Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Self-supervised robot self-modeling using a single egocentric camera

Yuhang Hu (yuhang.hu@columbia.edu)
Columbia University https://orcid.org/0000-0003-1964-5784
Boyuan Chen
Columbia University
Hod Lipson
Columbia University https://orcid.org/0000-0003-0769-4618

Article

Keywords:

Posted Date: February 15th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2303274/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: There is **NO** Competing Interest.

Self-supervised robot self-modeling using a single egocentric camera

Yuhang Hu[1], Boyuan Chen[2]*, Hod Lipson[1,3]

[1]Department of Mechanical Engineering, Columbia University, New York, NY, USA
[2]Department of Computer Science, Columbia University, New York, NY, USA
[3]Data Science Institute, Columbia University, New York, NY, USA

The ability of robots to model their own dynamics is key to autonomous planning and learning, as well as for autonomous damage detection and recovery. Traditionally dynamic models are pre-programmed, or learned from external observations and IMU data. Here, we demonstrate for the first time how a task-agnostic dynamic self-model can be learned using only a single first-person-view camera in a self-supervised manner, without any prior knowledge of robot morphology, kinematics, or task. We trained an egocentric visual self-model using random motor babbling on a 12-DoF robot. We then show how the robot can leverage its visual self-model to achieve various locomotion tasks, such as moving forward, backward and turning, all without any additional physical training. The accuracy of the egocentric model exceeds that of a model trained using an IMU. We also show how a robot can automatically detect and recover from damage. We suggest that self-supervised egocentric visual self-modeling could allow complex systems to continuously model themselves without additional sensors and prior knowledge.

*Current address: Mechanical Engineering and Materials Science, Duke University, Durham, NC, USA

Introduction

Most people cannot walk forward much further than a few steps with their eyes closed, even on a flat surface devoid of obstacles. Our proprioceptive senses are, by themselves, insufficient for controlling our locomotion ability. Most of us rely on visual feedback to provide signals for sustained locomotion (*1*). Vision not only helps us navigate and avoid obstacles, but also helps us keep balance and fine-tune our own kinematic self-awareness. If we see patterns on the ground moving backward, our brain can implicitly deduce that we are moving forward. In addition, our brain can learn to predict what will be the effect of certain muscle actions – in other words, our brain uses visual feedback to create and maintain our predictive dynamic self-model.

Our goal in this paper is to explore whether a robot can learn to model its own dynamics using only a series of images, instead of using explicit equations, CAD models, and information from IMUs, joint encoders and extrnal optical trackers. All of these approaches require extensive prior knowledge or infrastructure that may not exist when robots must learn to adapt to new conditions in-situ.

In this paper we report how a robot can learn to predict its future position and orientation, conditioned on motor commands, using only a sequence of five frames from its recent past, as seen from a single onboard camera (Fig. 1). This visual self-model can be used in several ways. For example, a planning algorithm can use the model to test the consequence of various candidate motor commands, and use these predictions to choose the action most suited for a particular task - before executing any of these tasks in reality. Alternatively, a damage detector can use this model to compare predicted motion to actual motion (from visual optometry). If predicted motion deviates from actual motion, then something must have gone wrong. These applications, and other examples, will be demonstrated in this paper.

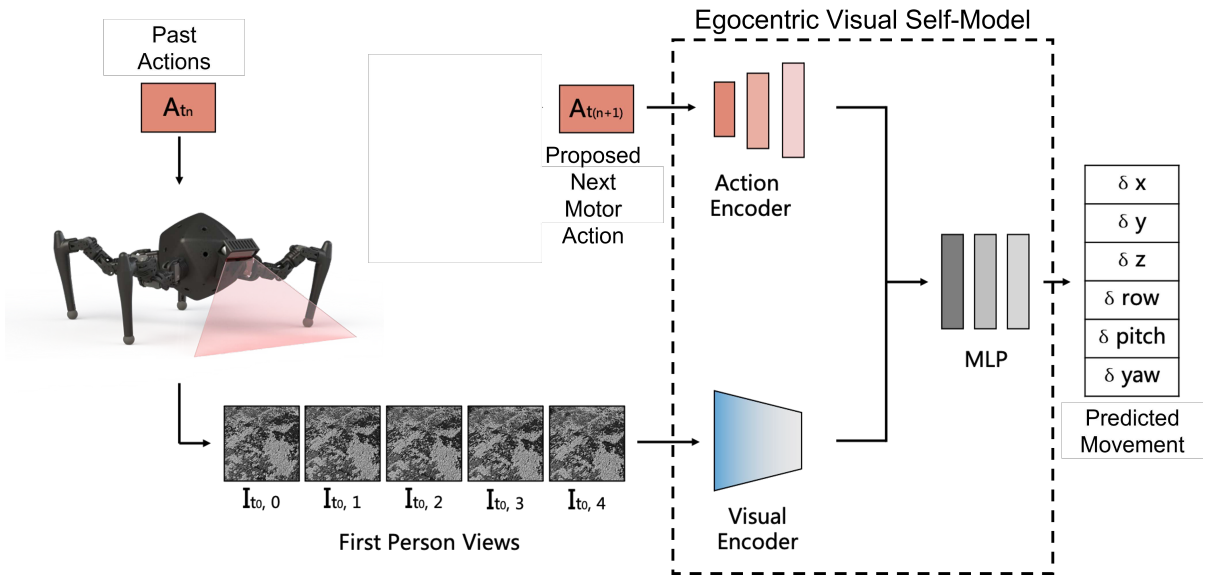


Figure 1: **Egocentric visual self-model:** A robot with an egocentric visual self-model can use a sequence of recent frames taken by its onboard camera, to predict the motion that will result from a proposed next action. This predictive ability can help the robot choose which action to take, before any action is taken in reality, and without any explicit knowledge about its geometry or kinematics.

Self-modeling

Self-modeling is a form of model-predictive-control (MPC) that decouples (deconvolves) a model of the robot (the "self-model") from a model of its environment and task. The key idea is that while the task and environment of a robot may change frequently, the robot itself is relatively constant across different tasks and environments. Therefore, separating out the self-model from the model of everything else, and reusing the self-model for new tasks, can simplify adaptation in situations that involve lifelong learning under variable tasks.

For example, if a tennis player must learn how to play badminton, they may need to learn new game rules, but will not need to relearn how to run across a court or grasp a racquet, because running and grasping are common actions performed by the same body. In this way, self-modeling facilitates transfer learning (*2*).

Importantly, the value of a self-model as compared to model-free learning increases with the complexity of the robot (*3*) and complexity of the task. Therefore, finding efficient ways to create accurate self-models is extremely valuable, and will offer a significant practical advantage as robot complexity continues to increase in the future.

The use of self-models for robot control is not new. Almost all robots today use some form of model predictive control. These forward-models are either analytical or data-driven. Analytical self-models are programmed using equations, or using simulated physics and CAD geometries (*4–6*). Data-driven models are learned using data captured from inertial measurement units (IMUs) and/or external tracking devices such as cameras and optical trackers (*7–16*).

Data-driven self-modeling acquires and maintains a computational representation of the robot morphology, kinematics and dynamics through a machine-learning process that is typically bootstrapped during an initial period of safe "motor-babbling", and potentially continued over the lifetime of the machine.

Our own past work involved learning self-models using data from internal inertial sensors (*17*) or external cameras (*18, 19*). Each of these self-modeling approaches however entailed its own limitation: Analytical models account only for limited dynamics. Data driven models that rely on proprioceptive sensors (e.g. IMUs) drift with time. Self-models that use external observations (e.g. external cameras) are not portable. Our goal here is to create self-models without proprioceptive or external sensors, and without relying on provided equations and existing geometric models.

Visual self-modeling

Visual self modeling is the creation of self-models using visual (camera) sensors only, without additional information. In visual self-modeling, we aim to capture the entire dynamics of the robot using video images alone, without any prior assumptions or knowledge about the robot

geometry, kinematics, or task.

The visual self modeling ability we explore here is similar, but fundamentally different, from "visual odometry". Visual odometry uses cameras to infer position and orientation of a camera for applications such as SLAM, Augmented Reality and Virtual Reality goggle tracking. However, visual odometry does not involve predicting the dynamics of the system harboring the camera, as function of motor commands, and thus, cannot serve as a self-model for motion planning (*20–22*). While we employ visual odometry as part of the learning process, we are ultimately interested in a model capable of predicting dynamics conditioned on motor actions.

Problem statement

The entire observation space of our robot contains only joint commands and the egocentric video feed (Fig. 1). Our key assumption is that visual data contains rich information about the ground, such as surface and texture, which will provide motion clues when viewed as a sequence of frames. Moreover, approaches based on visual modeling take advantage of the recent improvements in computer graphics, where modern simulators can offer image renderings sufficiently realistic that pre-training in simulation and sim-to-real transfer can work well for vision-based policies (*23–25*).

The self-model we introduce here learns the computational dynamics of the robot body in a self-supervised manner. During the modeling process, the self-model takes in a sequence of egocentric images, plus the current and future joint positions, and learns to predict the future states of the robot (Fig. 1). In doing so, the model implicitly learns such properties as mass distribution, geometry, friction, actuation and sensing delays, etc. Once acquired, the self-model can be used by any Model Predictive Control algorithms in real-time to satisfy various objectives.

In each timestep t_n, the commands A_t execute over a fixed period, during which time the

camera takes five consecutive pictures $I_{t,0} - I_{t,4}$. these five images are fed into the Visual Encoder when the movement is complete. When the commands for the next step are generated, they are input into the action encoder. The outputs from two encoders are fused to predict the future robot states.

Our experiments reveal that legged robots with a visual egocentric self-model can predict future position and orientation changes of the robot body more accurately than robots without visual inputs. Our system can also be trained purely in simulation and transfers to the real-world without additional learning. Important to such zero-shot transfer is our domain randomization on both the simulated environments and robot dynamics, as well as data augmentation for neural network training.

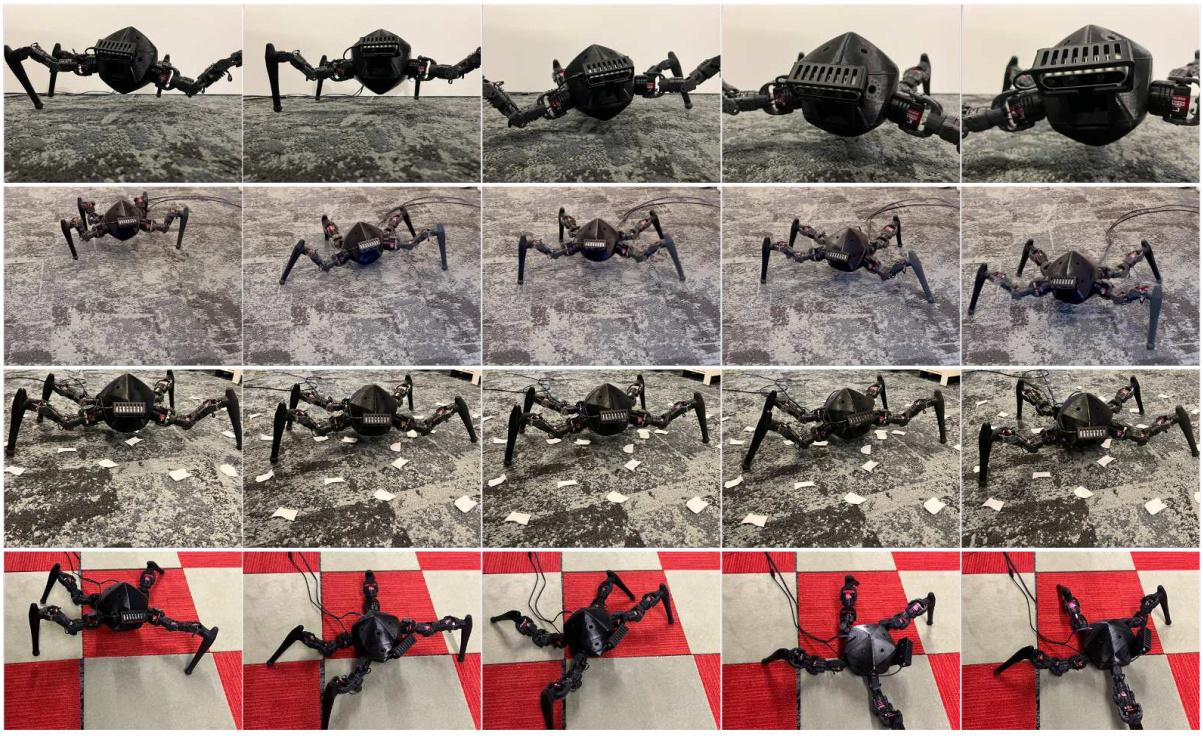


Figure 2: **The Egocentric Visual Model used in practice**. We demonstrate that our legged robot learns to perform basic locomotion tasks using only visual images and motor actions in the real world. The first two rows show the robot crawling on rugs under different light conditions (A) and (B). The robot can also locomote on terrains that were never seen during training (C), (D). All tasks use the same model trained in the simulation with domain randomization, and directly deployed on the robot.

Our key contributions can be summarized as follows: First, We propose a visual egocentric self-model network to effectively fuse sequences of high-dimensional visual observations and motor commands, to predict future robot states. We demonstrate the effectiveness of egocentric visual observations without IMU sensors or fiducial markers for legged robot locomotion on a physical robot platform. We present a domain randomization and data augmentation pipeline that enable effective zero-shot sim-to-real transfer on a legged robot in unseen environments. Finally, We recover a damaged robot to perform a moving forward task by re-training the egocentric visual self-model with real-world data purely from visual perception.

Architecture and usage of the model

The architecture of the egocentric visual-self model is implemented as a hybrid of residual networks, and recurrent neural networks (*26*) as shown in Fig.1. The robot observes two pieces of information: a sequence of frames from the camera, and a set of motor commands.

The visual information comprises five consecutive images with 128×128 size. The motor command comprises a vector of 12 motor angle values. To merge action commands with the images, we designed two encoders responsible for each type of observation: a Visual encoder and an action encode. The visual encoder consisted of 48-convolutional layers (*26,27*), followed by five Long Short-Term Memory (LSTM) modules (*28*), and an action encoder consisting of three fully connected layers with a Leaky Rectified Linear Unit.

The Action Encoder maps the action vectors into some high dimensional latent space through the fully-connected layer networks. The Visual encoder extracts information using a convolu-

tional neural network and leverages LSTM units to process the features and embed the concept of time, allowing the model to learn the temporal dynamics within the sequential pictures (*29*). Then, both outputs are concatenated and passed through five fully connected layers. Each layer is followed by a layer of Leaky ReLU activation function, and the output comprises six numbers representing the robot future change in position and orientation.

To train the model we used actions generated by a CPG sinusoidal gait generator optimized for forward locomotion in simulation using simple Hill Climbing (*30*). While the model will be used later for activities other than forward motion, we found that training the model using actions from a CPG is better than training the model on actions from a purely random motion generator. This suggest that purely random actions are inefficient for self-supervision.

During deployment, the robot proposes candidate motor commands. Each set of motor commands is fused with five recent gray-scale images to form the input data. Since the egocentric visual self-model runs in parallel on the GPU, we can obtain future states of the robot corresponding to multiple sets of candidate actions in a single cycle. This process is like a robot imagining its own future states through different trials. The robot then selects the action most suited for its objective functions. Therefore, a planner needs enough action candidates to allow the model to select the optimal action. The maximum number of motor commands per model run is the batch size on the GPU. We were able to test 100 candidate motor commands in parallel on an NVIDIA Geforce RTX 3090.

On our system, the visual self-model ran at about 40 Hz during our experiments. We pre-trained the egocentric visual self-model in a simulation that included a rug-like texture similar to the real world. Unlike plain ground texture, the rug texture has many micro-features that can give the Visual Encoder plenty of information for optical flow. We modified the imported terrain and texture maps in simulation for each episode to avoid overfitting to a single texture.

The CPG (used during the training phase only) created gaits for moving forward. However,

once the self-model was trained, the robot could use the self-model to accomplish other tasks, such as moving backwards or turning, suggesting that the model can predict consequences of action significantly different from those generated naive forward-moving gait. Therefore, we believe that the egocentric visual self-model can extrapolate somewhat outside the space of actions used in training and hopefully be useful for a broad range of locomotion tasks.

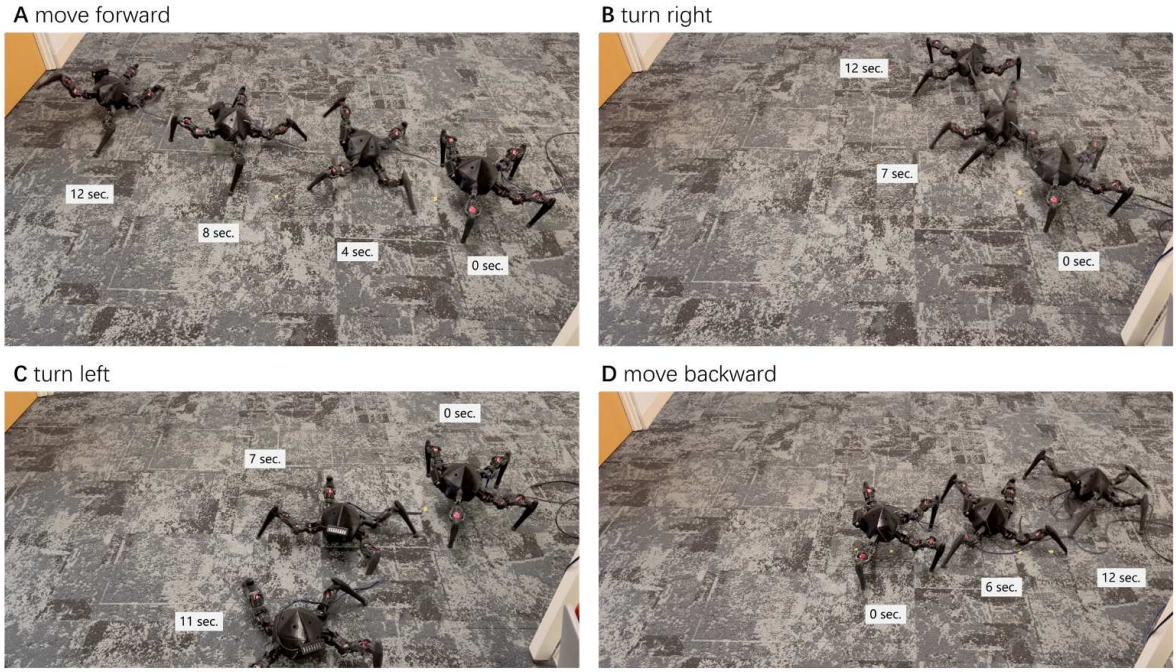


Figure 3: Basic locomotion tasks in the real-world environment. We deployed an egocentric visual self-model on a legged robot. The robot only relies on image sequences from the front camera and action commands to achieve (A) moving forward, (B) turning right, (C) turning left, and even (D) moving backward. It learns the skill from the simulation to anticipate hundreds of possible future states by perceiving the latest visual information and motor commands it will actuate. See S1 Movie for more details.

Results

To evaluate our approach, we first trained an egocentric visual self-model with the information from the front camera, and combined it with motor actions, to predict future robot states and

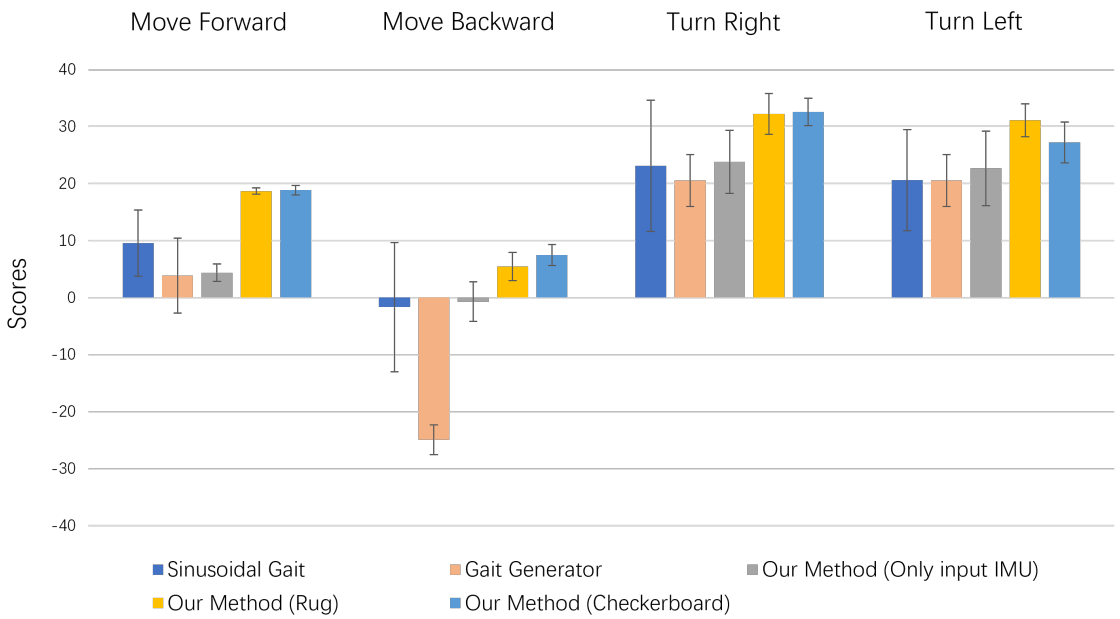


Figure 4: Evaluate the egocentric visual self-model in the real world. We compare our method with four baselines. In each experiment, the robot starts from the same pose, and each episode consists of 56 steps. After each run, we measure the final position and pose to calculate the scores.

perform basic locomotion tasks, as shown in Fig.3 and Movie 1. We then compared the egocentric visual self-model with other baselines using a simple CPG gait generator (*31, 32*) that generated sinusoidal gaits.

Evaluating robustness of the self-model in unseen environments

We performed experiments on an unseen carpet littered with 1x1 cm paper scraps (terrain G in Fig. 8). The paper scraps increase the difficulty of locomotion tasks because they create variable friction, causing the robot to slip when stepping on the paper in Fig. 5A and Fig. 5B. The robot was still able to complete the locomotion task, despite the controller never seeing scraps of paper during training.

To further evaluate our approach, we also processed the input image sequences by modifying exposure, contrast, and adding Gaussian blur. Four different terrains were applied during pre-

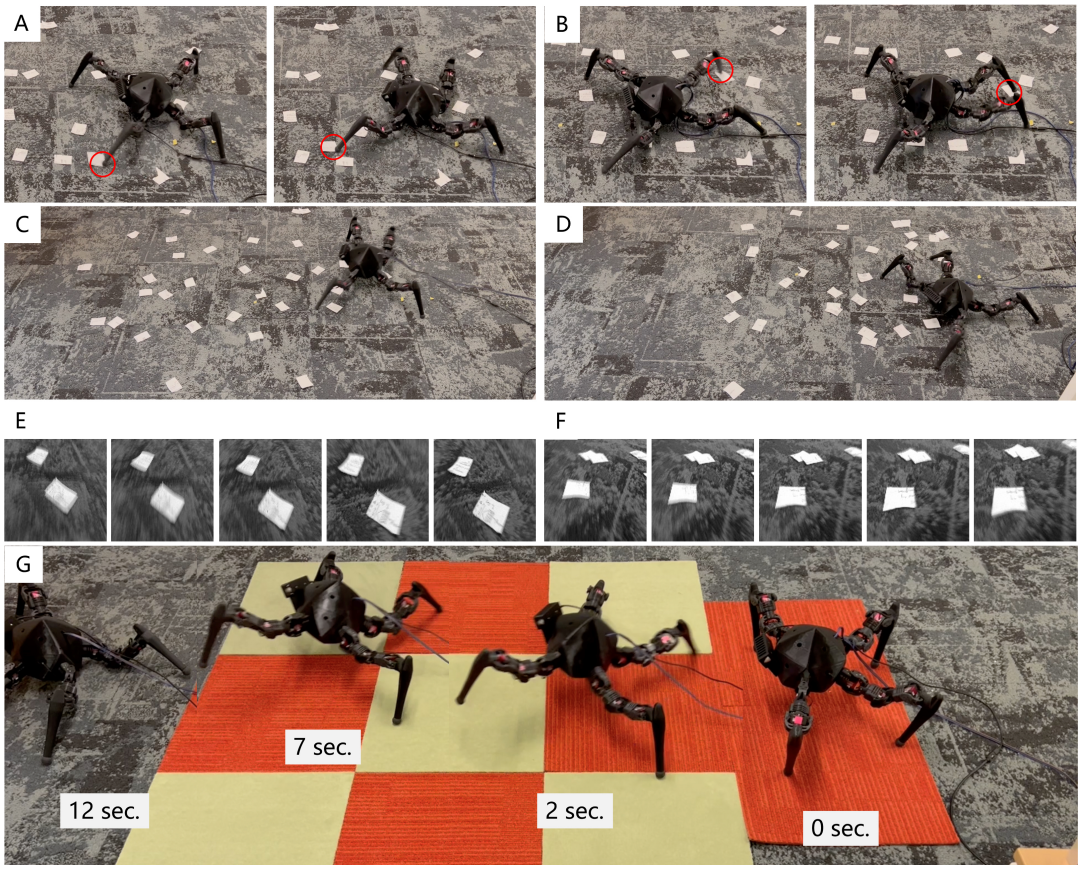


Figure 5: Assessing egocentric visual self-models on unseen terrain. (A, B, C, D and E) We performed the same experiments on carpets with pieces of slippery paper and terrain with checkerboard textures. (C) The robot is moving forward, and its first-view pictures are shown in E. (D) The robot is turning right and its first-view pictures are shown in F. (G) The robot moves forward with an egocentric visual self-model on unseen terrain.

training in simulation. We also tested our robot on a chessboard pattern that was unseen in pre-training. As shown in Fig. 3, the results of our method still outperform all baselines.

Evaluating the contribution of the Visual Encoder

Our method fuses high-dimensional image data and motor action to predict future robot state. We conducted experiments to quantitatively evaluate the effectiveness of the Visual Encoder in the pipeline by comparing it with other baselines.

First, we compared our method with the "Commands only - no vision" baseline. This baseline uses the same approach but replaces the input images with blank frames. The goal of this baseline is to separate the contribution of the images from an open loop self-model that goes from state and motor commands to the next state, without using camera input (e.g. (*19*)). We trained this baseline with the same network architecture. Specifically, we used the same dataset to train the egocentric visual self-model but fed only black images as input. This baseline is supposed to learn the robot's forward-kinematic body model through interaction with the physical environment and have an ability to predict future robot states based on the action produced by the gait generator in open loop. Intuitively, it represents animals or humans learning to walk blindly without visual input. This approach is not expected to perform as accurately as our method, which has visual input to help the model make its predictions. Therefore, comparing this baseline to our method helps highlight the contribution of the visual information.

The second baseline replaces image sequences with explicit IMU data as input, and otherwise remains consistent with our model in the motor action module. We use this baseline to compare the contribution of the high-dimensional visual information over the low-dimensional IMU information that is typically used in robotics. We used high accuracy IMU (HWT905-TTL MPU-9250 9-axis Gyroscope+Angle (XY 0.05° error). In simulation, the IMU data has perfect accuracy.

The results in Fig. 6 demonstrate that leveraging the high-dimensional visual sensor to predict future states can provide better results than only relying on IMU data alone. In Fig. 3 we show the evaluations of locomotion tasks in the real world using both methods. We believe that the high-dimensional observation should conclude the information of IMU data. The visual observation can also provide information that needs to describe robot configuration, kinematics, and dynamics. Thus, our egocentric visual self-model can perform better prediction and locomotion tasks in simulation and real-world environments.

The egocentric visual self-model is designed to predict six numbers representing the velocity of the robot body in x, y, z, roll, pitch, and yaw axis. We execute ten episodes for each task in each environment, and each episode contains 56 steps. The Table in Fig. 6 shows the results of our experiments in the simulation. We test our model with four different terrain textures in Fig. 8. We compare our method to the baselines using mean prediction loss and evaluation metrics for each task. The L2 metric was utilized for the prediction error. The results indicate that our approach is better than any other baselines in four tasks across all terrains.

Autonomous anomaly identification and adaptation

The egocentric visual model can endow the robot with abilities to recognize hardware anomalies, like broken legs or damaged actuators, and recover by adapting to the new body configuration. We conducted anomaly-detection experiments using a different leg module, which has a broken end-link as shown in Fig.7(A-C). Such damage typically happens when subjected to external force impact or fatigue fracture during testing or deployment, and is generally difficult to detect using direct sensors. Recovery usually requires human intervention to determine the cause and shut down the robot for repair.

To detect the anomaly, we first trained a deep visual odometry model using the same data as the egocentric visual self-model. The model takes 7 consecutive pictures captured by the camera between two steps as input and returns the state changes $(\Delta x, \Delta y, \Delta z, \Delta roll, \Delta pitch, \Delta yaw)$ between every two steps. This information is not predictive and does not factor in motor commands. The visual odometry captures the current (not the future) state of the robot. The visual odometry model has the same architecture as our egocentric visual self-model model, consisting of the visual encoder and MLPs, but omitting the motor command encoder.

We deployed the learned visual odometry model directly on the physical robot. As shown in the first two columns of Fig.7D, the robot can recognize that an abnormal situation (leg broken

in this case) may have occurred based on a large discrepancy between predicted and actual motion. The first two rows in Fig.7 E demonstrate the difference when we use the egocentric visual self-model to control the normal robot and the damaged robot to do the same forward motion task. Under normal circumstances, the robot will choose a forward gait according to its predicted future state to achieve forward motion. However, since its right rear leg was broken, following the original gait caused it to deflect to the right. The discrepancy between predicted and actual motion can trigger an alarm. It can also trigger an auto-recovery processes as follows.

In order for the robot to regain its previous ability to accurately predict future states based on visual information and motion commands, the damaged robot needs to re-learn its self-model to match its new, altered body configuration. By running motor babbling in the real environment for about 30 minutes, the robot collected 7,000 steps of data. This data includes the motor commands of the robot and the images captured by the camera. The Visual Odometry (VO) model outputs the change of robot state from the visual information as the ground truth label for updating the visual self-model. Egocentric visual self-models can be re-trained with this new data to successfully and accurately predict future states. Once the robot states predicted by the self model match the actual robot states measured by the Visual odometry, the self-model retraining has been completed. Fig.7D shows how the error becomes lower after updating the model, and the robot can recover to perform forward locomotion in Fig.7E. The experiment's videos are in S3, S4, S5.

Limitations and future work

We proposed an egocentric visual self-model that learns to predict future robot states by combining a sequence of images and motion commands. We demonstrated how the egocentric visual self-model can allow legged robots to perform basic locomotion tasks. The model can use high-dimensional visual observation and actions to infer the motion of the legged robot

body and generalize the information it needs to predict future states.

The main limitation of the presented method is that the robot can only make accurate predictions when the ground is static. The other limitation is that the robot cannot predict a long-term future state.

Humans use much more information from their vision to enable locomotion. We can quickly identify objects as our world coordinate system. We select static objects as reference and ignore moving objects – For example, we use the ground instead of a moving car as the origin. We also use persistent distance landmarks (such as the horizon or a distance building) as long-term visual references.

We expect that similar multi-modal strategies could enhance visual self-modeling for robots too. When properly trained, visual self models may be able to distinguish between reliable static textures and unreliable moving textures. A combination of ground-facing cameras as well as forward-facing cameras can combine short-term and long-term visual planing.

MATERIALS AND METHODS

Traditionally, visual observation is used for high-level control of legged robotic systems, such as planning, localization, and object detection, as we discussed above. The motion controller is a separate module that operates at a higher frequency than the camera. Our goal is to demonstrate the effectiveness of using egocentric visual observations for legged robot motion and propose a way to directly control the robot locomotion using vision, even though the overall controller operates at a lower frequency. An intuitive way to address these problems is to train a network that takes images as input and outputs action commands. However, the main challenges of using RGB images as observations are the high computational cost of visual information and various environmental changes, resulting in incoherent actions and the inability to perform locomotion. We propose a method for training a visual self-model that takes a sequence of

images with a list of commands produced by a prior action generator as inputs and outputs the future state of the robot. Specifically, we use convolutional neural networks and memory-based networks to process sequential visual data and map action commands into a high-dimensional latent space. Finally, we can use the objective function to access different action commands to perform various tasks.

Legged Robot Hardware

We designed a four-leg robot with 12 degrees of freedom from 3D printed and commonly available electronic parts shown in Fig.2. Specifically, each leg contains three Hiwonder LX-224 serial bus servos with a 0.3-degree accuracy and 20 kg·m torque at 7.4 voltage. We used position control to command the actuators from an onboard Raspberry Pi 4 controller. We equipped the robot with a front RGB camera with $87^\circ \times 58^\circ$ field of view and 60 frames per second facing down to the ground by 41 degrees. The onboard controller and the camera are connected to a desktop with one NVIDIA 3090 GPU.

Data Representation

The robot observation includes the five most recent visual images from the front camera. Instead of directly using the camera readings, we crop the input RGB images from $320 \times 240 \times 3$ to $240 \times 240 \times 3$ and resize them to $128 \times 128 \times 3$. Furthermore, we only use the gray-scale image rather than the raw RGB input to reduce the dimension of the visual observation by a large margin. Such a process is critical for real-time control of the physical robot. This is because all program steps are serial. By having a smaller network to process the reduced visual inputs, we can save the cost of the decision-making process during inference time, thus avoiding losing too much information about the robot states while the robot moves constantly. The Movie 2 shows the examples of data collection in the simulation.

In addition to the egocentric visual observations, the robot also takes its next-step action as inputs. Since we use position control for the robot, the action commands are encoded as a vector with 12 numbers representing the angular displacement for each motor. Our egocentric visual self-model aims to predict the robot states from its visual observation and next-step action. However, since we do not rely on any GPS-like systems to localize the robot, we do not have the pose of the robot in the global world coordinate. We therefore define the robot state as the change of the robot position and orientation in the robot coordinate frame known as $\Delta x, \Delta y, \Delta z, \Delta roll, \Delta pitch, \Delta yaw$.

We collect data in the Pybullet [1], a robotics physics simulation. As we presented above, the objective of the visual self-model is to predict the future robot state given the latest sequence of five frames and action commands. The robot needs to have the capability to figure out the motion from the five frames captured by the first-person camera aiming at the ground and predict the influence of the action commands it will take. Initially, we use a random policy that uniformly samples actions in an action space normalized to [-1,1] representing robot joints -90 degrees to 90 degrees and record images and robot state. However, this method is ineffective in reducing the loss of the model during training because the data we collected does not match that of the data required for the robot to achieve locomotion. Besides, the high dimensional observation space combined with action space is too large and even more extensive when we transfer the trained model to the real world using domain randomization.

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 Central Pattern Generator (CPG), used widely in the legged robot system (*32, 33*). Each motor action θ_i is generated by a single sinusoidal function

with the same period as shown 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 (*30*) 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 (*34*) function 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, σ = 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 satisfy our purpose of generating a gait similar to the initially optimized gait by adding certain

Table 1: Details of the Randomization

Observation Noise Parameter	Range
Crop range (ratio of the original image)	uniform [0.1, 1]
Rotation Range	uniform [$-\pi$, π]
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]

randomness, it can propose new and better gaits to accomplish different tasks by defining an objective function.

Environments Details

In this work, we have seven different environment domains, as shown in Fig.8. To reduce the gap between the simulation and the real-world environment, we use the same outer morphology of the main body CAD of our physical robot. Other robot CAD parts are simplified when building our URDF files to speed up the simulation computation. Also, we measure the mass of all the components and set the same number in the file. Besides, we calibrate the number of simulation timestamps in each step by running the same motor trajectories in the real world and the simulation. The trajectories are hand-designed to move each joint from a lower limited position to a limited upper position. We set 60 simulated steps for each action command to ensure the motor commands can be fully executed in the simulation. We collect 100 thousand steps for each texture (A-D) in the simulation to train the visual-self model.

Randomization and Data Augmentation

To perform better results, we randomize three aspects of the simulated environment: observation, action, and friction and use data augmentation during the training procedure.

Observation noise: The rug picture contains many features that our model can learn to utilize, but we only have one rug picture. Therefore, we randomly crop the original image into 1000 images and scale them to the same size. When collecting the data from the simulation, we sample these 1000 cropped images and also rotate and translate them randomly. During the training, the brightness of five observation images is changed arbitrarily. Apart from the brightness, we also blur the images by adding Gaussian noise. The details of noise levels are described in Table 1. Except for reducing the gap between simulation and the real environment, all of these processes are designed to force our model to learn the relationships among the five pictures rather than the patterns within the pictures.

Action noise: The robot is powered by a battery in the physical robot platform. The battery cannot always provide constant voltage due to its property, so motors are not performed flawlessly like in the simulation. During data collection, we add the noise to the robot joint position and torque in the simulation. The specific values of action noise can be found in Table 1.

Friction noise: The robot foot is built with a soft rubber knob, so it should be a relatively high friction coefficient. We randomize the friction coefficient (shown in Table 1) in the simulation when the robot starts a new epoch during data collection.

Training Egocentric Visual Self-model

Before training the model, we normalized the ground-truth value to the same standard deviation since the output values contain positive and negative numbers and different ranges. This preprocess effectively avoids model bias, which may predict specific output values better than others that do not significantly impact the loss. We use an Adam optimizer with a learning rate

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

of 1e-4 at the beginning and would be scaled by 0.1 if the validation loss does not decrease over twenty epochs. The hyperparameters used for training the visual self-model are listed in Table 2. The loss function can be written as:

$$L = MSE(f_p(I_t, A_t) - S_{t+1}). \tag{4}$$

Reward Function and Score Metrics

Once we have a trained model, we can make the robot perform various tasks by defining the reward function. For example, the robot can perform moving forward by selecting the action that can maximize the forward reward function. We define the following reward function for the robot to complete the basic walking motion in the experiment.

$$r_{forward} = 2 * \delta y - |\delta x| \tag{5}$$

$$r_{left} = \delta yaw \tag{6}$$

$$r_{right} = -\delta yaw \tag{7}$$

$$r_{backward} = -2\delta y - |\delta x| \tag{8}$$

In moving forward and backward tasks, the robot obtains scores by moving along the y axis and reduces scores if there is an offset on the yaw axis. The robot needs to rotate its body on

the yaw axis in the rotation tasks. The score metrics to evaluate the locomotion performance of robot in real-world experiments are shown below:

$$r_{forward} = 10 * y - 10 * |yaw| \tag{9}$$

$$r_{left} = 10 + 10 * |yaw| \tag{10}$$

$$r_{right} = 10 - 10 * |yaw| \tag{11}$$

$$r_{backward} = -10 * y - 10 * |yaw| \tag{12}$$

References

1. J. J. Gibson, *The ecological approach to visual perception: classic edition*. Psychology Press, 2014.

2. J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," *Advances in neural information processing systems*, vol. 27, 2014.

3. R. Kwiatkowski, *Deep Self-Modeling for Robotic Systems*. PhD thesis, Columbia University, 2022.

4. E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.

5. E. Coumans and Y. Bai, "2019. pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.

6. N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, pp. 2149–2154 vol.3, 2004.

7. A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," *arXiv preprint arXiv:2107.04034*, 2021.

8. J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control," in *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 1–9, IEEE, 2018.

9. J. Siekmann, K. Green, J. Warila, A. Fern, and J. Hurst, "Blind bipedal stair traversal via sim-to-real reinforcement learning," *arXiv preprint arXiv:2105.08328*, 2021.

10. N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning*, pp. 91–100, PMLR, 2022.

11. H. Miura and I. Shimoyama, "Dynamic walk of a biped," *The International Journal of Robotics Research*, vol. 3, no. 2, pp. 60–74, 1984.

12. M. H. Raibert, "Hopping in legged systems—modeling and simulation for the two-dimensional one-legged case," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 3, pp. 451–463, 1984.

13. H. Geyer, A. Seyfarth, and R. Blickhan, "Positive force feedback in bouncing gaits?," *Proceedings of the Royal Society of London. Series B: Biological Sciences*, vol. 270, no. 1529, pp. 2173–2183, 2003.

14. G. Bledt, M. J. Powell, B. Katz, J. Di Carlo, P. M. Wensing, and S. Kim, "Mit cheetah 3: Design and control of a robust, dynamic quadruped robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2245–2252, IEEE, 2018.

15. M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, *et al.*, "Anymal-a highly mobile and dynamic quadrupedal robot," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 38–44, IEEE, 2016.

16. T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *arXiv preprint arXiv:1812.11103*, 2018.

17. J. Bongard, V. Zykov, and H. Lipson, "Resilient machines through continuous self-modeling," *Science*, vol. 314, no. 5802, pp. 1118–1121, 2006.

18. B. Chen, R. Kwiatkowski, C. Vondrick, and H. Lipson, "Full-body visual self-modeling of robot morphologies," *arXiv preprint arXiv:2111.06389*, 2021.

19. R. Kwiatkowski and H. Lipson, "Task-agnostic self-modeling machines," *Science Robotics*, vol. 4, no. 26, p. eaau9354, 2019.

20. M. He, C. Zhu, Q. Huang, B. Ren, and J. Liu, "A review of monocular visual odometry," *The Visual Computer*, vol. 36, no. 5, pp. 1053–1065, 2020.

21. L. Zhou, G. Huang, Y. Mao, S. Wang, and M. Kaess, "Edplvo: Efficient direct point-line visual odometry," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 7559–7565, IEEE, 2022.

22. Y. Bao, Z. Yang, Y. Pan, and R. Huan, "Semantic-direct visual odometry," *IEEE Robotics and Automation Letters*, 2022.

23. A. Staranowicz and G. L. Mariottini, "A survey and comparison of commercial and open-source robotic simulator software," in *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*, pp. 1–8, 2011.

24. O. M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

25. OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving rubik's cube with a robot hand," *CoRR*, vol. abs/1910.07113, 2019.

26. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

27. K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

28. S. Hochreiter and J. Schmidhuber, "Lstm can solve hard long time lag problems," *Advances in neural information processing systems*, vol. 9, 1996.

29. S. L. Oh, E. Y. Ng, R. San Tan, and U. R. Acharya, "Automated diagnosis of arrhythmia using combination of cnn and lstm techniques with variable length heart beats," *Computers in biology and medicine*, vol. 102, pp. 278–287, 2018.

30. S. Russell and P. Norvig, "Artificial intelligence: a modern approach," 2002.

31. A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: a review," *Neural networks*, vol. 21, no. 4, pp. 642–653, 2008.

32. L. Righetti and A. J. Ijspeert, "Programmable central pattern generators: an application to biped locomotion control," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 1585–1590, IEEE, 2006.

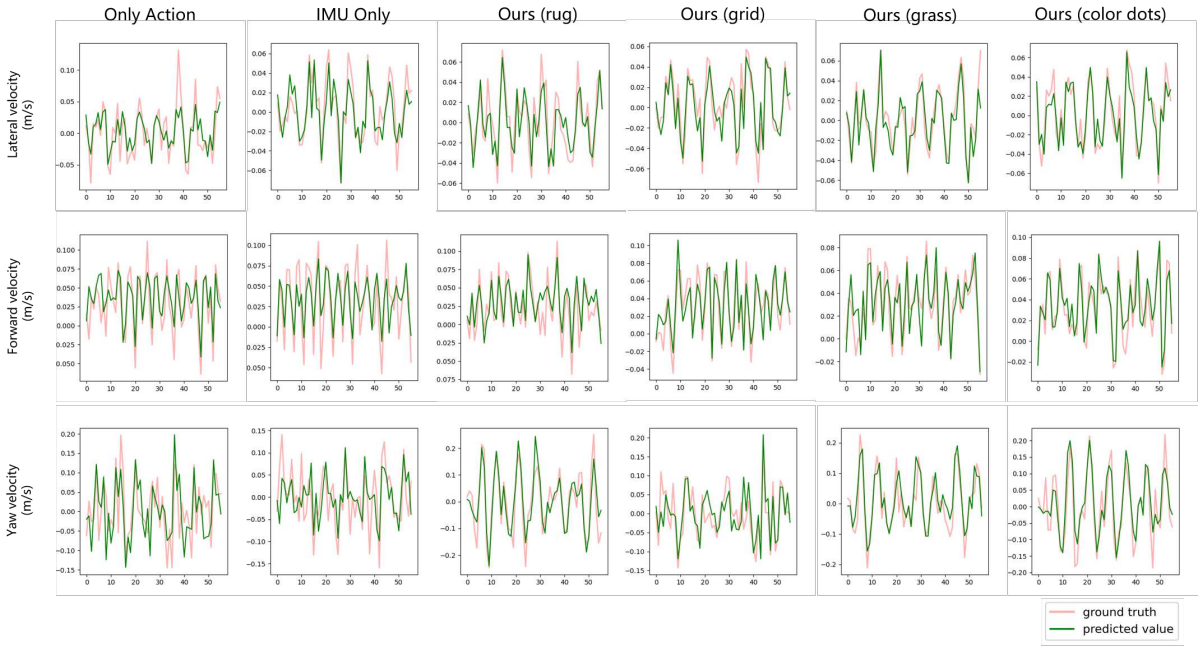
33. Y. Fukuoka, H. Kimura, and A. H. Cohen, "Adaptive dynamic walking of a quadruped robot on irregular terrain based on biological concepts," *The International Journal of Robotics Research*, vol. 22, no. 3-4, pp. 187–202, 2003.

34. C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer school on machine learning*, pp. 63–71, Springer, 2003.

Supplementary materials

Movies S1 to S5. S1: Robot basic locomotion in the real world. S2: Data collection in the simulation. S3: Normal Robot moves forward. S4: Damaged Robot moves forward. S5: Damaged Robot moves forward after autonomous anomaly identification and adaptation.

(A)



(B)

	Action Only		IMU Only		Ours (rug)		Ours (grid)		Ours (color dots)		Ours (grass)	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
Move Forward	3.1E-03	3.9E-03	2.4E-03	3.0E-03	1.9E-03	2.3E-03	1.9E-03	2.3E-03	1.4E-03	2.2E-03	1.4E-03	1.5E-03
Move Backward	1.9E-03	1.9E-03	1.8E-03	1.9E-03	1.4E-03	1.7E-03	1.4E-03	1.7E-03	1.3E-03	1.6E-03	1.3E-03	1.4E-03
Turn Right	2.8E-03	3.8E-03	2.1E-03	1.8E-03	1.6E-03	1.7E-03	1.6E-03	1.7E-03	1.6E-03	1.5E-03	1.6E-03	1.6E-03
Turn Left	2.3E-03	2.4E-03	2.2E-03	2.5E-03	1.5E-03	1.6E-03	1.5E-03	1.6E-03	1.4E-03	1.3E-03	1.3E-03	1.4E-03

Figure 6: We compare the predictions of the presented method with two other baselines. (A) 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. (B) The table shows the quantitative evaluations of our egocentric visual self-model predictions compared with baselines.

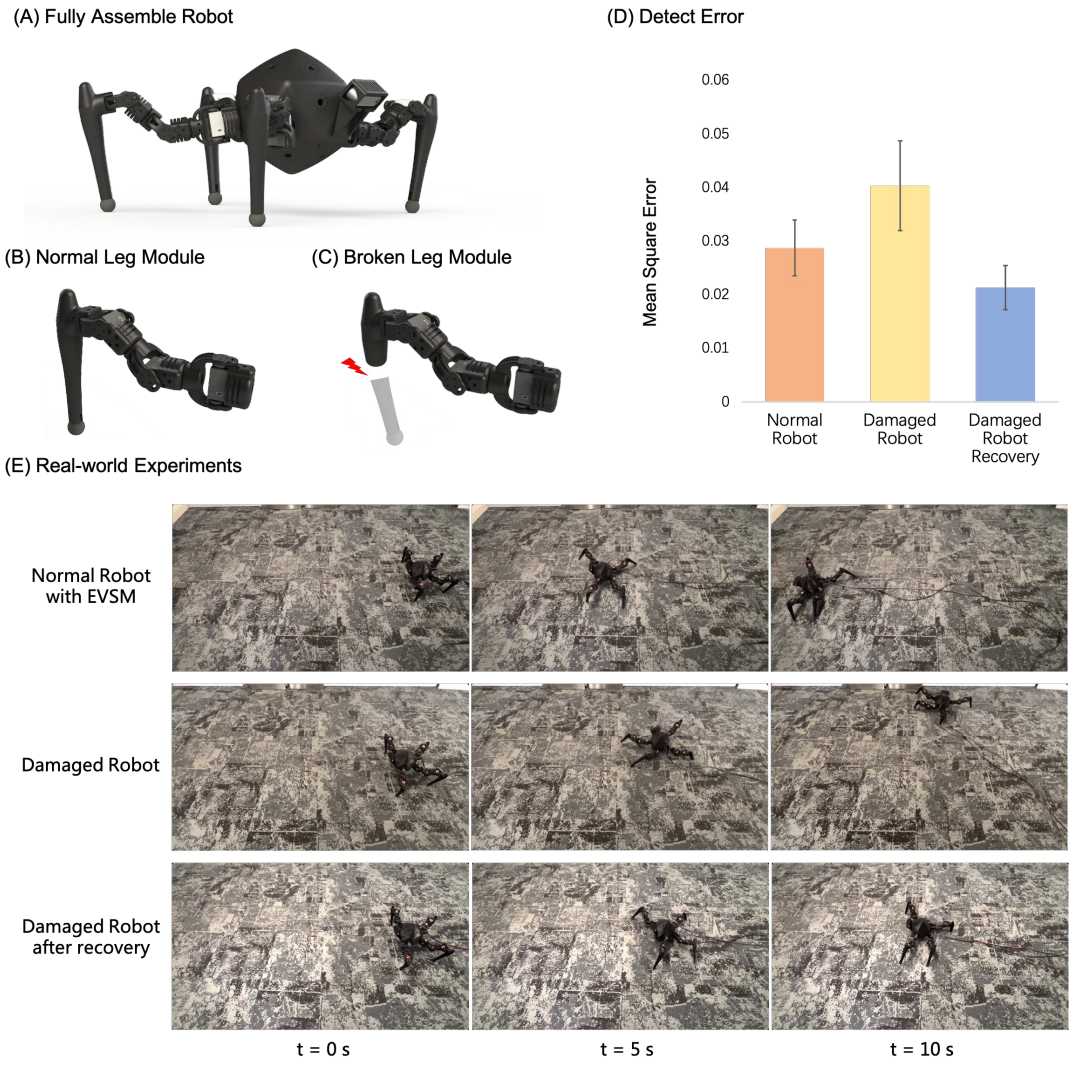


Figure 7: Description of anomalies. (A) Our fully assembled robot has four legs and a camera aimed at the ground. (B) The robot leg module consists of three motors and a rubber ball foot in contact with the ground. (C) To test the resilience ability of the robot, we cut the end-link model in half to act as a damaged robot leg. (D) Error detected by the visual odometry model. (E) Resilience experiments in the real world. The first row shows the normal robot moving forward with the egocentric visual self-model. In the second row, we replaced the right hind leg of the robot with a damaged leg and deployed the same egocentric visual self-model to control the robot. The last row presents using the updated egocentric-visual self-model on the damaged robot to control the robot to move forward.

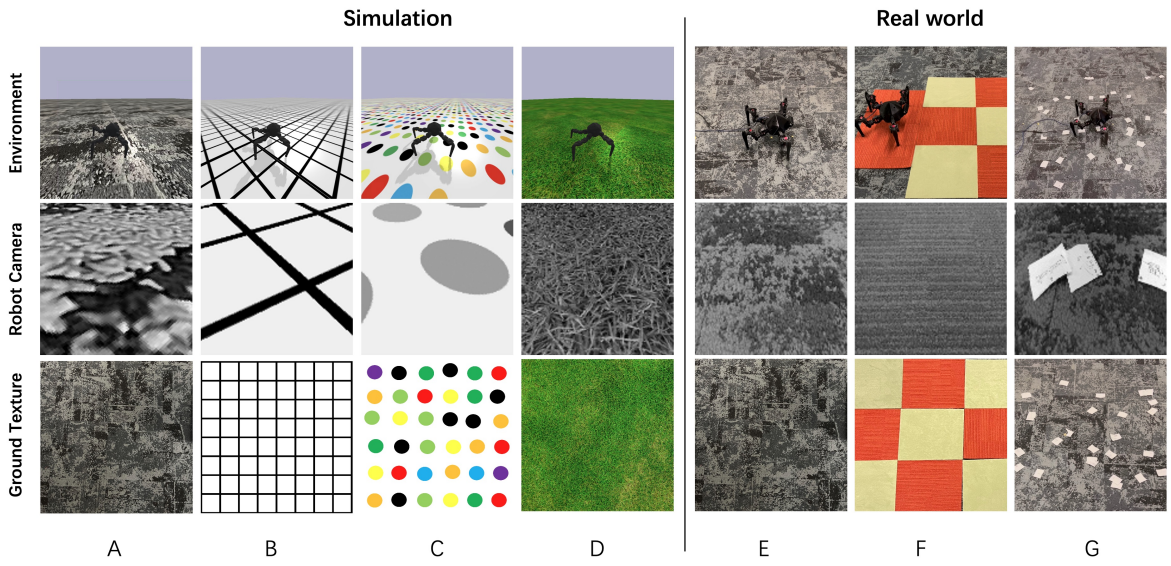


Figure 8: Five environments of our experiments. The left four columns are simulated environments with different terrains: rug (A), grids (B), color dots (C), grass (D). The columns on the right side are terrains in the real-world environment: rugs (E), checkerboard (F), and rugs with scraps of paper (G).

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- EgoVSMSupplementarymaterials.zip