or vice versa) along the arc, at random intervals during the training, improved the performance
- The trained models can follow an arc of any radius $r \in [0.2, 0.6]$ which is important for generalization capabilities
- The velocity term in the reward helped perform path following more smoothly. However, it failed to move along the path with the desired velocity.

## VI. Task 3: Trajectory Following with Energy Minimization

We know that a continuum robot can reach a specific point in multiple ways, however we wish to reach the point with minimum work done. Considering the work done by the robot to reach a point as $|\sum_{i=1}^{5} \tau_i \theta_i|$, we wish to minimize this term. Hence we modify the reward function as shown below:

$$S = [x_r, v_r, x_{tar}, v_{tar}]_{(18\times1)}$$

$$x_r = [x_1, y_1, ..., x_{tip}, y_{tip}]_{(12\times1)}$$

$$v_r = [v_{x,tip}, v_{y,tip}]_{(2\times1)}$$

$$x_{tar} = [x_t, y_t]_{(2\times1)}$$

$$v_{tar} = [v_{x,tar}, v_{y,tar}]_{(2\times1)}$$

$$R = w_d r_d(S) + w_v r_v(S) + w_E r_E(S)$$

$$r_d(S) = |\sqrt{(x_{tip} - x_t)^2 + (y_{tip} - y_t)^2}|$$

$$r_v(S) = |\sqrt{(v_{x,tip} - v_{x,tar})^2 + (v_{y,tip} - v_{y,tar})^2}|$$

$$r_E(S) = |\tau_1 * \theta_1 + ... + \tau_5 * \theta_5|$$

$$w_d = -40, w_v = -10, w_E = -1$$

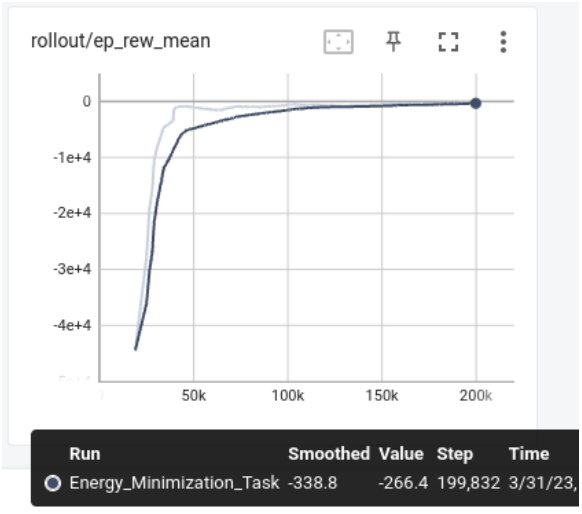



Fig. 16: RL Training plot, Mean Reward of **-226.4** after training time of **13** hours

We can see the results of the model below. While, we can see that the energy term has reduced, but now the path following is not as good as before (Model 3). This can be attributed to the complex reward function we have for this model. Since it has mutliple terms, it is a very tedious task to appropriately scale the three terms to achieve optimal performance as we expect.

| | Energy Minimization Task | Trajectory Following (Model 3) |
|---|---|---|
| **Mean Reward** | -4.41 | -3.08 |
| **Mean Distance Error** | 0.053 | 0.045 |
| **Mean Velocity Error** | 0.127 | 0.128 |
| **Mean Energy Term** | 1.02 | 1.55 |

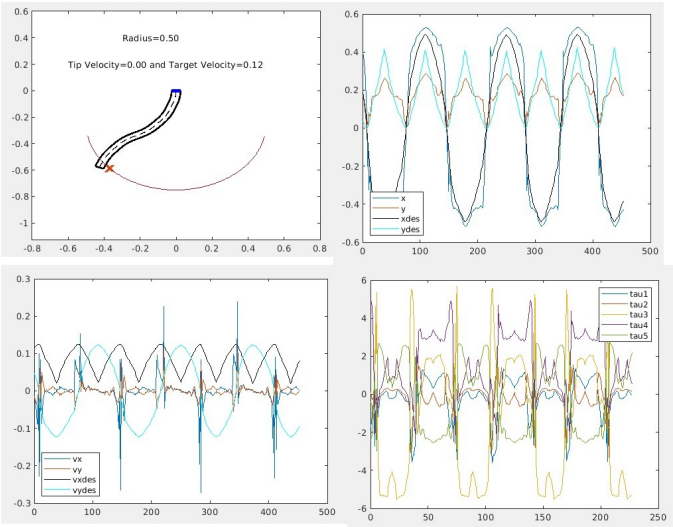



Fig. 17: **(a)** The Matlab visualization showing the trajectory following task with energy minimization **(b)** The path following plot **(c)** Velocity tracking plot **(d)** The torques applied

## VII. TASK 4: FOLLOWING AN ELLIPTICAL TRAJECTORY

After exploring the task of following a family of arcs, we move to a more challenging problem of following a more complicated trajectory - an ellipse. The equations of the elliptical trajectory are given below:

$$x_{tar} = asin(\theta) \qquad v_{x,tar} = a\omega cos(\theta)$$
$$y_{tar} = a\sqrt{1-e^2}(1-cos(\theta)) \qquad v_{y,tar} = a\omega\sqrt{1-e^2}sin(\theta))$$

| | |
|---|---|
| **Semi Major Axis Range** | 0.18 - 0.25 |
| **Eccentricity (e)** | 0.5- 0.8 |
| **Target Angular Velocity (ω)** | 0.5 - 1.0 |

Fig. 18: Parameters for training the agent

### A. Effect of scaling the torques

Initially, the model didn't work well when the torques were in the range $[-4.5, 4.5]$. Hence for this task, I scaled the torques to the range $[-8, 8]$. If we look at the performance

$$S = [x_r, v_r, x_{tar}, v_{tar}]_{(18\times1)} \qquad R = w_d r_d(S) + w_v r_v(S)$$
$$x_r = [x_1, y_1, ..., x_{tip}, y_{tip}]_{(12\times1)} \qquad r_d(S) = |\sqrt{(x_{tip}-x_t)^2 + (y_{tip}-y_t)^2}|$$
$$v_r = [v_{x,tip}, v_{y,tip}]_{(2\times1)} \qquad r_v(S) = |\sqrt{(v_{x,tip}-v_{x,tar})^2 + (v_{y,tip}-v_{y,tar})^2}|$$
$$x_{tar} = [x_t, y_t]_{(2\times1)} \qquad w_d = -40, w_v = -10$$
$$v_{tar} = [v_{x,tar}, v_{y,tar}]_{(2\times1)}$$

Fig. 19: State and Reward for training is same as **MODEL 3**

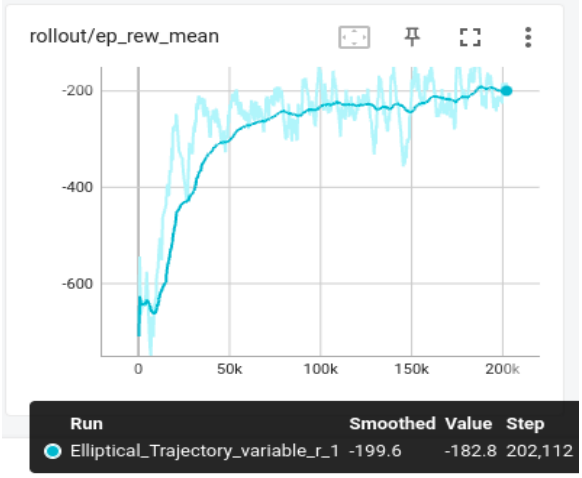



Fig. 20: RL Training plot, Mean reward of **-182.8** after training time of **23 hours**

of the model, we can see that follows the elliptical trajectory pretty well.

### B. Curriculum Learning for this task

Since this was more challenging task than earlier case of arc following, directly training the model on variable parameters: semi-major axis, eccentricity and angular velocity resulted in poor performance. Hence, we implemented the curriculum learning approach where we increased the lesson difficulty at each level. First after training the model on fixed parameters, we slowly increase the parameter range so that the agent learns the task progressively.

### C. Reducing the target velocity

To enable more accurate path following of the elliptical trajectory, we tried to implement the RL training at a lower target angular velocity of $0.05$ rad/sec. (Almost $10-20$ times slower than earlier case). However, the soft robot still fails at the upper portion of the elliptical trajectory.

| | Elliptical Path Following |
|---|---|
| **Mean Reward** | -3.00 |
| **Mean Distance Error** | 0.0475 |
| **Mean Velocity Error** | 0.109 |

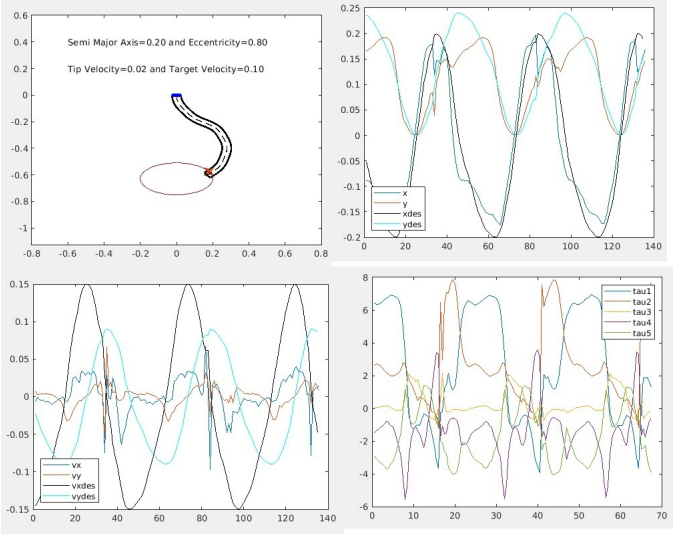



Fig. 21: **(a)** The Matlab visualization showing the ellipse following task **(b)** The path following plot **(c)** Velocity tracking was not efficient **(d)** The torques applied

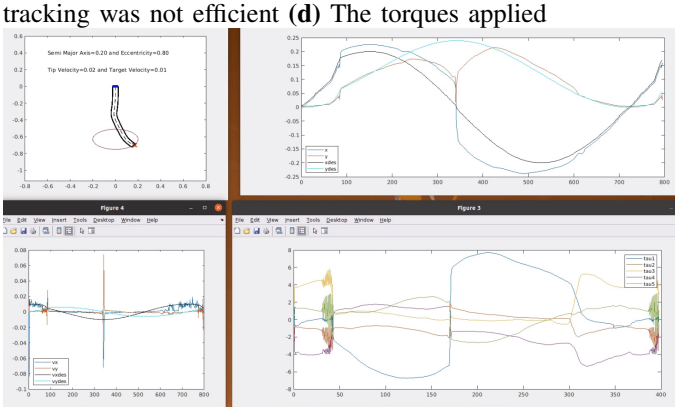


Fig. 22: **The Case with slower target angular velocity. (a)** The Matlab visualization showing the ellipse following task **(b)** The path following plot. I t particularly fails to follow the path in the top section of the ellipse **(c)** Velocity tracking **(d)** The torques applied

## VIII. Comparison with Existing Published works

The problem with the existing literature is that, the only measure of performance most papers have provided is the **Mean Episodic reward**. We have also provided this metric in our work, however this makes it harder for directly comparing the results of one paper with another because the reward functions could be different.

### *A. **Paper 1:** Elastica: A compliant mechanics environment for soft robotic control [7]*

In this work, the task is to achieve continuous 3D tracking of a randomly moving target. The State, action, reward framework of the agent is as follows.

$$\text{State} = [x_a, x_t, v_a, v_t]$$

Coordinates of robot tip, coordinates of target, velocity of robot tip, velocity of target

$$\text{Action} = [\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6]$$

The applied torque in each direction controlled by 6 equidistantly spaced control points leading to a total of 12 degrees of freedom.

$$R = -n^2 + g(S) + \begin{cases} 0.5 & d < n < 2d \\ 2.0 & n < d \end{cases}$$

where is $n$ the square of the L2 norm distance between the tip of the arm and the target $n = ||x_a - x_t||_2$ and $d$ is a defined bonus range distance (here $d = 5$ cm). Additionally, because the allowed action space was capable of causing the simulation to become unstable, a penalty term of $-1000$ was added any time a NaN was detected in the state information, which indicated the simulation had become unstable. Detection of a NaN would cause the episode to end.

The best performance was achieved by using the Soft-Actor Critic (SAC) Algorithm.

### *B. **Paper 2:** Model-Free Reinforcement Learning with Ensemble for a Soft Continuum Robot Arm [5]*

The authors proposed a system that could learn control policies for a continuum robot arm to reach target positions using its tip not only in simulations but also in the real world. They used a pneumatically controlled continuum robot arm that operates with nine flexible rubber artificial muscles. Each artificial muscle can be controlled independently by pressure control valves, demonstrating that the policy can be learned using a real robot alone.

The state, action and reward are defined as follows:

| Item | Dimensions | Descriptions |
|---|---|---|
| State | | |
| Position of each point | 3 | $(x, y, z)$ [m] |
| Velocity of each point | 3 | $(v_x, v_y, v_z)$ [m/s] |
| Pressure of each actuator | 9 | $p_s$ [MPa] |
| Time derivative of the pressure | 9 | $\dot{p}_s$[MPa/s] |
| Action | | |
| Pressure of each valve | 9 | $p_a$ [MPa] |

Note that the positions of parts other than the tip were ignored. In the learning phase, the target position was randomly determined for each episode within a prede- termined reachable range covering a cylinder with a height of 30 cm and a radius of 30 cm. Each episode consisted of 300 steps. The states are listed in Table above, added to the three-dimensional difference between the target position and the continuous arm tip position. The reward function was given as

$$r(s, a) = -||\text{tip position} - \text{target position}||_2$$

The reward function did not include a penalty term that was dependent on the magnitude of the action, that is, the magnitude of air pressure in a real robot. As mentioned in the introduction, there is a lower need for penalty terms in soft robots than in hard robots, and penalty terms also impede learning; thus, they were not utilized in this study.

In this paper, the authors proposed an Ensembled Light- weight model-Free reinforcement learning Network (ELFNet) but the baseline they have compared with is again the Soft-Actor Critic (SAC) Algorithm.

### C. ***Paper 3:*** *Position Control of Cable-Driven Robotic Soft Arm Based on Deep Reinforcement Learning [10]*

In this paper, the authors combine the data-driven modeling method with the reinforcement learning control method to realize the position control task of robotic soft arm, the method of control strategy based on deep Q learning.

**State** : the state space of the system consists of the following three parts: the first part is the current soft arm shape parameter, which is composed of six coordinates' information; the second part is the current state of the steering gear driven by the soft arm, which consists of four servo parameters; the third part is the target position of the target soft arm, which is composed of two coordinates' information.

**Action** : In the soft arm problem, the original motion space is high-dimensionally continuous, such as the servo control frequency of the drive cable, and for the characteristics of the deep Q learning algorithm, the original motion space needs to be discretized. The soft robot arm is driven by four servos, and each servo can choose two actions: tighten or relax.

**Reward** :

If the distance between tip of the robot and target is within some bound $\epsilon$, then we terminate the episode and move to new one. The reward is given by

$$\text{If} \quad ||X_c - X_t|| < \epsilon \quad R = 2$$

Elseif the distance between current tip position and target is smaller than last tip pose and target, the episode is not terminated and the reward is given by

$$||X_c - X_t|| < ||X_l - X_t||, \quad \text{then} \quad R = 1$$

Else, $R = 0$

In this paper, the authors adopt Deep Q-learning for training the RL model.

### D. ***Paper 4:*** *Efficient reinforcement learning control for continuum robots based on Inexplicit Prior Knowledge [11]*

In this work also we look at the path following task where the task is to achieve continuous 2D tracking of a randomly moving target. The State, action, reward framework of the agent is as follows.

State = [Coordinates of target w.r.t to the robot tip, Accumulated actions]

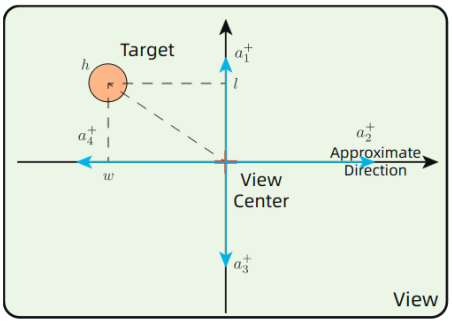



Fig. 23: The origin is the tip of the robot and the state information contains the distance of the target from the tip

Action = [Actions are the control values of four motors, each is between (0, 1)]

The reward function $r$ is essentially the raw reward is the amount of state change between current state and next state. There are also some special scenes need to amend the raw reward which are listed below:

$$R = \begin{cases} r - 10 & \text{any}(s') \text{ is None} \\ r - 10 & \text{out of view scope} \\ r + 10 & |s_0'| < \epsilon \text{ and } |s_1'| < \epsilon \\ r + 100 & \text{finish the tracking route} \end{cases}$$

In this paper, the authors adopt the soft actor-critic (SAC) as thier policy gradient algorithm and Model-Based Policy Optimization (MBPO) as their MBRL algorithm.

## IX. CONCLUSION

In this work, we explore data-driven approaches for control of a continuum robot using Reinforcement Learning. We have started with the task of training the soft robot to follow an arc of any radius. An important takeaway here is that, we have to train the robot to move in both clockwise and anti-clockwise directions and this transition must be made at random time intervals. This significantly improves the performance of the model. Overall, using the trained model the robot was able to follow the path but there was a lot of jitter. Then we looked at how increasing the state space improves the performance by a little margin. However, the best performance was achieved only after the reward term was modified to include the velocity error term. This acted as regularization term which prevented high velocities of the robot and improved the path following performance. Then, we looked at improving this performance by including an energy minimization term. However, the reward function was too complex with three terms that the model was not able to learn the path following task so well. Lastly, we looked the trajectory following of a more complicated path - an ellipse. The training was successful in this task as well, however there is some scope of improvement for this task as the robot deviates from the path during some portion of the upper half of the ellipse.

For future works, since we have already shown the working of RL control for this task in simulation, we could

test the performance on the real robot. Here, we can build a curriculum learning framework, where we increase the randomization of the properties of the soft robot (Tensile and Bending stiffness etc.) after each lesson. This is essential as the real world robot can have some variations in its properties (over time) and using the curriculum learning framework, we can train the model to achieve optimal performance in the real world as well.

## REFERENCES


[1] J. Barbič, F. S. Sin, and D. Schroeder, "Vega FEM Library," 2012, http://www.jernejbarbic.com/vega.

[2] J. Yao, Q. Cao, Y. Ju, Y. Sun, R. Liu, X. Han, and L. Li, "Adaptive actuation of magnetic soft robots using deep reinforcement learning," 2022.

[3] C. Yang, J. Yang, X. Wang, and B. Liang, "Control of space flexible manipulator using soft actor-critic and random network distillation," *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2019, pp. 3019–3024.

[4] G. Li, J. Shintake, and M. Hayashibe, "Deep reinforcement learning framework for underwater locomotion of soft robot," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 12 033–12 039.

[5] R. Morimoto, S. Nishikawa, R. Niiyama, and Y. Kuniyoshi, "Model-free reinforcement learning with ensemble for a soft continuum robot arm," *2021 IEEE 4th International Conference on Soft Robotics (RoboSoft)*, 2021, pp. 141–148.

[6] M. A. Graule, T. P. McCarthy, C. B. Teeple, J. Werfel, and R. J. Wood, "Somogym: A toolkit for developing and evaluating controllers and reinforcement learning algorithms for soft robots," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4071–4078, 2022.

[7] N. Naughton, J. Sun, A. Tekinalp, T. Parthasarathy, G. Chowdhary, and M. Gazzola, "Elastica: A compliant mechanics environment for soft robotic control," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3389–3396, 2021.

[8] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[9] S. A. Suryadevara and A. Shah, "B.Tech Project Phase-I Report, Control of Soft Robots Using Deep Reinforcement Learning," 2022.

[10] Q. Wu, Y. Gu, Y. Li, B. Zhang, S. A. Chepinskiy, J. Wang, A. A. Zhilenkov, A. Y. Krasnov, and S. G. Chernyi, "Position control of cable-driven robotic soft arm based on deep reinforcement learning," *Inf.*, vol. 11, p. 310, 2020.

[11] J. Liu, J. Shou, Z. Fu, H. Zhou, R. Xie, J. Zhang, J. Fei, and Y. Zhao, "Efficient reinforcement learning control for continuum robots based on inexplicit prior knowledge," *CoRR*, vol. abs/2002.11573, 2020. [Online]. Available: https://arxiv.org/abs/2002.11573


## ACKNOWLEDGMENT


I acknowledge the support of Prof. Abhishek Gupta, Prof. Shivaram Kalyanakrishnan and Mr. Shubham Agrawal, and I wholeheartedly thank them for their guidance and fruitful discussions.