Supplemental Materials for *Egocentric Visual Self-Modeling for Autonomous Robot Dynamics Prediction and Adaptation* IROS 2024 Submission

Yuhang Hu* Boyuan Chen Hod Lipson
Columbia University, Duke University

March 13, 2024

1 Environment Textures

In our experiments, we used a diverse set of environment textures to train and evaluate the egocentric visual self-model (Fig.1). These textures were carefully selected to represent various real-world scenarios and to promote the model's robustness and generalization capabilities.

For the simulation experiments, we used four distinct textures:

1. Rug texture (Fig.1A): This texture features a high-frequency pattern with random small features, resembling a carpet or a rug. The rich visual details in this texture provide ample information for the visual encoder to learn from, enabling the model to capture fine-grained motion cues.

2. Grids texture (Fig.1B): The grid texture consists of a regular pattern of straight lines, forming a checkerboard-like structure. This texture is particularly useful for evaluating the model's ability to perceive and interpret structured geometric patterns.

3. Color dots texture (Fig.1C): This texture comprises a scattered arrangement of circular color dots on a white background. The round features in this texture help assess the model's capacity to handle distinct, localized visual elements and their relative motion.

4. Grass texture (Fig.1D): To simulate outdoor environments, we included a grass texture that mimics the appearance of a natural grassy surface. This texture is crucial for testing the model's performance in more organic and unstructured settings, similar to what a robot might encounter in real-world outdoor applications.

By incorporating this diverse set of environment textures, we try to comprehensively evaluate the egocentric visual self-model's performance across a wide range of visual conditions. The selected textures challenge the model's ability to learn and adapt to different visual patterns, geometries, and surface properties, ultimately contributing to its robustness and generalization in both simulated and real-world environments.

2 Comparison of Prediction Accuracy

Figure2 presents a comparison of the prediction accuracy of our egocentric visual self-model against two baselines: "Only Action" and "IMU Only". The experiments were conducted in a simulated environment, and the plots show the predicted and ground truth values for lateral velocity (first row), forward velocity (second row), and yaw velocity (third row) as the robot moves forward during the test procedure.

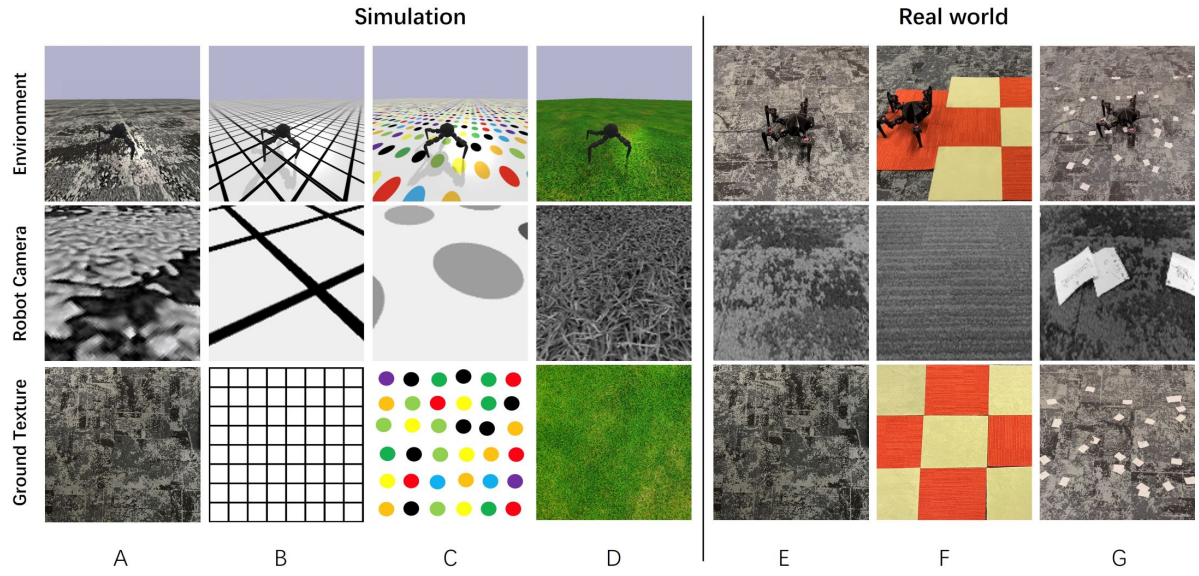

Figure 1: Five environments of our experiments. The left four columns are simulated environments with different terrains: rug (A), grids (B), color dots (C), grass (D). The columns on the right side are terrains in the real-world environment: rugs (E), checkerboard (F), and rugs with scraps of paper (G).

The "Only Action" baseline relies solely on the action commands to predict the robot's future state, without any visual input. The "IMU Only" baseline uses data from an Inertial Measurement Unit (IMU) instead of visual observations to predict the robot's motion. Our method, denoted as "Ours", utilizes the egocentric visual self-model, which takes both visual observations and action commands as input to predict the robot's future state.

The results demonstrate that our method consistently outperforms the two baselines across all tested terrains, including rug, grid, grass, and color dots. The green curves, representing the predictions of our egocentric visual self-model, closely match the red curves, which indicate the ground truth values. This strong alignment suggests that our method is capable of making robust and precise predictions of the robot's motion.

In contrast, the predictions made by the "Only Action" and "IMU Only" baselines exhibit significant deviations from the ground truth values. The "Only Action" baseline, which does not consider any sensory input, fails to accurately predict the robot's motion, highlighting the importance of incorporating sensory information for accurate state estimation. The "IMU Only" baseline, while utilizing sensory data from the IMU, still falls short of the performance achieved by our method, which leverages rich visual information.

The results demonstrated our egocentric visual self-model can effectively capture and integrate visual cues from the environment, enabling more accurate predictions of the robot's motion. These results show the importance of incorporating visual information in robot self-modeling and highlight the effectiveness of our approach in predicting robot motion across diverse terrains.

3 Robot Gait Generator

In order to allow the model to be trained on data that is more representative of the robot locomotion, we design a gait generator to produce the action sequences for our robot to select for locomotion. The gait generator is based on the CPG, used widely in the legged robot system. Each motor action θ_i is generated by a single sinusoidal function with the same period as shown

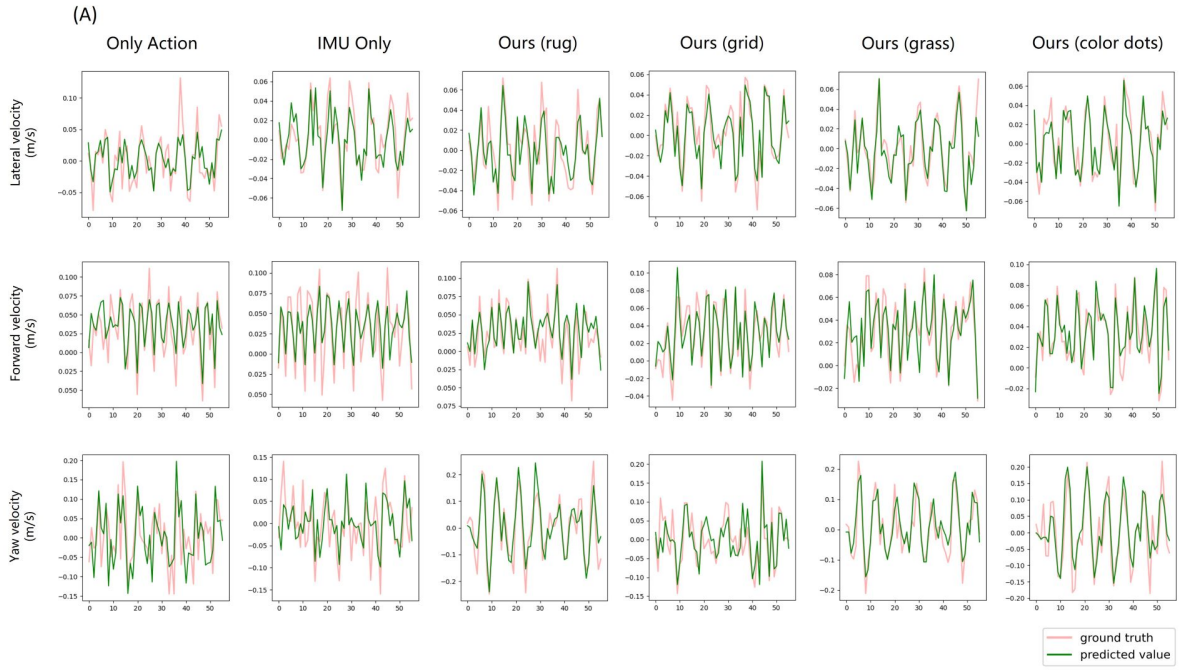

Figure 2: We compare the predictions of the presented method with two other baselines. Each plot shows the predicted and ground truth values when the robot moves forward during the test procedure. Our method can perform robust and relatively precise predictions than other baselines crass all the terrains.

below:

$$\theta_i(t) = a_i \times sin(t/T \times 2\pi + b_i) + c_i. \tag{1}$$

We use a mathematical optimization algorithm Hill Climbing [**?**] to optimize the parameters a_i, b_i, c_i in sinusoidal gait functions. Initially, we randomize the coefficients and test the gait in the simulation using 200 steps. The best gait parameters will be replicated into 20 copies. Among 16 copies, one of the coefficients is initialized randomly, and the other four individuals are initialized entirely from scratch. After each epoch, the rewards of all individuals are computed, and the set of parameters that can reach the highest reward is selected to iterate this process. The formal objective function is:

$$R(\Delta x, \Delta y) = \sum_{t=1}^{n} (100 \times \Delta y - 50 \times |\Delta x|). \tag{2}$$

We finish the optimization by 2-hour parallel computation on a 16 cores CPU core computer and obtain an ideal sinusoidal gait that can make the robot move forward in the simulation. We applied Gaussian noise to our sinusoidal gait to produce new action a_n close to our optimized sinusoidal gait action a_o [**?**]. These new actions can be written as:

$$a_n(i,t) = Gaussian(\mu = a_o(i,t), \sigma) \tag{3}$$

Compared to random motor babbling, using the optimized sinusoidal function with Gaussian noise provides action sequences that are closed to the optimized gait action and avoid the robot submerging in useless movement data, like crawling on the ground or flipping its body. In our tests, $\sigma = 0.2$ can provide good performance. Under this parameter, the robot can reach various states on the ground without falling easily. The gait generator is not only for collecting data for training visual self-model but also sampling action during the testing procedure. Because it can

Table 1: Details of the Randomization

Observation Noise Parameter	Range
Crop range (ratio of the original image)	uniform [0.1, 1]
Rotation Range	uniform $[-\pi, \pi]$
Translation Range	uniform [-3, 3]
Brightness Range	uniform [0.1,10]
Gaussian Blur Kernel Size	uniform [3, 41]
Gaussian Blur Kernel Standard Deviation	uniform [0.1,5]
Motor Torque Range	uniform [1.5,2]
Motor Position Range	$N(0, 0.05)$
Friction Range	uniform [0.5,0.99]

Table 2: Hyperparameters for training visual self-model

Hyperparameters	Value
Optimizer	Adam (Kingma and Ba, 2014)
Initial learning rate	1e-4
Batch size	128
Input size	$5 \times 128 \times 128$.
Output size	6
Hardware configuration	16 cores CPU + 1 GPU

satisfy our purpose of generating a gait similar to the initially optimized gait by adding certain randomness, it can propose new and better gaits to accomplish different tasks by defining an objective function.

4 Reward Function and Score Metrics

Once the egocentric visual self-model is trained, we can use Model Predictive Control to control robots to perform locomotion tasks, each corresponding to distinct reward functions. For instance, to perform forward movement, the robot chooses the action that maximizes the reward function for moving forward. The following formulas provide the basic walking motions that the robot aims to achieve in the experiment:

$$\text{Forward task:} \quad 2 * \delta y - |\delta x| \tag{4}$$

$$\text{Turn left task:} \quad \delta yaw \tag{5}$$

$$\text{Turn right task:} \quad -\delta yaw \tag{6}$$

$$\text{Backward task:} \quad -2\delta y - |\delta x| \tag{7}$$

4.1 Task Characteristics

In both forward and backward tasks, the robot earns scores by moving along the y-axis and loses scores when there is a deviation from the y-axis. For the rotation tasks, the robot is required to rotate its body on the yaw axis.

4.1.1 Score Metrics

To evaluate the performance of the robot in locomotion, the below score metrics are used, with a, b set as 10 in real-world tasks:

$$r_{forward} = a * y - b * |yaw| \tag{8}$$
$$r_{left} = a + b * yaw \tag{9}$$
$$r_{right} = a - b * yaw \tag{10}$$
$$r_{backward} = -a * y - b * |yaw| \tag{11}$$