

SoftManiSim: A Fast Simulation Framework for Multi-Segment Continuum Manipulators Tailored for Robot Learning (Supplementary Material)

Anonymous Author(s)

Affiliation

Address

email

1 Robot Prototype

Figure 1 depicts the robot used in our experiments, consisting of a flexible backbone stabilized by spacers. At the gripper end, four cables are fixed and pass through the spacers, providing the flexible backbone of the robot. The spacers have enough clearance to follow the curvature of the main backbone. The cables are running in parallel and constrained with respect to each other using spacers. The backbone curvature can be manipulated by pulling and pushing the cables. The robot is powered by four brushless DC motors from Maxon Motors, each equipped with a quadratic encoder. The motors are controlled by PID position controller modules (EPOS4 Compact 50/5 CAN), which receive feedback from the encoders and interface with a PC via the CAN protocol for setting and retrieving control parameters. A Logitech RGB camera is mounted on the robot's base and for precise location tracking of the robot's tip, an ArUco marker [1, 2] is attached to it, which serves as a critical component in the feedback loop of the control system.

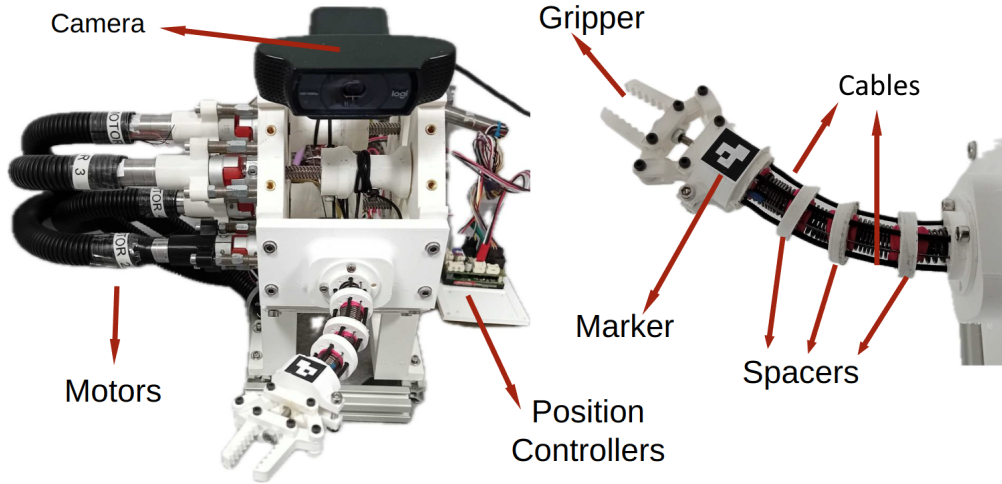


Figure 1: Prototype of our cable-driven continuum robot: The main backbone curvature can be manipulated by pulling and pushing the cables which are controlled by four brushless DC motors, each equipped with a quadratic encoder and position controllers.

Physical parameters of the robot used in the simulations are given in Table 1. In the model verification experiment, the robot's initial length ($\ell(0)$), its second moment of inertia, and polar moment of inertia were measured manually. Later, the robot's tip position calculated from the mathematical model of the robot was compared with the camera measurements at 10 points across the robot's workspace. A least squares algorithm was used to fit model predictions to experimental data to find values of the robot's stiffness and shear modulus. All these parameters are reported in Table 1.

Table 1: Physical parameters of the robot.

$\ell(0)$ [m]	I [m ⁴]	J [m ⁴]	E [kPa]	G [kPa]
0.07	7.363×10^{-9}	1.4726×10^{-8}	300	70

Throughout this work, for all training scenarios, we used the same network architecture and almost the same hyperparameters as described in Table 2.

2 Detailed Information About Reaching Target Task

In the reaching scenario, each episode begins with initializing the environment by randomly placing a target within the robot’s working area (depicted by green cubes in Figure 3 and Figure 4), effectively setting a new goal for each session. The length of each episode is set to one, meaning that upon each reset, the robot receives only one observation — the position of the random target — and must immediately decide on an action. This setup compels the robot to rapidly adapt and optimize its policy to minimize the positional error in a single step. The SAC algorithm continuously updates this policy based on the rewards and penalties received, which assess how closely the robot’s end-effector reaches the target while remaining within operational bounds. This stringent single-step episode structure accelerates the learning process, demanding high efficiency and accuracy from the robot’s decision-making strategies.

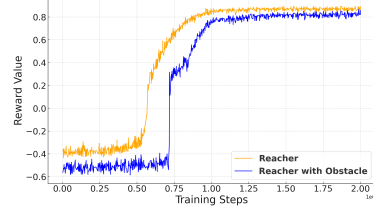


Figure 2: Reward progressions during training.

2.1 Reward Function

The reward function for the soft manipulator robot is designed to finely control the robot’s behavior in a three-dimensional workspace, across two scenarios: reaching a target and reaching while avoiding an obstacle. The function is defined as:

$$\text{reward} = \text{penalty} + e^{-50 \times (\text{distance}^2)},$$

where distance is the Euclidean distance between the robot’s end-effector and the target position. The penalty component is tailored to ensure that the robot operates within its designated bounds and adapts to additional task complexities when an obstacle is present.

For the basic reaching task, the penalty is applied as follows:

$$\text{penalty} = \begin{cases} -0.5 & \text{if } z > 0.28 \text{ or } z < 0.07 \\ 0 & \text{otherwise} \end{cases},$$

this penalty discourages the robot from moving beyond predefined vertical boundaries, effectively reducing the reward when the robot operates outside safe operational zones, thereby enforcing adherence to safe and efficient paths.

In the reaching task with obstacle avoidance, an additional penalty is introduced:

$$\text{obstacle penalty} = \begin{cases} -1.0 & \text{if the robot contacts the obstacle} \\ 0 & \text{otherwise} \end{cases},$$

this ensures that the robot not only aims to reach the target but also learns to navigate around obstacles, further complicating the learning process by penalizing contact with obstacles. Such a mechanism promotes the development of more complex navigation strategies and enhances the robot’s ability to handle real-world environments where obstacles are common.

The use of the exponential decay function, $e^{-50 \times (\text{distance}^2)}$, is critical in both scenarios. It creates a strong incentive for the robot to minimize distance to the target, as rewards diminish rapidly with

Table 2: Network architecture and hyperparameters

Component	Details
Actor Network	MLP with 2 hidden layers Each layer: 256 units, ReLU activation Output layer: Action size, tanh activation
Critic Network	MLP with 2 hidden layers (Twin Critics) Each layer: 256 units, ReLU activation Output: Single value (Q-value), no activation
Learning Rate	0.0003
Batch Size	64
Discount Factor (γ)	0.99
Replay Buffer Size	50000
Number of epoch	2×10^6

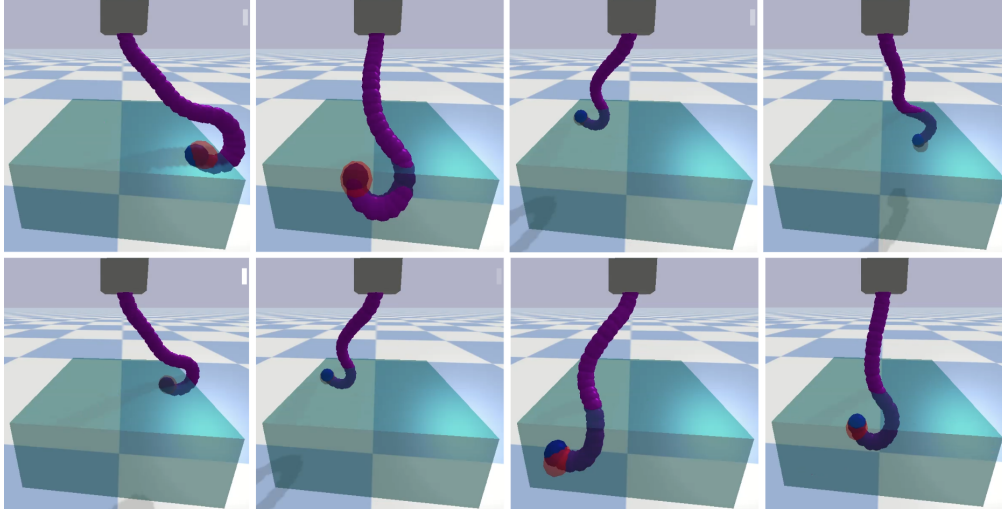


Figure 3: Performance of a five-segment continuum robot in a 3D environment. The robot learned how to reach a target (red sphere), highlighting skills developed through reinforcement learning.

54 increased distance. This sharp gradient is crucial for reinforcing precise and controlled movements
55 of the end-effector towards the target, thereby playing a pivotal role in the learning algorithm by
56 enhancing the speed and accuracy of the robot's operational capabilities in varied task environments.

57 Figure 2 shows the progression of mean rewards received during the training of two policies over
58 2 million steps. This plot illustrates the learning progression for two teacher scenarios: one where
59 the robot solely focuses on reaching a target ("Reacher"), and another where it must reach the target
60 while also avoiding an obstacle ("Reacher with Obstacle"). In both cases, the rewards trend upward,
61 indicating successful learning and adaptation to their respective tasks. However, the introduction
62 of an obstacle in the second scenario introduces a complexity that slightly delays convergence
63 compared to the first scenario. This is reflected in the different trajectories of the reward curves, with
64 the "Reacher with Obstacle" scenario showing a slightly more gradual ascent and later stabilization.
65 The final plateau at a high reward value in both scenarios suggests that the robot effectively learned
66 to reach the target under both sets of conditions, optimizing its path and strategy to maximize the
67 received reward, thereby demonstrating the capability of the reinforcement learning model to adapt
68 to increased task complexity.

69 In the evaluation of the performance over 100 trials, the mean Euclidean distance between the end-
70 effector and the target was found to be 0.0109 meters with a standard deviation of 0.00471 meters
71 in the scenario without obstacles. This demonstrates a high degree of accuracy and consistency in
72 reaching the target. In contrast, when obstacles were introduced, the average distance increased to

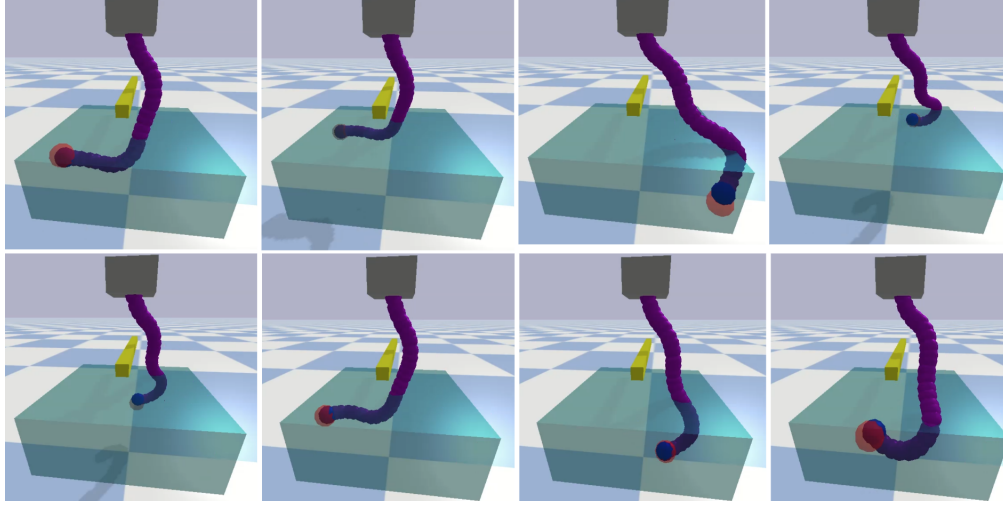


Figure 4: Performance of a five-segment continuum robot in a 3D environment: Demonstrating advanced reaching Skills through reinforcement learning. This robot learned to reach a target (red sphere) while skillfully avoiding a yellow bar obstacle, showcasing its refined abilities acquired from reinforcement learning techniques.

0.0167 meters, and the standard deviation widened to 0.00864 meters. This increase in both the mean and variability indicates a noticeable impact of obstacle presence on the robot’s ability to reach the target precisely, reflecting the added complexity and navigational challenges introduced by the obstacles.

3 Custom Gym Environments

Using SoftManiSim framework, we have developed ten different customized gym environments specifically tailored for continuum robots. Illustrated in Figure 5, these environments are crafted to challenge the robots with a variety of scenarios, each designed to mimic different aspects of real-world applications and the complex physical interactions that continuum robots may face. For detailed examples of how these environments can be applied, please refer to `examples\gyms`.

This development is particularly beneficial for the robotics community as it provides a rich set of tools for testing and refining robotic control policies. By offering a variety of standardized yet challenging scenarios, these environments enable researchers and developers to benchmark and enhance the performance of their robotic systems under controlled but varied conditions. Moreover, sharing these resources fosters a collaborative atmosphere within the community, promoting shared learning and accelerating innovation in robotic design and functionality. The availability of these environments ensures that both new and experienced researchers can explore the nuances of robot-environment interaction, thus contributing significantly to the field of continuum robotics.

4 Real Robot Experiments

4.1 Training Dataset

We aim to generate a dataset to validate and to fine-tune our mathematical model. A series of demonstrations was performed by an operator, who adjusted the lengths of various cables to enable the robot’s tip to move in multiple directions. Data capturing involved recording both the robot inputs, $\mathbf{u}_t \in \mathbb{R}^3$, and the Cartesian coordinates of the robot tip, $\mathbf{x}_t \in \mathbb{R}^3$, at a frequency of 15 Hz, forming the training dataset $\mathcal{D} = \{\mathbf{x}_t^k, \mathbf{u}_t^k\}_{k=1}^N$, $N = 100000$. The camera and marker were used to track the robot’s position. We employed the collected dataset, comparing the model’s outputs with the corresponding targets from the dataset to verify and refine our mathematical model parameters.

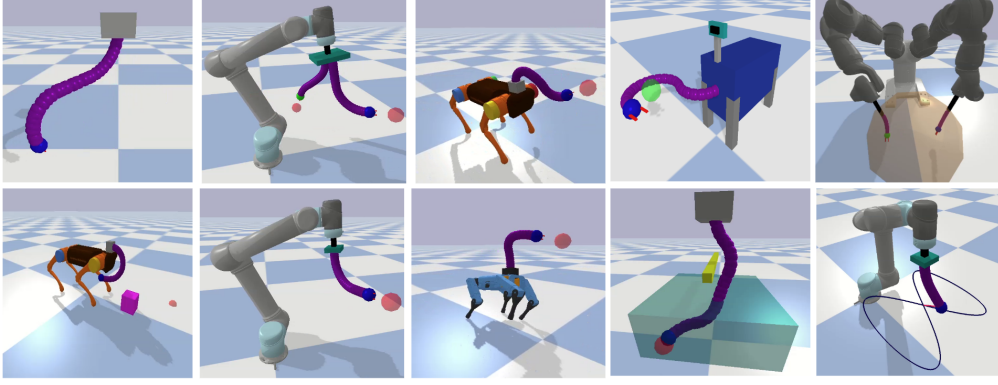


Figure 5: Ten different customized gym environment; To enhance policy learning for continuum robots, we have developed a set of custom Gym environments within our SoftManiSim frameworks that can be used as a baseline.

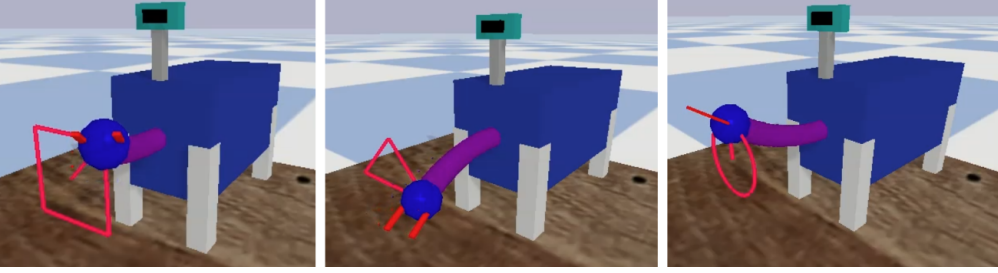


Figure 6: Trajectory tracking results in SoftManiSim: The simulated robot is set to follow a set of predefined trajectories.

Subsequently, the mismatches were utilized to train a shallow neural network, designed to address and compensate for any mismatches in the model.

4.2 Control Policy Learning

After verifying and refining the mathematical model using the dataset, the next step involves designing a control policy capable of effectively managing the dynamics of the continuum robot. The control policy aims to map observed robot states to actions that drive the robot towards a desired state.

We used our customized gym environment for the robot and SAC algorithm to train a control policy. Each episode begins with a random reset of target positions simulating different starting scenarios and enhancing the robustness of the learning process. The reward function is designed to encourage the agent to minimize the distance between the robot’s current end-effector position and the desired position. The reward at each step is calculated as:

$$\text{reward} = e^{-500 \times (\text{distance}^2)}$$

This exponential decay ensures that rewards are higher when the robot’s end-effector is closer to the target, providing a strong gradient for learning. After each interaction, the transitions (state, action, reward, next state) are stored in a replay buffer. The SAC algorithm samples batches from this buffer to update the policy and value networks. The learning process involves adjusting the networks to predict more accurate value estimates and to propose actions that maximize these estimates plus the entropy term.

Post-training, the learned policy is validated both in simulated scenarios and real-world tests to ensure its effectiveness. In these simulations and also the real robot experiments, the robot is programmed to follow designated trajectories in two-dimensional space, including: i) an equilateral triangle on

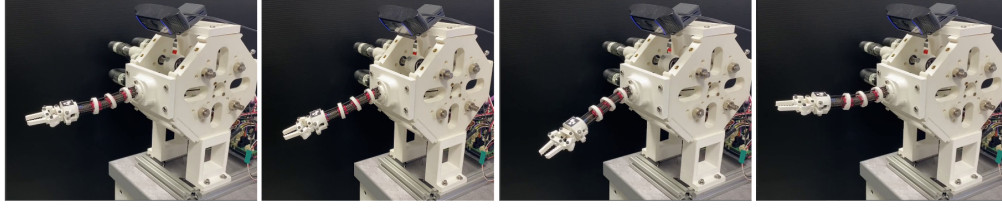


Figure 7: Representative snapshots of the robot while performing the trajectory tracking task (please watch the video).

the XY plane, with each side being 0.04 meters long; **ii**) a square trajectory on the X-Y plane with each side extending 0.025 meters; **iii**) a circular trajectory in the XY plane with a radius of 0.02 meters.

Figure 6 shows the simulation results, as demonstrated, the robot successfully tracked the trajectories. The robot achieved precision with Mean Absolute Errors (MAE) in the x, y, and z directions as follows:

- for the equilateral triangle: 1.77, 1.63, 2.02 mm.
- for the square trajectory: 2.27, 2.03, 2.92 mm.
- for the circular trajectory: 2.54, 1.97, 1.81 mm.

These values demonstrate the robot’s accuracy in tracking the designated trajectories, providing a detailed quantitative assessment of the learned policy’s effectiveness in both simulated and real-world environments.

4.3 Experiments Results

Table 3 presents the Root Mean Square Error (RMSE) measurements in millimeters for trajectory tracking on a real robot, encapsulating its precision across three different geometric paths: triangle, square, and circle. here is the results of trajectori tracking on the real robot:

Table 3: Trajectory tracking results

	RMSE (mm)		
	\tilde{x}	\tilde{y}	\tilde{z}
Triangle	2.87	3.14	3.08
Square	3.08	3.89	3.82
Circle	1.38	1.88	2.19

For the triangular trajectory, the robot exhibited an RMSE of 2.87 mm, 3.14 mm, and 3.08 mm in the x, y, and z directions respectively, indicating a consistent level of precision across all three axes. The square trajectory showed slightly higher errors, with RMSE values of 3.08 mm in x, 3.89 mm in y, and 3.82 mm in z, reflecting the additional challenges this shape may pose in maintaining accuracy. Notably, the circular trajectory demonstrated the best tracking performance with the lowest RMSE values — 1.38 mm in x, 1.88 mm in y, and 2.19 mm in z — highlighting the robot’s enhanced capability to handle continuous, curvilinear paths with higher precision. Figure 7 shows a set of representative snapshots of the robot while performing this task. Our supplementary materials include a video showing the results.

5 Detailed Results of Non-Prehensile Object Manipulation

In this simulation, a continuum robot integrated onto a Unitree A1 quadruped is tasked with non-prehensile object manipulation, specifically pushing a cube towards a target. During the initialization phase in each run (test/train), the environment is set up which positions the target at a randomly determined location with the x-coordinate between 0.55 to 0.7 meters and the y-coordinate between -0.1 to 0.1 meters, ensuring variability and challenge in starting positions for each trial. The reward function is articulated as follows:

$$\text{reward} = e^{-300 \times (\text{distance_obj}^2)} + 0.5 \times (\text{touch})$$

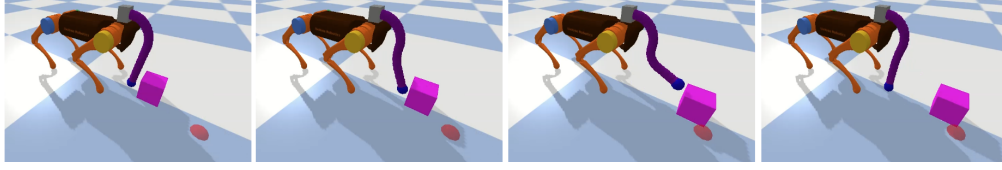


Figure 8: Sequential snapshots showing a quadruped with a three-segment continuum neck, manipulating a cube towards a target (red sphere).

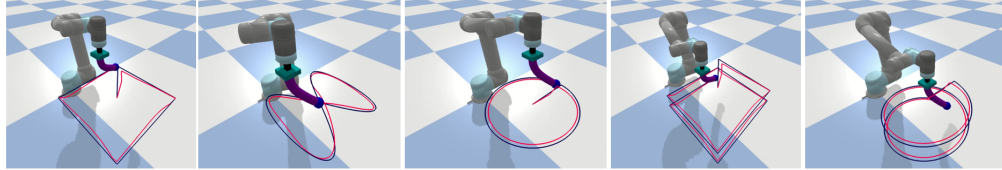


Figure 9: Trajectory tracking results: the robot is tasked with following various paths in both 2D and 3D spaces. The red and black lines indicate the actual and desired paths, respectively.

where distance_obj is the Euclidean distance to the target, and touch is a binary indicator that adds a bonus if the robot's tip makes contact with the cube, thus encouraging effective interaction with the object.

After training using SAC algorithm, the results demonstrate high precision in the robot's performance. The average absolute errors for 50 trials in reaching the target's x and y coordinates are approximately 0.048 and 0.046, respectively, and the average distance from the target is 0.092 meters. These results highlight the effectiveness of the control strategy in enabling the robot to adapt and accurately manipulate objects towards varying target positions. Figure 8 shows sequential snapshots showing a quadruped with a three-segment continuum neck, manipulating a cube towards a target. Our supplementary materials include a video showing the results.

6 Detailed Results of Trajectory Tracking

In this task, a UR5 robot integrated with a two-segment extendable and bendable continuum robot, each segment capable of extending up to 0.03 m, is utilized. The simulated robot was programmed to follow various complex trajectories in both 2D and 3D spaces, designed to test its precision and control capabilities. These trajectories included a square in the X-Y plane with 0.4 meters sides, a figure-eight curve described by specific sinusoidal equations for x and y coordinates over a 20-second period, a circular path with a 0.2-meter radius, a helical trajectory with a 0.2-meter radius and a 0.1-meter pitch, and a square-helical path combining square and helical movements. The effectiveness of the robot's path following was quantitatively assessed by calculating the Mean Squared Errors (MSE) in the X, Y, and Z coordinates for each trajectory. The results, summarized in the provided Table 4 and shown in Figure 9, indicate varied performance across different trajectories. The helical trajectory showed the most precise control, with the lowest average MSE of 0.000153, suggesting that the robot manages consistent vertical movements well. The circular trajectory also exhibited low error rates, emphasizing the robot's ability to maintain steady curvilinear motion. In contrast, the figure-eight and square trajectories had higher MSEs, particularly in the horizontal plane, indicating challenges in managing more complex path changes and corner navigation. The square-helical trajectory achieved a moderate average MSE, highlighting a blend of challenges in maintaining precision in both linear and vertical displacements. These insights can guide further refinements in control algorithms, particularly focusing on improving accuracy in trajectories involving abrupt direction changes and complex geometric patterns.

Table 4: Mean Squared Errors (MSE) for Different Trajectories.

Trajectory	MSE X	MSE Y	MSE Z	Average MSE
Square	0.000592	0.000398	0.000062	0.000351
Circle	0.000214	0.000341	0.000030	0.000195
Eight Figure	0.000930	0.000223	0.000075	0.000409
Helix	0.000190	0.000256	0.000014	0.000153
Moving Square	0.000431	0.000333	0.000041	0.000268

7 Python Interface

The `SoftManiSim` class is designed to facilitate the simulation of soft robots using Pybullet physics engine. This class serves as a comprehensive interface that initializes and manages various aspects of the simulation environment, ensuring a seamless and flexible setup process. The constructor of the `SoftManiSim` class takes several parameters, including an optional bullet instance, number of segments (`_number_of_segment`), color configurations for the robot’s body and head (`body_color`) and (`head_color`), the radius of the body spheres (`body_sphere_radius`), the number of spheres composing the robot’s body (`number_of_sphere`), the number of segments in the robot (`number_of_segment`), and a boolean to toggle the graphical user interface (GUI). If no bullet instance is provided, the constructor initializes a new Pybullet instance. The `create_robot` method is invoked at the end of the constructor to assemble the robot based on the provided parameters, ensuring that all necessary components are correctly instantiated and configured. This methodical and thorough initialization process makes the `SoftManiSim` class a powerful tool for researchers and developers, offering a high degree of control and customization over the soft robot simulation, ultimately contributing to more efficient and accurate experimental setups in the field of soft robotics.

7.1 API Documentation

Below is the API documentation for the `SoftManiSim` class, detailing essential methods, their arguments and functionalities:

Table 5: API Descriptions of SoftManiSim

Method	Argument	Description
<code>__init__</code>	bullet body_color head_color body_sphere_radius number_of_sphere number_of_segment gui	Optional physics engine instance, defaults to None, initializes PyBullet if not provided. RGBA color for the robot's body. RGBA color for the robot's head. Radius of spheres used to build the robot's body. Number of spheres constructing the robot's body. Number of segments in the robot's body. Boolean to toggle graphical interface, defaults to True.
<code>create_robot</code>	-	No arguments, sets up the robot's physical structure within the simulation. This function is invoked at the end of the constructor.
<code>move_robot_ori</code>	action base_pos base_orin camera_marker	Array of actions defining movement commands for robot segments. The base position of the robot in the simulation space. The base orientation of the robot, specified as Euler angles. Boolean to display camera markers, defaults to True.
<code>calc_tip_pos</code>	action base_pos base_orin	Array of actions affecting the tip's position and orientation. The base position from which the tip's calculations start. Base orientation affecting the tip's calculation.
<code>capture_image</code>	removeBackground	Boolean to decide whether to remove background from the image, defaults to False.
<code>in_hand_camera_capture_image</code>	-	No arguments, captures image from the robot's in-hand camera.
<code>is_robot_in_contact</code>	obj_id	Object ID to check for contact with the robot.
<code>is_gripper_in_contact</code>	obj_id	Object ID to check for contact with the robot's gripper.
<code>suction_grasp</code>	enable	Boolean to enable or disable the suction grasp mechanism.
<code>set_grasp_width</code>	grasp_width_percent	Percentage of maximum grasp width to set for the gripper.
<code>add_a_cube</code>	pos ori size mass color textureUniqueId	Position to place the cube in the simulation. Orientation of the cube, given as a quaternion. Dimensions of the cube. Mass of the cube. RGBA color of the cube. Optional texture ID for the cube's surface.
<code>wait</code>	sec	Duration in seconds to delay the simulation.

References

- [1] S. Garrido-Jurado, R. M. noz Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280 – 2292, 2014. ISSN 0031-3203. doi:<http://dx.doi.org/10.1016/j.patcog.2014.01.005>. URL <http://www.sciencedirect.com/science/article/pii/S0031320314000235>.
- [2] S. Garrido-Jurado, R. M. noz Salinas, F. Madrid-Cuevas, and R. Medina-Carnicer. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51: 481 – 491, 2016. ISSN 0031-3203. doi:<http://dx.doi.org/10.1016/j.patcog.2015.09.023>. URL <http://www.sciencedirect.com/science/article/pii/S0031320315003544>.