Autonomous Self-Constructing Modular Robotic Arm using MVMAE-SAC Control for Adaptable Disaster Response

Daniel Zhu
Abbey Park High School, Oakville, ON, Canada
danielzhu501@gmail.com

Abstract

Small robotic vehicles can navigate through narrow gaps in rubble but typically lack the strength and dexterity required for manipulation tasks, while larger robotic systems capable of such tasks are often unable to pass through confined spaces that block access to the affected area. The purpose of this project is to investigate whether a low-cost, 3D-printed, modular robotic system can bridge this gap by navigating through confined spaces as separate vehicles before self-assembling into a manipulator, controlled by a reinforcement learning agent operating on only a monocular camera and Time-of-Flight sensor, eliminating the need for expensive sensing hardware such as motor encoders, LiDAR, or depth cameras and reducing overall system cost to a level accessible to underfunded agencies and first responder teams. During training, the camera feed is duplicated and randomly augmented to create multiple views that are encoded using a Multi-View Masked Autoencoder (MVMAE) for spatial representation learning and sim-to-real adaptability. The encoder is trained jointly with a Soft Actor-Critic (SAC) policy, with gradients propagated through the combined encoder reconstruction and policy loss. The agent trained using the MVMAE architecture achieved a 12.4% increase in average episodic reward and a 40.4% reduction in action standard deviation compared to the single-view augmented benchmark, indicating more stable control behavior. The MVMAE-SAC trained model achieved an 89.6% success rate on target-reaching tasks.

Index Terms

Modular robotics, reinforcement learning, masked autoencoders, sim-to-real transfer.

I. Introduction

The February 2023 Turkey-Syria earthquake killed over 50,000 people and damaged more than 160,000 buildings, demonstrating a severe limitation in current rescue robotics: smaller robots can navigate through tight spaces but lack the strength to manipulate objects, while large manipulators like the Brokk series provide the necessary reach and payload but cannot fit through confined passages in collapsed structures. Modular self-reconfigurable systems such as M-TRAN and SMORES-EP can change morphology but prioritize locomotion over manipulation and rely on expensive hardware inaccessible to most first responders. The proposed system prototype is a 3D-printed, modular, self-constructing robotic manipulator designed to navigate through confined spaces as separate vehicles before assembling into a larger arm capable of performing manipulation tasks. The assembled arm is controlled by an agent, trained using a novel reinforcement learning framework, that receives input from a wrist-mounted monocular camera and Time-of-Flight (ToF) sensor.

We test if a distributed system of mobile robots can autonomously self-assemble into an affordable but effective functional robotic arm capable of performing accurate manipulation tasks using a policy controlled by a wrist-mounted camera and Time-of-Flight sensor.

II. Preliminaries

A. DrQv2-Style Soft Actor-Critic (SAC)

The actor network consists simply of three forward linear layers, with activation functions. The forward layers output a mean action μ, which is passed through a tanh function to clip its values between [-1, 1]. Next, a standard deviation σ is chosen from the scheduler, which gives higher standard deviations early in training to encourage exploration, and lower standard deviations in later stages to encourage deterministic actions. The mean action and selected standard deviation are used to generate a Gaussian distribution centered at μ and σ as the spread. The mean action, which represents the policy's predicted optimal action for the current state, is not executed directly by the agent during training. Instead, the action is randomly sampled from a Gaussian distribution centered at this mean.

The online critics predict Q-values for an action a_t taken under the observation state s_t. A Q-value represents the expected cumulative discounted reward the agent is predicted to receive under the given circumstances. Using two separate critics, two different Q-values, Q_1 and Q_2, are estimated using a three-layer neural network similar in structure to the actor network. The target critics calculate the target Q-value, representing a stable value that the online critics should learn to predict given a certain action-state pair. It is calculated using the immediate reward and the discounted estimate of the next state's value, computed from the target critics

$$Q_{\text{target}} = r_t + \gamma \min\left(Q'_1(s_{t+1}, a_{t+1}),\ Q'_2(s_{t+1}, a_{t+1})\right) \quad (1)$$

Target critics are able to produce more stable Q-value predictions because they are delayed versions of the online critics, meaning their weights update much more slowly. This is achieved using Polyak averaging, where the target critic weights gradually track the online critics rather than changing abruptly. The minimum of the two target Q-values is taken

because it is a well-known issue in actor-critic methods that Q-values can become overestimated.

In off-policy reinforcement learning, transitions from past timesteps are stored and reused later for training instead of being discarded after one update. The Soft Actor-Critic algorithm is one such example. It stores states, next states, actions, and rewards as transitions in a replay buffer. These transitions are randomly sampled in batches during training steps. The critic networks are run, outputting predicted Q-values Q_1 and Q_2, and a stable target Q-value Q_{target}, which are used to compute a mean squared error (MSE) loss representing how close the predicted Q-values were to the target

$$\mathcal{L}_{\text{critic}} = \frac{1}{N}\sum_{i=1}^{N}\left[\left(Q_1^{(i)} - Q_{\text{target}}^{(i)}\right)^2 + \left(Q_2^{(i)} - Q_{\text{target}}^{(i)}\right)^2\right] \tag{2}$$

The actor loss is taken using simply the -mean(Q), where Q is the Q-value predicted by the actor. Networks are trained with the Adam optimizer [4].

B. Multi-View Masked Autoencoder (MVMAE)

The encoder receives two image views. In standard MVMAE, these images come from different camera views. However, in the proposed utilization of MVMAE, the same camera view is copied, augmented, and then passed as a second view. Each view is first converted into a sequence of patch tokens by a small strided convolutional layer. This stage effectively divides the image into patches and generates a sequence of patch tokens. After patch embedding, 2D sine-cosine positional embeddings are added to each token. These positional embeddings encode the spatial position of each patch within the image grid, allowing the transformer to retain information about patch locations, since the transformer architecture itself does not preserve spatial structure.

The encoder then applies multi-view masking. In this masking strategy, the two views are treated differently. One view is fully masked, meaning all of its patch tokens are hidden from the encoder. The second view is partially masked, where only a subset of patch tokens remain visible, and the rest are removed. As a result, the encoder only processes a small number of visible tokens from one view while receiving no direct visual information from the other view.

The remaining visible tokens are then processed by a stack of Vision Transformer blocks. Self-attention allows each visible patch token to incorporate contextual information from other visible patches, while the MLP applies nonlinear transformations to refine the token features.

Because one view is completely masked, the encoder must rely on the partially visible tokens from the other view to capture the underlying structure of the scene. This encourages the encoder to learn view-invariant representations that capture semantic information about objects rather than memorizing the appearance of individual patches. This is particularly beneficial in settings where the two views are augmentations representing real-world camera artifacts such as noise, blur, and exposure variation. As a result, the model is forced to learn representations that are invariant to these distortions, since the only way to reconstruct the masked view is to understand the underlying scene structure rather than artifact-specific appearance.

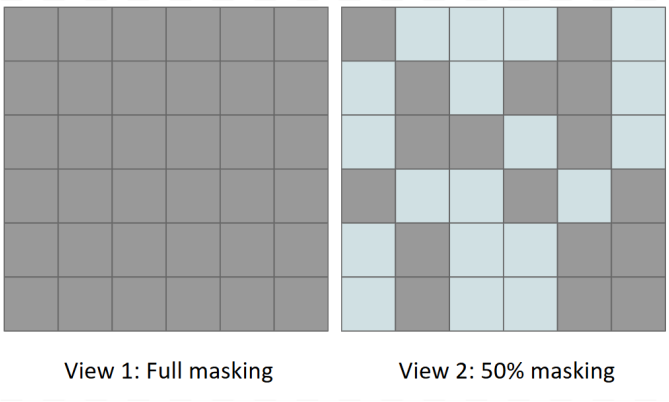

Fig. 1. Multi-view masking: one view fully masked, the second partially masked.

The decoder is a lightweight transformer that takes as input the full set of patch tokens from both views, including both the visible encoded tokens produced by the encoder and the masked tokens, which are represented as learned mask tokens. Positional embeddings are re-added at this stage to ensure each token retains its spatial location regardless of whether it was visible or masked during encoding. The decoder then applies a series of transformer blocks, allowing the mask tokens to attend to the encoded visible tokens and aggregate the contextual information needed for reconstruction. A linear projection head maps the final token representations to pixel space, producing a reconstruction of each patch. The reconstruction loss is computed as the mean squared error between the predicted and original pixel values of the masked patches only, following the approach of MAE [3]. This forces the decoder to properly recover the masked regions, and in turn forces the encoder to produce embeddings representative enough to support that recovery. The decoder is discarded in the RL pipeline during inference; it exists purely to train the encoder.

III. System Design

A. Hardware Components

The Raspberry Pi Zero 2W (PiZW) was selected as the primary embedded system due to its extensive versatility in each respective field. The PiZW features a 2.4GHz 802.11 b/g/n Wi-Fi chip, providing a stable 30-35 Mbps for fast and reliable communication via the host computer. The model also provides a Camera Serial Interface (CSI) port allowing one ArduCam to be connected, as well as a dedicated Serial Data Line (SDA) and Serial Clock Line (SCL) utilized by the Inter-Interconnected Circuit (I2C) communication protocol vital for supporting the distance measuring Time-of-Flight (ToF) sensor.

The DS3218 20kg metal gear servo was selected as the joint actuator for all three degrees of freedom. Standard hobby servos with plastic gear trains were considered as a lower-cost alternative, but plastic gears deform progressively

under repeated high-torque loading, making them unsuitable for sustained rescue manipulation tasks. The DS3218's metal gear train provides significantly higher durability under these conditions; its 21.5 kg·cm stall torque provides a sufficient margin for payload capacity, and its operating range of -15°C to 70°C covers the environmental conditions expected in collapsed structure rescue scenarios.

The 12V 30RPM worm gear motor was selected for the base movement and rotation joint, which serves as one of the three degrees of freedom of the arm. Worm gear mechanisms are self-locking, which means the worm can hold its rotational position against gravitational and lateral loads even without power. This property prevents the base wheels from being influenced by inertia, as well as increasing the rotational accuracy. Furthermore, the ability to hold itself without continuous power consumption helps to increase battery efficiency, a metric vital for limited operation runtime situations within search and rescue contexts.

In addition, the N20 12V 53RPM metal gear motor was selected for driving the wheels of the middle and end modules. The N20's compact form factor fits within the tight spatial constraints of the modular chassis geometry, while its metal gear train provides higher durability than plastic alternatives under the continuous rotational loading of a driven wheel across varied rubble terrain. At 53RPM, the N20 provides sufficient speed for navigation while remaining controllable enough for precise docking, and its standard spur gear arrangement is appropriate since wheel actuation requires no positional holding.

The system requires three distinct voltage rails: 6.8V for the DS3218 servos, 5V for the Raspberry Pi Zero 2W, and 12V for the drive motors, all derived from the 7.4V nominal battery. The XL4015 buck converter, with a conversion efficiency of 85% to 90%, was selected for both the 5V and 6.8V. The 12V motor rail requires boost conversion since it exceeds the battery voltage, and the XL6019 buck-boost converter was selected for its step-up voltage efficiency of 90% to 94%. Selecting dedicated switching converters over a single linear regulator solution helps to reduce thermal output, thereby extending mission runtime, both critical constraints in search and rescue deployment.

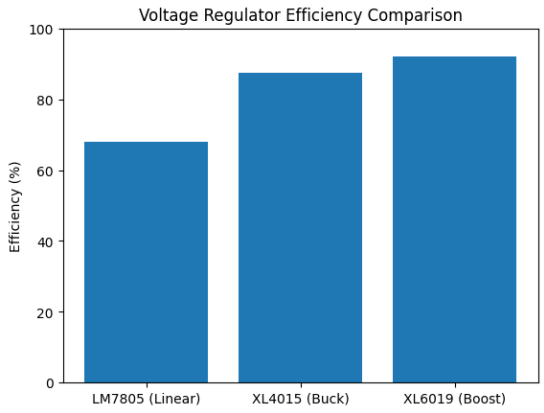

Fig. 2. Power distribution across the three voltage rails.

MOSFET-based drivers such as the TB6612FNG use metal-oxide-semiconductor field-effect transistors (MOSFET), which have a significantly lower on-resistance, typically 0.5Ω per channel, resulting in a voltage drop of under 0.5V during usage. The TB6612FNG also supports two independent motor channels, PWM speed control up to 100 kHz, and a maximum continuous current of 1.2A per channel with 3.2A peak, sufficient for the drive motors used in this system. The TB6612FNG was therefore selected over the L298N due to having approximately 20% higher efficiency and producing approximately 1.5W less thermal energy per channel, caused by the significantly lower on-resistance of MOSFETs compared to BJTs.

The PiZW's single CSI port prevents the use of stereo cameras, the most common configuration used for depth perception in reinforcement learning robotics. Therefore, a system combining a Time-of-Flight (ToF) sensor and an ArduCam, where the ToF sensor compensates for the lack of stereo depth by providing accuracy ranging from 3cm up to 400cm, regardless of surface colour or ambient light conditions. Infrared proximity sensors were considered for ranging but rejected due to susceptibility to surface reflectivity and ambient light interference common on the irregular surfaces in rubble environments. Another candidate was LiDAR sensors, which provide a higher angular resolution and longer range. However, their cost of $100 to several thousand dollars per unit makes them impractical for a modular and budget-oriented platform. The ArduCam was selected for its direct compatibility with the PiZW's CSI port through an adapter cable, providing lower latency than USB alternatives.

A 7.4V 3,000mAh 2S 15C Li-Ion battery was selected over LiPo and NiMH alternatives. NiMH batteries were rejected due to their significantly lower energy density of approximately 60 to 120 Wh/kg compared to 150 to 200 Wh/kg for Li-Ion, meaning an equivalent NiMH pack would be substantially heavier, reducing load capacity. LiPo batteries offer comparable energy density, but their soft pouch casing makes them more prone to physical damage during potential collisions, whereas Li-Ion cells in hard cylindrical casings provide better mechanical protection.

B. Communication Between Modular Robot Segments

Each module of the robot requires real-time communication for coordinated autonomous operation and joint commands, while still requiring real-time low-latency frames and ToF sensor data to be sent between the PiZWs and the host computer running RL inference. Bluetooth was considered a lower-cost alternative; however, because it only has a maximum output of approximately 2 to 3 Mbps, this would be insufficient for transmitting camera frames at the frame rates required for real-time inference. This is why the PiZW's integrated 2.4GHz 802.11 b/g/n WiFi was selected instead, providing 30 to 35 Mbps of stable throughput sufficient for simultaneous image transmission and control signalling across all three modules. On a local network, UDP achieves a median one-way latency of approximately 12 milliseconds compared to 16 milliseconds for standard TCP; while faster, UDP's connectionless fire-and-forget approach ultimately risks corrupting joint commands

during a rescue manipulation sequence. Therefore, MQTT over TCP was selected with TCP_NODELAY explicitly enabled, disabling Nagle's algorithm, which would otherwise buffer small control packets, essentially bundling up 10 minor 1-degree servo adjustments into 1 major 10-degree command, and introducing up to 40ms of artificial delay per command, achieving latency approaching UDP while retaining TCP's guaranteed delivery.

The RL inference model is computationally too expensive to run on the PiZW's quad-core 1GHz ARM Cortex-A53, requiring it to run on a host computer instead. An ad-hoc peer-to-peer WiFi network between modules was considered as the first possible candidate, but this was quickly ruled insufficient since all data must route through the host computer anyway. The next and final approach was to have all three modules connected to a shared router, with the host computer on the same network acting as the central inference node. MQTT was selected as the messaging protocol for its publish-subscribe architecture, lightweight header overhead, and local network latency of approximately 1 to 15 ms for small payloads, which is suitable for the control loop frequency required by the RL controller.

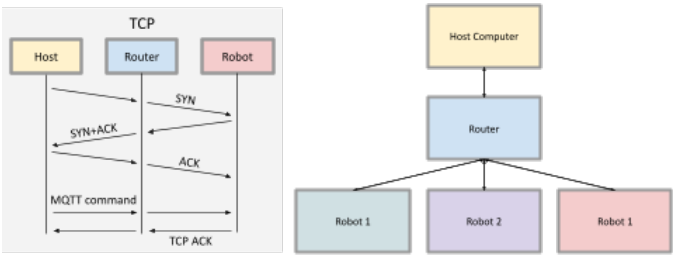

Fig. 3. Network architecture: modules and host computer on a shared router.

Command latency directly affects the feasibility of sim2real transfer, since the simulation assumes instantaneous joint commands while the real system introduces network delays between inference and joint output. DiAReL [5] notes that communication latency introduces action delays that can violate the Markov property in standard MDPs, causing suboptimal policy behaviour during sim2real transfer. However, on a local WiFi network, MQTT delivers control messages with latencies of approximately 1 to 15 ms, which is negligible relative to the DS3218's mechanical response time of approximately 140 ms per 60° of travel under no-load conditions, meaning the real system behaves effectively instantaneously from the perspective of the RL control loop and the Markov property is preserved in practice. Clock synchronisation across the three Raspberry Pi Zero 2Ws is managed using NTP, which provides synchronisation accuracy of approximately 1 to 10 ms on a local network, sufficient for correlating sensor timestamps across modules during state observation construction for the RL controller.

IV. Method

A. AI Control Pipeline

With the goal of maintaining affordability, complex sensing systems and high-precision actuators are outside the scope of this project. Thus, a novel control pipeline must be developed. Industrial manipulators typically rely on extensive sensing infrastructure, including motor encoders for joint position and velocity feedback, tactile sensors, LiDAR or depth cameras for spatial mapping, and inertial measurement units (IMUs). In contrast, the system developed in this project relies only on an RGB image from a low-resolution camera and a distance measurement from a direct time-of-flight (dToF) sensor. Both sensors are mounted on the robot wrist at the base of the end effector and face forward toward the workspace.

Reinforcement learning methods used in robotics commonly operate on structured state inputs such as joint encoder values, torque measurements, or organized range sensor data. These signals are typically low-dimensional and directly correspond to physical quantities describing the robot and its environment. However, an RGB image represents a high-dimensional and unstructured observation consisting of thousands of pixel values. Directly using raw pixel inputs significantly increases the difficulty of learning meaningful representations and makes policy convergence substantially slower and more sample inefficient. Additionally, the extreme disparity between the high dimensionality of the RGB image, which may contain tens or hundreds of thousands of pixel values, and the single scalar distance measurement from the ToF sensor can cause the visual input to dominate the learned feature representation.

To address this challenge, the image observation is first processed by a Vision Transformer (ViT) encoder. Vision Transformers adapt the transformer architecture, originally developed for sequence modeling in natural language processing, to image data by dividing the image into patches and treating them as tokens. Through self-attention, the encoder learns spatial relationships between different regions of the image and produces a compact embedding that captures the global visual context. These embeddings provide a structured spatial representation that can be more effectively used as input to the reinforcement learning policy.

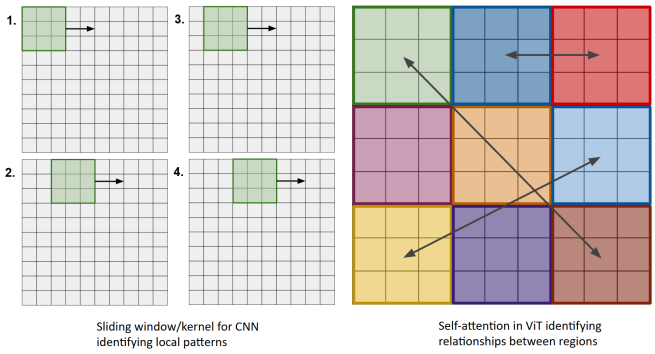

Fig. 4. ViT-based encoding pipeline.

Traditionally, Convolutional Neural Networks (CNNs) are used to identify patterns in images. However, a CNN, or any of its variants, was not chosen for this project. This was an intentional design choice. CNNs use a sliding kernel, meaning they determine patterns in local areas of the image, but they have significant difficulty interpreting the overall global context of a scene. Vision Transformers, on the other hand, use

self-attention to learn relationships between different patches of the image, allowing them to capture spatial relationships between regions and thus understand the global context.

The processed embeddings from the ViT encoder are then flattened and concatenated with the ToF sensor readings, and then fed as input into the policy. For the proposed pipeline, Soft Actor-Critic was chosen as the appropriate policy to use. SAC is an off-policy algorithm, meaning it stores past states inside a replay buffer instead of simply discarding them like on-policy algorithms such as PPO. Random batch sampling from the replay buffer allows the agent to be exposed to the same observations multiple times, enabling it to extract more learning signals from each experience. In addition to significantly increasing sample efficiency, using a replay buffer also decorrelates the data used to train the ViT encoder. The SAC policy uses entropy regularization. SAC adds an entropy term to its objective and attempts to maximize the sum of the reward and the entropy term, which encourages exploratory or random actions.

MuJoCo is a high-performance physics engine designed for robotics and control research, capable of simulating rigid body dynamics, contacts, forces, and sensors with high numerical accuracy. In this environment, the robot joints are controlled using incremental angle commands ranging from -1 to 1 degrees, forming a continuous action space. This control strategy was chosen to accommodate the servo motors' lack of encoders and the limited control interface of the Miezei servos. These servos can only be commanded by specifying a target angle, which prevents reliable measurement or direct control of torque and velocity. Furthermore, due to inaccuracies in servo positioning caused by the weight of the arm, incremental control allows the robot to make frequent small corrections rather than committing to large movements that may overshoot the target. The observation space is constructed from two sensor modalities. First, a camera mounted on the robot wrist, just below the end effector, provides visual observations of the scene. Second, a Time-of-Flight (ToF) distance measurement is simulated using ray tracing within MuJoCo. The ToF reading is normalized by dividing the measured distance by 2.0 m and clipping the result to the range [0, 0.99]. Invalid or missing readings are assigned a value of 1.0, allowing the policy to distinguish between a valid far-distance reading and sensor failure. Together, these sensors provide the agent with visual and depth information necessary to localize and approach the target object. The following figures show visualizations of the robot setup within the MuJoCo environment.

The right scene image is not used as input to the policy and serves only as a visualization for the user, while the left POV image is fed into the policy during training and evaluation.

1) Flow of a Training Update Step: The update step is designed to step the gradients of both the ViT encoder, actor, and critic, effectively training the encoder simultaneously with the policy. Below is a diagram showing the workflow starting from the sampling of a transition from the replay buffer to the generation of the total loss. The flowchart is color-coded as follows: red denotes parts of the MVMAE framework, blue denotes parts of the critic networks, and green denotes actor components. Loss calculation nodes are highlighted in yellow.

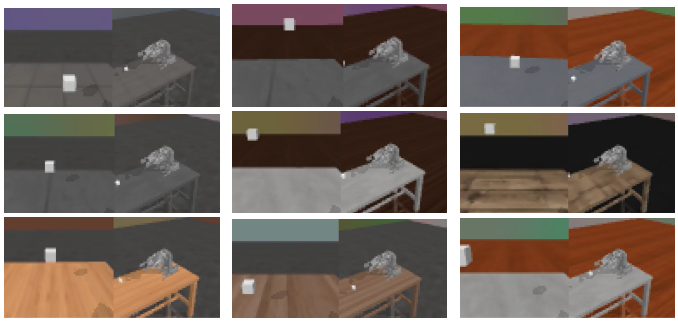

Fig. 5. Visualizations of the robot setup within the MuJoCo environment.

Gray represents miscellaneous operations and nodes.

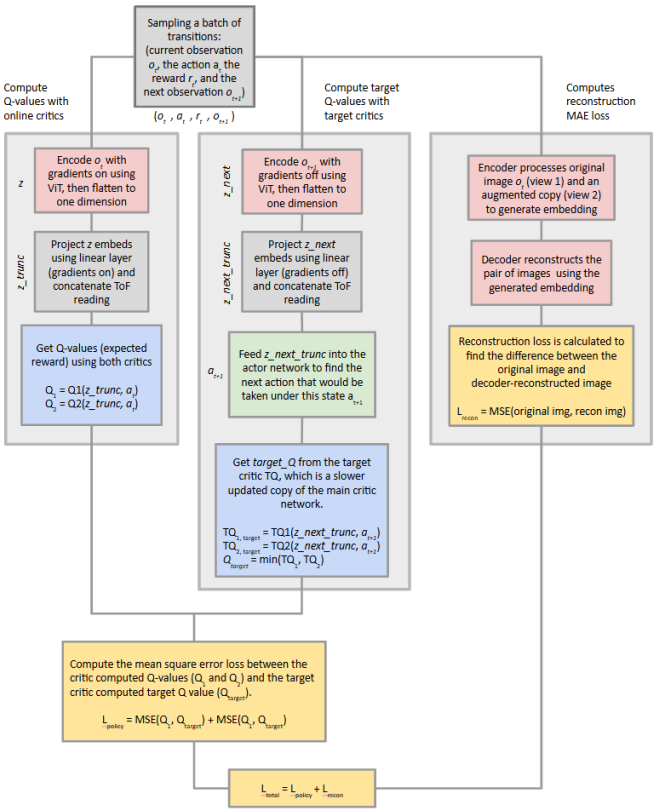

Fig. 6. Training update flowchart: red – MVMAE framework; blue – critic networks; green – actor components; yellow – loss computation; gray – miscellaneous operations.

The update begins by sampling a batch of transitions from the replay buffer, containing the current state s_t, the action a_t, the reward r_t, and the next observation s_{t+1}. The state o_t consists of the current image frame and the Time-of-Flight (ToF) sensor reading. The image is first encoded by a ViT encoder to produce a latent embedding. During this step, gradients are enabled so that the encoder parameters can be updated during training. A linear projection layer is used to map the latent embedding to a smaller dimension so that it does not squash the ToF sensor signal during learning. The ToF reading is then concatenated to the latent embedding. Two online critics are each fed the final concatenated embedding and the action a_t

as input, and two Q values are calculated. The image in state s_{t+1} is passed through the encoder with gradients disabled. This forward pass is used only to compute the target Q-value for the critic update. The latent embedding is passed through a linear projection layer to reduce dimensionaly and the ToF value is concatenated. Use the actor, with gradients off, is used to calculate what action would be taken in the state s_{t+1}. This is action a_{t+1}. Using the two target critics, which are slow-moving copies of the two online critics, each is fed the action a_{t+1} and the concatenated embedding. Two target Q values are calculated, and their min is taken. Taking means square error loss between the target Q value and the computed Q values is taken.

The MVMAE framework, including the encoder and decoder components, must be updated in this step. The input image and an augmented copy of the same image are passed through the encoder, which first divides both images into patches. A random subset of patches from one view is masked according to a predefined masking ratio, while the other view is fully masked. Only the remaining visible patches are processed by the encoder to produce a latent representation. The encoder's output, along with mask tokens representing the missing patches and their positional information, is then passed to the decoder. The decoder attempts to reconstruct the original image pair by predicting the pixel values of the masked patches using information from the unmasked patches of the other view. The reconstructed patches are matched against the corresponding patches from the original input image pair to calculate a reconstruction loss, which is computed as the mean squared error between the predicted and ground-truth pixel values.

Total loss is calculated by taking the sum of the policy loss and the reconstruction loss. Backpropagation is run on the total loss. By backpropagating the combined loss, gradients from both the reinforcement learning objective and the reconstruction objective update the shared encoder. This encourages the encoder to learn representations that both support accurate reward prediction and capture the underlying structure of the environment. Gradient descent is performed on all parameters using the Adam optimizer. There are two parameter groups to update. The first updates the parameters associated with the policy loss, including the actor and critic networks. The second updates the parameters associated with the reconstruction loss, such as the decoder and other components used for image reconstruction.

The reward function is designed to provide dense rewards that guide the goal of the policy, which is to reach a cube at random positions in space. The reward components are as follows:The approach reward is a long-tail tolerance function of the camera-to-cube distance. Full reward is given for distances up to 0.06 m, and the reward gradually decreases beyond this range. This shaping encourages the robot to reduce the distance to the cube while still providing informative feedback when the robot is farther away. The alignment reward is a long-tail tolerance function of the combined angular misalignment from the optimal centering of the cube. The misalignment is computed as the Euclidean magnitude of the horizontal and vertical angular errors relative to the optimal viewing

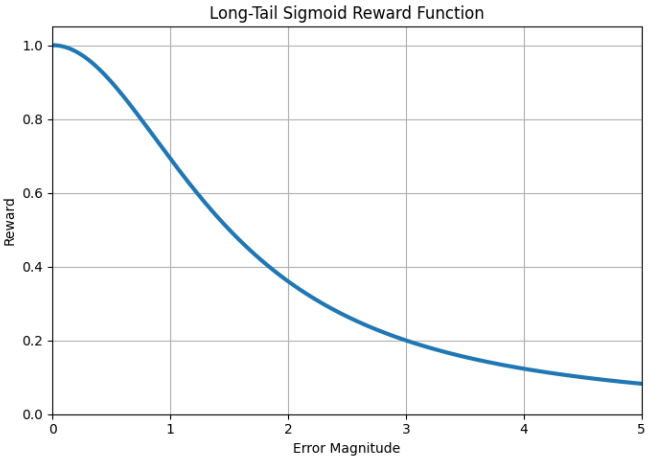

Fig. 7. Long-tail sigmoid alignment reward as a function of error magnitude.

direction. This term encourages the robot to orient the camera such that the cube is centered within the field of view.The ToF proximity reward is a long-tail tolerance function of the measured Time-of-Flight (ToF) distance when the sensor ray intersects the cube. Full reward is given for distances up to 0.03 m, corresponding to close proximity to the object. Beyond this range, the reward decreases smoothly according to the long-tail tolerance function, encouraging the robot to move progressively closer to the cube while maintaining correct sensor alignment. Due to the nature of a sigmoid long-tail function, the reward tends to plateau as error reaches zero. This gives the robot no incentive to close the last small distance between the target, since it gains no extra reward. Thus, an extra bonus was added for any proximity closer than 10 cm.

The importance of passing errors through a long-tail sigmoid function instead of using a linear reward function is to ensure that rewards decay gradually as the error increases. This prevents the reward signal from becoming uninformative when the agent is far from the target. A linear reward function typically decreases at a constant rate and may quickly reach very large negative values or zero, providing little useful feedback for learning. In contrast, the long-tail sigmoid decreases more slowly, allowing the agent to still receive meaningful reward signals even at large errors. As illustrated in the graph above, the reward remains slightly positive when the error is large, encouraging the agent to continue reducing the distance to the target rather than receiving no learning signal at all.

Augmentations are made to the policy input to mimic actual non-ideal conditions. In addition, scenes are randomized so the policy can transfer to real-world target placement and background textures. The following techniques are randomly applied to each episode or frame.

- *Scene appearance:* texture randomization, skybox gradient variation, target start-position randomization, cube rotation randomization, lighting direction randomization, lighting position randomization, diffuse intensity variation, ambient intensity variation
- *Camera imaging:* camera translation drift, camera rotation drift, camera scale drift, border fill variation dur-

ing warping, exposure variation, gamma variation, white balance variation, local illumination/vignette variation, optical blur variation, resolution downsampling and resampling, interpolation method variation, sensor read noise, signal-dependent shot noise, JPEG compression artifacts, JPEG quality variation

- *ToF sensing:* measurement bias, distance-dependent Gaussian noise, millimeter quantization, near-range specular failure simulation, random measurement spikes, measurement dropout, temporal smoothing using EMA
- *Temporal latency and delays:* action delay randomization, observation delay randomization

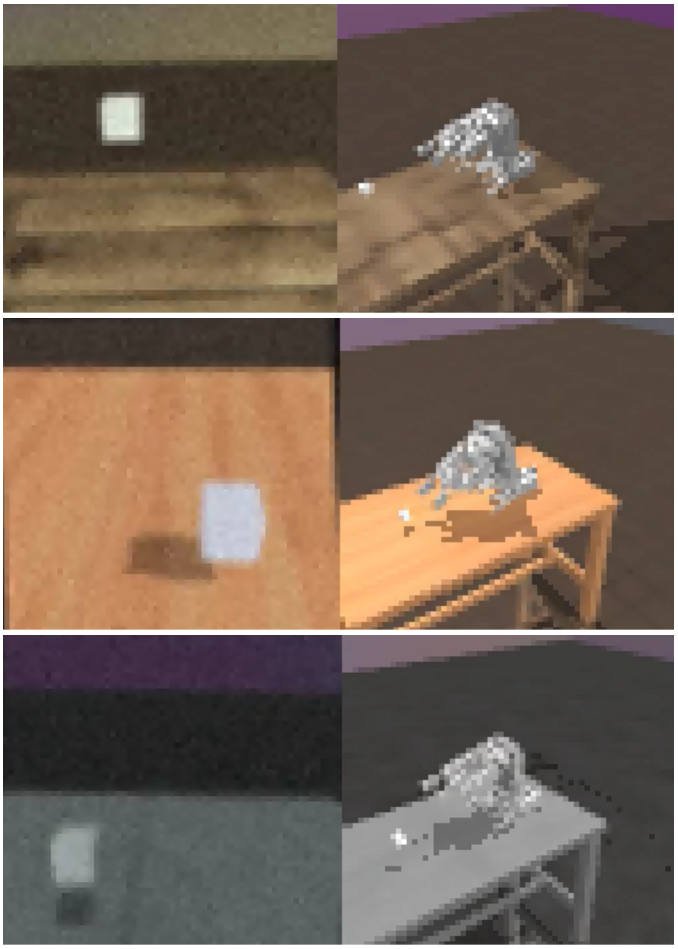

Fig. 8. Randomized simulation scenes: camera observation (left) and scene view (right).

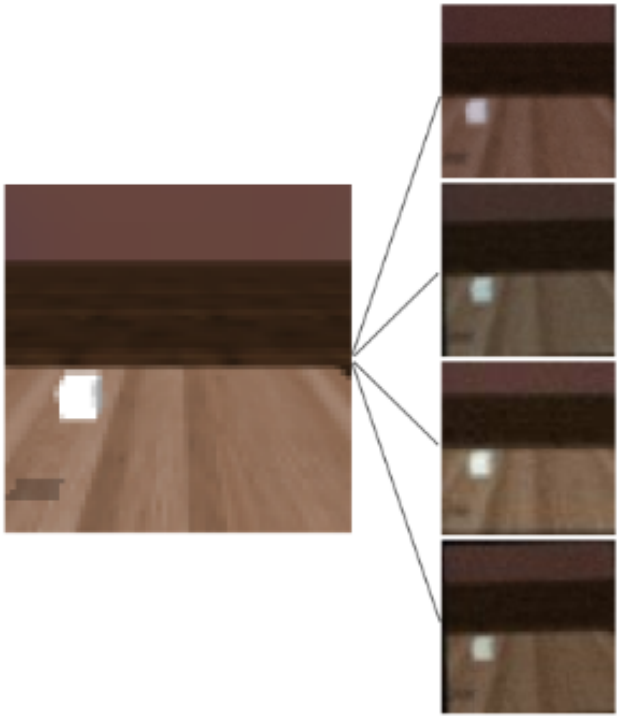

Fig. 9. Demonstrating Domain Augmentations to Simulate Noise and Inaccuracies

V. Results

The full 10-episode evaluation log is given in Appendix A (Table III).

The SAC agent, trained using a multi-view masked autoencoder with domain augmentations over 1,000,000 timesteps, was evaluated across 100 episodes in which the robotic arm must reach a target object and attempt to maintain its position with the target positioned toward the bottom of its claws. The first 10 evaluation episodes were chosen to be representative of the evaluation. The agent achieved a success rate of 90%, completing the task in 9 of 10 episodes. The single failure in Episode 4 was characterized by a minimum Time-of-Flight reading of 4.51, indicating the arm never started meaningfully approaching the target. Among successful episodes, the total reward varied between Episode 5, with a total reward of 429.63 and 7 successful steps (steps where the target is positioned well), and Episode 9, with a total reward of 1180.02 and 79 successful steps, reflecting inconsistencies in holding while at the target. Despite slight variance, the agent consistently achieved first contact with the target within the first third of the episode, under 33 steps.

The full 10-episode evaluation log is given in Appendix A (Table IV).

The SAC agent, trained using a single-view masked autoencoder with domain augmentations over 1,000,000 timesteps, was evaluated across 100 episodes in which the robotic arm must reach a target object and attempt to maintain its position with the target positioned toward the bottom of its claws. The first 10 evaluation episodes were chosen to be representative of the evaluation. Across these episodes, the agent achieved

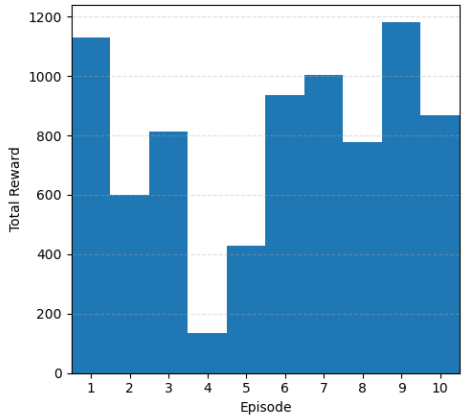

Fig. 10. Total Reward for MVMAE-SAC Agent Trained with Domain Augmentations

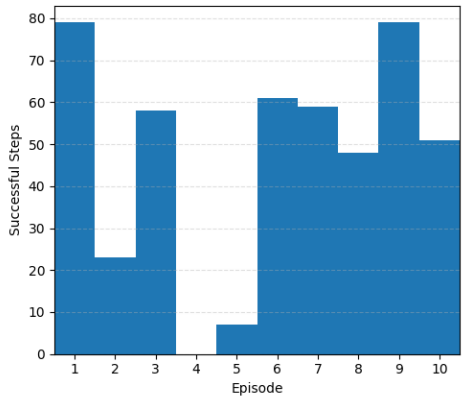

Fig. 11. Number of Successful Steps for MVMAE-SAC Agent Trained with Domain Augmentations

a 70% success rate, completing the task in 7 of 10 episodes. Failures occurred in Episodes 1, 4, and 8, where the minimum Time-of-Flight readings remained relatively high (0.1621, 0.1452, and 0.2407 respectively), indicating the arm never approached the target closely enough to establish contact. Among successful episodes, the total reward varied substantially depending on how long the arm remained positioned near the target. Episode 7 produced the lowest successful reward of 526.38, while Episode 10 achieved the highest reward of 1047.73, reflecting longer periods maintaining a stable position near the target. This variation indicates that while the policy often succeeds at reaching the object, the stability of its final positioning remains somewhat inconsistent across successful episodes. When the task was successful, the agent generally reached the target within the first two thirds of the episode. The earliest first-contact occurred in Episode 3 at step 21, while the latest occurred in Episode 6 at step 68, indicating moderate variation in approach time across successful runs.

1) Benchmark: Single-View Trained Encoder without Domain Augmentations: The full 10-episode evaluation log is given in Appendix A (Table V).

This benchmarking test was conducted to evaluate the impact of using augmentations for effective sim-to-real transfer. Since it is difficult to run 150 evaluation episodes in a real-

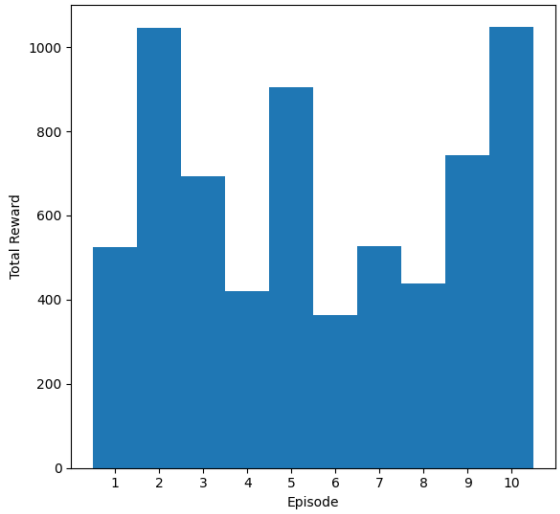

Fig. 12. Total Reward per Episode, Single-View MAE-SAC Evaluation

world environment, the test was performed by training the architecture on simulated environment frames without any augmentation, representing the clean simulation domain. The trained policy was then evaluated on frames that included all of the aforementioned augmentations designed to mimic real-world visual and sensor variations. The SAC agent, trained using a multi-view masked autoencoder without domain augmentations over 1,000,000 timesteps, was evaluated across 100 episodes in which the robotic arm must reach a target object and attempt to maintain its position with the target positioned toward the bottom of its claws. The first 10 evaluation episodes were chosen to be representative of the evaluation. The agent achieved a success rate of 50%, completing the task in 5 of 10 episodes. One failure in Episode 4 was characterized by a minimum Time-of-Flight reading of 2.00, indicating the arm never started meaningfully approaching the target. Among successful episodes, the total reward varied between Episode 2, with a total reward of 567.77 and 4 successful steps, and Episode 5, with a total reward of 739.10 and 5 successful steps, reflecting limited stability while positioned at the target. Despite variance, the agent generally achieved first contact with the target within the first two thirds of the episode, occurring between 25 and 65 steps across successful runs.

TABLE I
COMPARING MULTI-VIEW TO SINGLE-VIEW METHODS OF TRAINING (ALL EPISODES)

Metric	Multi-View MAE-SAC (Augs)	Single-View MAE-SAC (Augs)
Overall Success Rate	89.6%	71.4%
Avg Total Reward	787.24	700.48
Avg Success Steps	46.50	22.40
Avg Minimum ToF Reading	0.0101	0.0757
Avg Action Std Deviation	0.186	0.312

Tables I and II summarize the average evaluation metrics across all episodes. In Table I, which compares the two encoder training methods, the Multi-View MAE-SAC agent achieved an overall success rate of 89.6%, while the Single-View MAE-SAC agent achieved a 71.4% success rate. The Multi-View agent obtained an average total reward of 787.24,

TABLE II
COMPARING SINGLE VIEW MAE TRAINED WITH AND WITHOUT DOMAIN AUGMENTATIONS (ALL EPISODES)

Metric	Single-View MAE-SAC (Augs)	MAE-SAC (Without Augs)
Overall Success Rate	71.4%	56.2%
Avg Total Reward	700.48	529.53
Avg Success Steps	22.40	3.50
Avg Minimum ToF Reading	0.0757	0.1593
Avg Action Std Deviation	0.312	0.411

compared to 700.48 for the Single-View agent. In terms of stability near the target, the Multi-View model maintained correct positioning for an average of 46.50 success steps per episode, whereas the Single-View model averaged 22.40 success steps. The average minimum Time-of-Flight reading was 0.0101 for the Multi-View agent, while the Single-View agent recorded an average minimum ToF reading of 0.0757. The Multi-View agent also recorded a lower average action standard deviation of 0.186, compared to 0.312 for the Single-View agent.

Table II compares the performance of the Single-View MAE-SAC agent when trained with and without domain augmentations. The agent trained with augmentations achieved an overall success rate of 71.4%, compared to 56.2% when trained without augmentations. The augmented model obtained an average total reward of 700.48, whereas the non-augmented model achieved 529.53. In terms of stability near the target, the augmented model maintained correct positioning for an average of 22.40 success steps per episode, while the model trained without augmentations averaged 3.50 success steps. The augmented model also recorded a lower average action standard deviation of 0.312, compared to 0.411 for the model trained without augmentations.

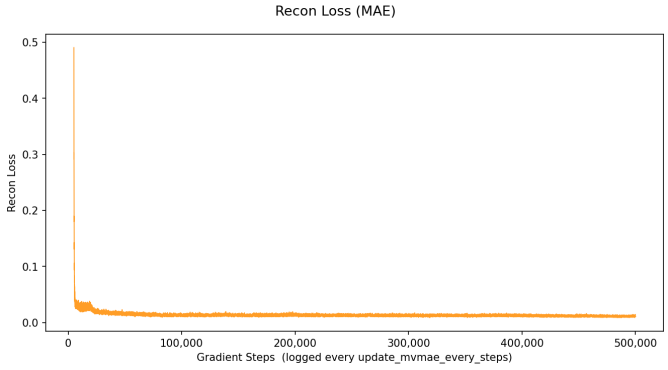

Fig. 13. Decoder Reconstructions Loss Across Training MVMAE-SAC for 1,000,000 Timesteps (500,000 Gradient Steps) with Augmentations

The decoder reconstruction loss decreases sharply in the first 10,000 gradient steps, falling from a peak of approximately 0.49 down to around 0.03, indicating that the MVMAE encoder-decoder rapidly learns to reconstruct masked image patches from the augmented views. Beyond this initial phase, the loss continues to decline very gradually until settling at a healthy 0.01 at around 50,000 gradient steps.

The mean episode reward starts at approximately 500 at the beginning of training and increases steadily across the full 1,000,000 environment frames, reaching values in the range of 2,400 to 2,800 by the end. The overall trend is a consistent upward trajectory, though with notable episode-to-episode fluctuation throughout, particularly between early timesteps where there was significant oscillation. The standard deviation is relatively narrow in the early stages of training, sitting at roughly $\pm$200-300 around 0-100,000 timesteps, which widens progressively as training advances. By the later stages of training, from around 700,000 frames onward, the standard deviation expands considerably, with error bars regularly spanning $\pm$500 and in some cases exceeding $\pm$1000. This widening persists through to the end of training, meaning the variance across evaluation episodes grows alongside the mean reward. Training reward shows a regular, linear progression upward.

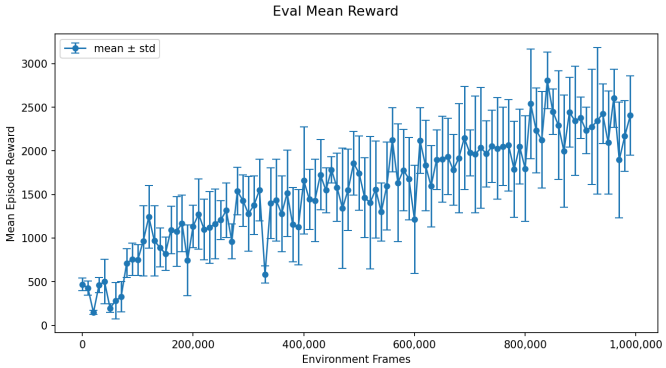

Fig. 14. Mean Episodic Evaluation Reward Across Evaluation Steps Ran Throughout Timesteps during Training for 1,000,000 Timesteps with Domain Augmentations

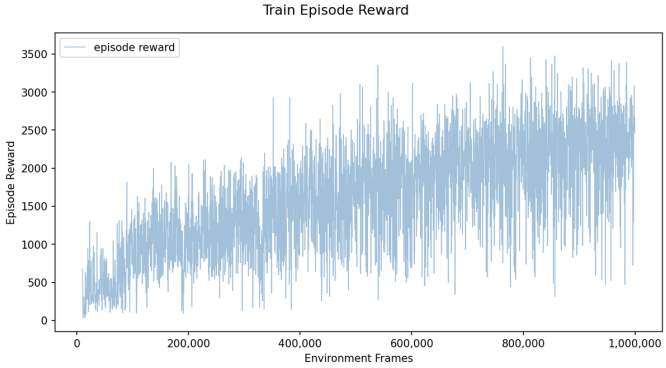

Fig. 15. Mean Episodic Training Reward Across 1,000,000 Training Timesteps with Domain Augmentations

VI. ANALYSIS AND CONCLUSION

The encoder trained with a multi-view architecture (MVMAE), jointly trained with the SAC agent, demonstrated superior performance across every measured metric when compared against both benchmarks. When compared to the single-view augmented baseline, the MVMAE agent achieved an 18.2 percentage point higher success rate of 89.6% compared to 71.4%, and a 12.4% higher average total reward of 787.24 compared to 700.48. It also maintained more than double the average success steps, with 46.50 compared to 22.40, indicating that the policy not only reaches the target

more consistently but is able to maintain stable positioning for significantly longer once contact is established. The average minimum Time-of-Flight reading of 0.0101 compared to 0.0757 further reflects the multi-view agent's ability to position the end effector more precisely at the target. This highlights the impact multi-view training had, allowing the model to ignore background artifacts that could otherwise be unintentionally identified as a target, distracting it from holding position near the real target. The MVMAE model also recorded a lower average action standard deviation of 0.186 compared to 0.312, indicating that the policy produces less erratic joint commands across episodes. This result is particularly meaningful given the hobby-grade servo hardware used in this system, where high-frequency jitter can accelerate mechanical wear and not only place additional strain but also cause inaccuracies. A smoother and more consistent policy therefore translates directly into more reliable and hardware-friendly operation during real-world deployment.

The second benchmark comparison isolates the contribution of the domain randomization pipeline by evaluating an identical single-view architecture trained with and without augmentations. The augmented agent achieved a success rate of 71.4% compared to 56.2% for the non-augmented model, and averaged 22.40 success steps per episode compared to just 3.50, indicating that while the non-augmented policy occasionally reaches the target, it rarely maintains stable positioning once contact is established. This behaviour is consistent with a policy that learned features specific to the clean simulation environment, which do not transfer reliably when evaluated under visual perturbations. The difference in action standard deviation of 0.312 compared to 0.411 further reflects the non-augmented model's less stable behaviour under augmented evaluation conditions, producing more erratic joint commands due to stronger reactions to background noise and artifacts.

Trends in the training metrics for the MVMAE-SAC model were analyzed to further confirm that the model was functioning correctly and to determine whether additional training would be beneficial or if optimal performance had already been reached. Episodic evaluation rewards plateaued at around 800,000 timesteps, which corresponds to approximately 400,000 gradient steps. Based on this observation, it was decided not to continue training further. In addition, sim-to-real was performed with the current trained model successfully, as demonstrated by the following:

The reconstruction loss curve was also analyzed during training. The loss decreases to near zero within the first 100,000 timesteps, which is a promising sign. A reconstruction loss of approximately 0.01 indicates that the encoder embeddings capture a highly representative latent description of the camera frame. Achieving this level of reconstruction accuracy early in training is important, as it provides stable visual features for the critic and actor networks to learn from during policy optimization.

A. Conclusion

The MVMAE-SAC model achieved a 12.4% higher average total reward and a 40.4% lower action standard deviation compared to the single-view augmented benchmark, indicating more stable control behaviour. The model also achieved an 89.6% success rate on the target-reaching task, demonstrating that the proposed control pipeline can reliably position the arm at the target. These results demonstrate that a low-cost, modular manipulator can self-assemble and perform reliable target-reaching tasks using vision-based reinforcement learning, reducing dependence on expensive hardware and enabling faster deployment by first responders with limited robotics training. Future work will focus on physical sim-to-real deployment of the trained policy on the fully assembled robotic arm and further evaluation under real-world operating conditions.

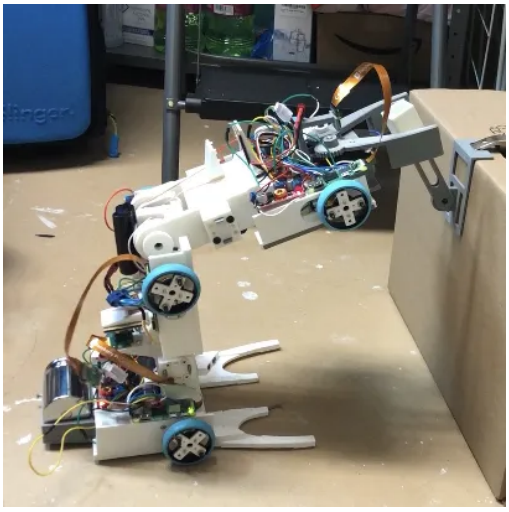

Fig. 16. Model transferred to operate on real robot successfully, demonstrating robot holding at target in relatively noisy environment

APPENDIX
EVALUATION LOGS

TABLE III
EVALUATION METRICS FOR THE MVMAE-SAC AGENT TRAINED OVER 1,000,000 TIMESTEPS WITH DOMAIN AUGMENTATIONS (FIRST 10 OF 100 EVALUATION EPISODES)

Ep	Success	Total Reward	Min ToF Reading	First Success	Success Steps
1	1	1130.35	0.0069	22	79
2	1	601.13	0.0122	31	23
3	1	812.44	0.0091	27	58
4	0	134.06	4.5083	–	0
5	1	429.63	0.0182	19	7
6	1	934.21	0.0055	24	61
7	1	1005.03	0.0036	28	59
8	1	776.88	0.0143	33	48
9	1	1180.02	0.0209	22	79
10	1	868.61	0.0004	23	51

TABLE IV
EVALUATION METRICS FOR THE SINGLE-VIEW MAE-SAC AGENT TRAINED OVER 1,000,000 TIMESTEPS WITH DOMAIN AUGMENTATIONS (FIRST 10 OF 100 EVALUATION EPISODES)

Ep	Success	Total Reward	Min ToF Reading	First Success	Success Steps
1	0	525.04	0.1621	–	0
2	1	1046.15	0.0208	24	75
3	1	693.23	0.0028	21	18
4	0	419.71	0.1452	–	0
5	1	905.64	0.0258	33	36
6	1	664.64	0.0036	68	20
7	1	526.38	0.0953	56	1
8	0	437.86	0.2407	–	0
9	1	743.39	0.0432	30	12
10	1	1047.73	0.0175	37	62

TABLE V
EVALUATION METRICS FOR THE SINGLE-VIEW MAE-SAC AGENT TRAINED OVER 1,000,000 TIMESTEPS WITHOUT DOMAIN AUGMENTATIONS (FIRST 10 OF 100 EVALUATION EPISODES)

Ep	Success	Total Reward	Min ToF Reading	First Success	Success Steps
1	0	464.89	0.2160	–	0
2	1	567.77	0.0492	63	4
3	1	712.19	0.0472	25	14
4	0	396.33	2.0000	–	0
5	1	739.10	0.0565	65	5
6	0	452.43	0.2465	–	0
7	1	733.47	0.0261	37	8
8	0	264.61	0.6387	–	0
9	1	632.14	0.0567	58	4
10	0	332.35	0.1015	–	0

REFERENCES

[1] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. Proceedings of the 35th International Conference on Machine Learning (ICML).

[2] Hafner, D., Lillicrap, T., Ba, J., & Norouzi, M. (2020). Dream to control: Learning behaviors by latent imagination. International Conference on Learning Representations (ICLR).

[3] He, K., Chen, X., Xie, S., Li, Y., Dollár, P., & Girshick, R. (2022). Masked autoencoders are scalable vision learners. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).

[4] Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. International Conference on Learning Representations (ICLR).

[5] Malmir, M., Josifovski, J., Klarmann, N., & Knoll, A. (2026). DiAReL: Reinforcement learning with disturbance awareness for robust sim2real policy transfer in robot control. IEEE Transactions on Control Systems Technology, 34(2), 1037-1043.

[6] Sferrazza, C., Seo, Y., Liu, H., Lee, Y., & Abbeel, P. (2023). The power of the senses: Generalizable manipulation from vision and touch through masked multimodal learning. Proceedings of the 7th Conference on Robot Learning (CoRL).

[7] Shah, K., Crandall, R., Xu, J., Zhou, P., George, M., Bansal, M., & Chellappa, R. (2024). MV2MAE: Multi-view video masked autoencoders. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops.

[8] Srinivas, A., Laskin, M., & Abbeel, P. (2020). CURL: Contrastive unsupervised representations for reinforcement learning. Proceedings of the 37th International Conference on Machine Learning (ICML).

[9] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., & Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

[10] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30.

[11] Xiao, T., Radosavovic, I., Darrell, T., & Malik, J. (2022). Masked visual pre-training for motor control. Proceedings of the 6th Conference on Robot Learning (CoRL).

[12] Yarats, D., Fergus, R., Lazaric, A., & Pinto, L. (2021). Mastering visual continuous control: Improved data-augmented reinforcement learning. International Conference on Learning Representations (ICLR).

[13] Yarats, D., Kostrikov, I., & Fergus, R. (2021). Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. International Conference on Learning Representations (ICLR).