How Far Can a 1-Pixel Camera Go?
Solving Vision Tasks using Photoreceptors and Computationally Designed Visual Morphology

Appendix

Overview

The Appendix provides further discussions, details, and evaluations as outlined below:

- Sec. 1 presents various controlled **analysis experiments**: 1) we probe computational designs for transfer across tasks, 2) we show that the photoreceptor agent can do target detection, 3) we show the effectiveness of the design optimization method, and 4) we evaluate the importance of different design variables.
- Sec. 2 provides additional **visualizations** of different **designs**, including random, intuitive, and computational designs and their corresponding performance.
- Sec. 3 studies whether photoreceptor sensors allow **extracting information** about the **state** of the **world** and whether better-performing designs lead to a more accurate world state estimation.
- Sec. 4 provides **additional results** of using 4×4 PR grids for continuous control tasks in **DMC**.
- Sec. 5 provides a detailed description of our **experimental settings**. We provide details on the control policy training process and design optimization.
- Sec. 6 describes details of the **human survey** used to collect intuitive designs for both navigation and continuous control tasks.

1 Analysis Experiments and Ablations

1.1 Do designs transfer between tasks?

Optimizing and deploying a visual sensor design for each new task can be time- and cost-consuming. One would wish to optimize a design, deploy it on the robot once, and recycle it for different downstream applications without needing to repeat the process. Whether this is possible and to what extent is an *intrinsic property* of the robot's morphology and tasks, which we evaluate with the following experiment. We evaluate the performance of different designs (random, intuitive, and computational) on two pairs of tasks for the same agent morphology. Fig. 1 shows that while there exists a correlation between the performance on different tasks, some designs are over-specialized for a particular task (signified with red). This suggests that some form of regularization is needed during the design optimization to achieve a good trade-off between the performance on different tasks.

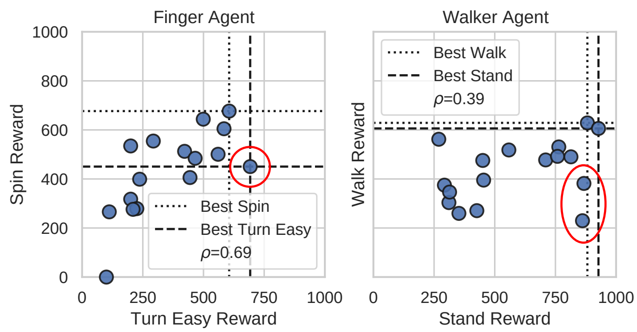

Fig. 1: Do designs transfer across tasks? We consider two pairs of tasks for two agent morphologies in DMC: `(Turn Easy, Spin)` for `Finger` (left) and `(Stand, Walk)` for `Walker` (right). Each point on a plot represents one design for the corresponding agent, and the x and y axes show the performance on two corresponding tasks. ρ is the Spearman's rank correlation. In both cases, we find that a correlation exists in general, but some designs might still be primarily tuned for one task (shown in red).

1.2 Photoreceptor-based agents can do Target Detection

In the `TargetNav` task, to confirm that photoreceptors can perform target detection, i.e., identify the green (visible) sphere and move towards it, we test the behavior and performance of the trained PR agent with a transparent (invisible) sphere as the target instead of the green sphere.

Fig. 2 shows trajectory visualizations comparing the two settings: one with a green target sphere and the other with a transparent sphere in otherwise identical episodes. We see that initially, in both cases, the PR agent follows the same trajectory. In the episode with the green target, the PR agent is able to recognize it and move towards it, while in the episode with the transparent target, the agent does not see it (as expected) and continues searching for it. For a quantitative comparison, to demonstrate that the PR agent is indeed performing target detection and moving towards it to achieve success, rather than only conducting efficient exploration, we compare the agent's success rate in both target settings. The average performance values in Fig. 2 shows that the PR agent is indeed performing target detection, resulting in a much higher success rate.

1.3 Does the task affect the computationally obtained design?

Fig. 3-left shows that even with the same initial random design, the proposed design optimization method converges to different designs for different tasks, namely, `PointGoalNav` and `TargetNav`.

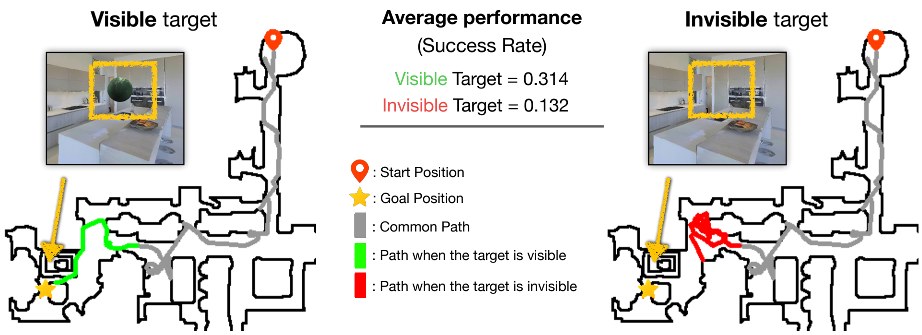

Fig. 2: The photoreceptor agent successfully navigates to the green target sphere when it is visible, but continues searching when the sphere is transparent. Furthermore, the average performance of photoreceptor agents with a visible target is significantly higher than with an invisible target. The successful navigation to a visible sphere, contrasted with the continued search behavior in the case of an invisible sphere, along with the performance disparity between evaluations with visible and invisible targets, demonstrates that the photoreceptor agent actively identifies the target sphere to navigate to it rather than accidentally encountering it.

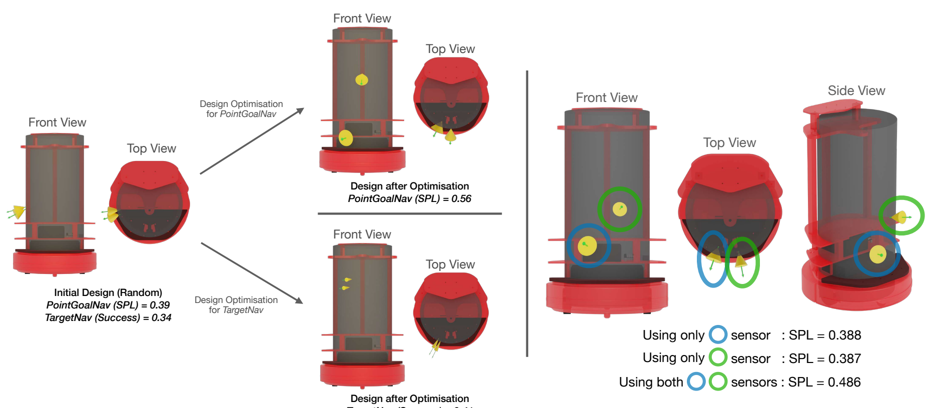

Fig. 3: *Left:* Computational design obtained on `PointGoalNav` and `TargetNav` staring from the same initial random design. The proposed design optimization method produces different designs for the navigation tasks while improving performance on the specific task. Each performance value (Success for `TargetNav` or SPL for `PointGoalNav`) is obtained by training a control policy with the design fixed as shown in the images. ***Right:* Ablation of sensor grids shows that the proposed design optimization method uses available photoreceptor sensors well.** In `PointGoalNav`, designs using only one of the two sensor grids from the computational design perform significantly worse than when using both sensor grids. This shows that the design optimization method places the sensor grids so that the information from both is complementary.

1.4 Design optimisation method (computational design) uses available sensors well

We run a design ablation to show that the proposed design optimization method is optimizing the placement of sensors to maximize performance. We choose the simplest setting of K=2 grids of 4×4 photoreceptors in the `PointGoalNavigation` setting. From the computational design with K=2, we create two designs of K=1 by picking one of the two sensor grids in the computational design. In Fig. 3-right, we show a comparison between the original computational design and the ablated designs, showing that the design optimization utilizes the placement of the additional sensor grid effectively as neither of the two sensor grids alone performs well but together the performance is significantly boosted.

1.5 Comparing importance of different design space variables

We run additional experiments in order to compare the importance of the different design variables in the design space defined in the main manuscript, i.e., $[x_i, y_i, z_i, \mathrm{yaw}_i, \mathrm{pitch}_i, \mathrm{fov}_i]$. For measuring the importance of a specific design variable, x_i for example, starting from the computational design, for all PR grids, we set all design variables except x_i to their initial values (before optimization). This comparison between different design axes is shown in Fig. 4-left for K=2 grids of 8×8 PRs in `PointGoalNavigation`, which shows that the height y_i and the pitch pitch_i design variables are the most important. We also show visualizations for each of these design change in Fig. 4-right.

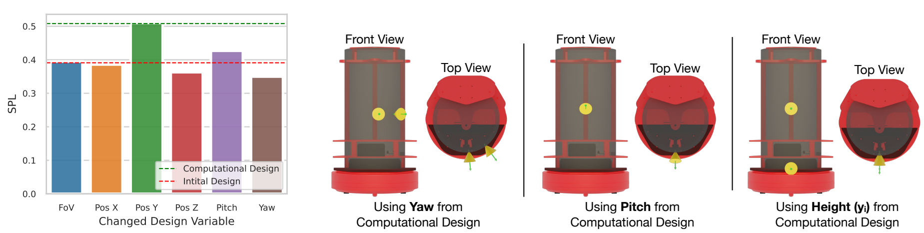

Fig. 4: ***Left:*** **Comparison of usefulness of different design axes.** We evaluate the performance of each design created by starting from the computational design, and for all PR grids, setting all but one design variables to their initial values (before optimization). ***Right:*** **Visualization of designs comparing importance of different design axes.** Each design is created by starting from the computational design, and for all PR grids, we set all design variables except one to their initial values (before optimization).

1.6 Distorting the computational design to analyse the success of design optimisation

For probing the design optimization landscape, we create designs by interpolating between the computational and initial designs using the following formula:

$$\theta_{interpolated} = (1-\alpha)\times\theta_C + \alpha\times\theta_I \quad (1)$$

where θ_C is the computational design and θ_I is the initial design. We choose an exponentially increasing distance from the computational design for the interpolation, i.e., $\alpha \in \{0.05, 0.1, 0.2, 0.4, 0.8\}$ and train control policies for the obtained designs. The performance obtained for each such design (see Fig. 5), shows that designs farther away from the computational designs perform much worse.

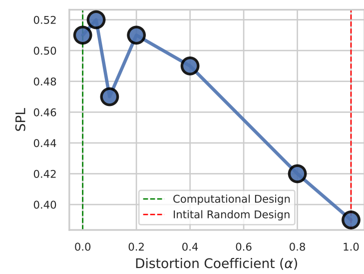

Fig. 5: Designs farther away from the computational design perform worse. We conduct a design distortion experiment in the `PointGoalNav` task where we compute interpolated designs between the computational and initial designs using the formula : $\theta_{interpolated} = (1-\alpha)\times\theta_{\mathrm{C}} + \alpha\times\theta_{\mathrm{I}}$, choosing an exponentially increasing α.

2 Design Visualizations

In this section, we provide additional visualizations of the designs we obtained through computational optimization, intuitive survey or random sampling and their respective performance on the corresponding task.

2.1 Computational vs Random Design Visualisations

In Fig. 6 and Fig. 7, we show the initial random design and the corresponding computational design obtained using the proposed design optimization method for DeepMind-Control and Fig. 8 shows the same for the `PointGoalNav` task. The figures also show the improved reward corresponding to the computational design.

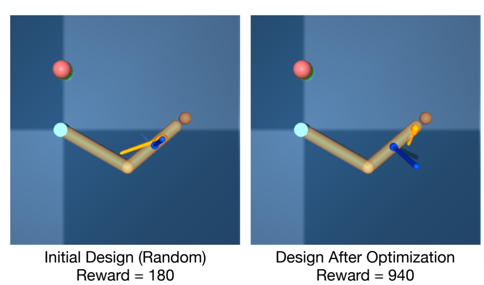

Fig. 6: Visualisation and performance comparison on `Reacher-Hard` task of initial random and computationally (through optimisation) obtained designs.

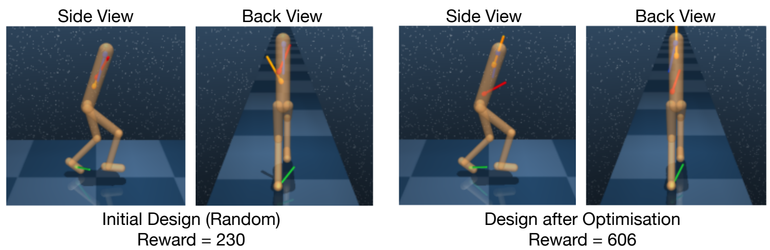

Fig. 7: Visualisation and performance comparison on `Walker-Walk` task of initial random and computationally (through optimisation) obtained designs.

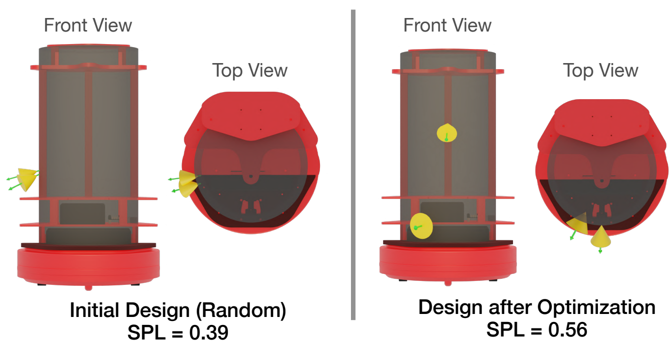

Fig. 8: Visualisation and performance comparison on `PointGoalNav` task of initial random and computationally (through optimisation) obtained designs.

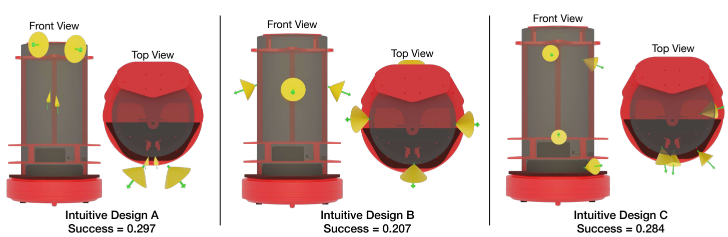

Fig. 9: Visualisation and performance comparison on `TargetNav` task of different intuitive designs collected using the human study survey describe in Sec. 6.

2.2 Intuitive Design Visualisations

Fig. 9, shows visualizations for some of the intuitive designs collected using the survey described in Sec. 6 and their corresponding performance on the `TargetNav` task. This shows that the variance in the performance of intuitive designs is high as well.

3 Can photoreceptors extract information about the world state?

In Sec. 4.2 of the main paper, we demonstrated that an agent equipped with only a few photoreceptors can perform well in solving active vision-based tasks. One would expect such a PR agent to be able to extract useful information about the state of the world using its visual sensors. In this section, we explore whether photoreceptors can extract information about the state of the world and self, and whether better-performing designs extract state information more accurately.

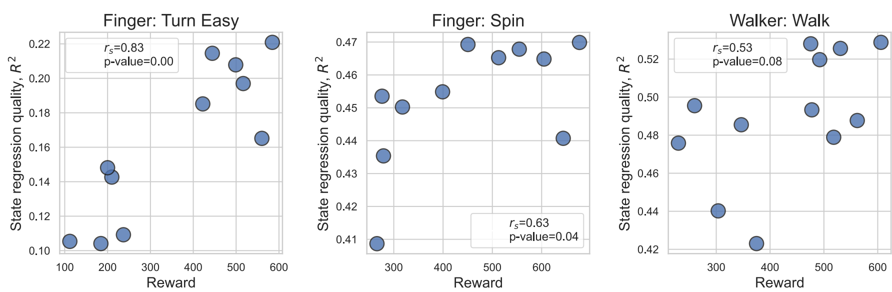

Fig. 10: The quality of the state estimation vs. the reward for different photoreceptor designs. For each design of the photoreceptor sensors, y-axis shows the quality of the state regression from the visual signal provided by this design as measured by the coefficient of determination R^2 averaged along all available state variables on a test set (see Sec. 3 for more details), and x-axis shows the reward achieved by the agent using this design. We use different designs (random, computational, and intuitive) for 1x1 photoreceptors with $K \in \{2, 4\}$. First, we find that $R^2 > 0$ in all cases, meaning that there is useful information about the state contained in the signal of the photoreceptor sensors. More interestingly, we find evidence for the correlation between state regression quality and reward. *This suggests that the quality of the state regression can be a good proxy for finding a well-performing design in terms of the reward on the active task.*

We consider three tasks from the DMC Suite: `Finger: Spin`, and `Finger: Turn Easy` and `Walker: Walk`. For each task, we collect rollouts using the best-performing policy available. At each step, we collect the default state information provided by the DMC benchmark. For example, for the `Walker: Walk` task, this includes the height of the body, orientations, and velocities of each body part. These state values are the default variables used as input by the state-based control algorithms and, therefore, provide a sufficient description of the world and agent states.

In addition to the state information, we collect visual sensory data for different designs (random, computational, and intuitive), achieving different reward values. Then, for each design, we regress the state values from the visual sensory data using the same backbone as for the policy network (we train it from scratch). We use 80000 timestamps for training and 20000 for testing (test time stamps come from different episodes). For each state variable, we measure the coefficient of determination R^2 on the test set and average it over all state dimensions, representing the overall quality of the state estimation from the photoreceptor sensors with the corresponding design.

Fig. 10 shows the quality of the state estimation versus the reward for each design (we use 1x1 photoreceptors with $K \in \{2, 4\}$). First, we find that the R^2 is greater than zero, which means that it is possible to extract more information about the state than the overall mean value (since $R^2 = 0$ corresponds to a mean prediction).

We also find evidence for the correlation between the quality of the state regression and the performance of the agent with the corresponding design. This suggests the quality of the state regression can be a good proxy for a design optimization method. This is useful because this proxy represents a supervised learning task for which a design optimization can be easier to perform than directly optimizing the performance on the active reinforcement learning task.

4 Additional Results for DMC

4.1 Using the Grids of 4x4 Photoreceptors

In addition to the results in Fig. 4 for using 1x1 photoreceptors, we explore whether using a grid of 4x4 photoreceptors further improves the performance of the PR agents. Fig. 11 presents the results on the four most difficult tasks (i.e., where neither agent achieved high performance close to the optimal reward of 1000).

4.2 Design Optimization Results

Fig. 12 shows design optimization results for DMC tasks.

5 Experimental Details

5.1 PointGoal Navigation Setting

In `PointGoalNav`, the agent is randomly initialized in an environment and asked to navigate to a target point given relative to the start state. The episode ends

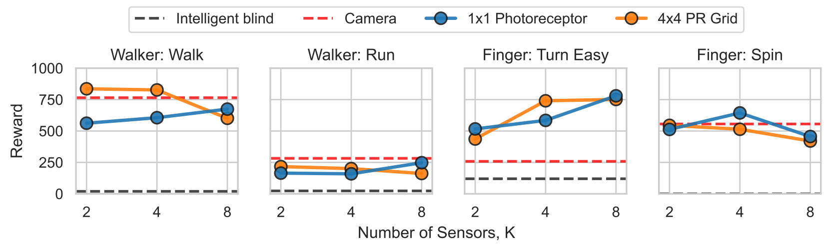

Fig. 11: Additional results for DMC using grids of 4x4 photoreceptors. For the four most difficult tasks (where neither agent achieves near-optimal performance, i.e., the reward of 1000), we explored using more sensors by arranging PRs in the 4x4 grids similar to the navigation tasks (see Sec. 3.1 of the main papers for more details). We find that it considerably improves the performance for the `Walker: Walk` task and either does not improve or even deteriorates the performance on other tasks.

if the agent calls the stop action. It succeeds in the episode if it stops within a 0.2-meter radius of the target point.

Observation Space. The agent has access to an idealized `GPS+Compass` input that provides its current position and rotation *relative* to the starting state. It also receives the relative position of the target point. In addition, the agent observes egocentric RGB views through its photoreceptors.

Action Space. The agent can execute 4 actions: `move_forward` (0.25m), `turn_left` (30°), `turn_right` (30°), and `stop`.

Reward. At every timestep t, the agent at state s_t has the geodesic (shortest path) distance d_t to the target. It applies an action a_t and transits to the next state s_{t+1} whose geodesic distance to the goal is d_{t+1}. It receives a reward r_t in the form of

$$r_t = \begin{cases} 2.5 * \text{Success} & \text{if } a_t \text{ is } \texttt{stop} \\ d_t - d_{t+1} - c^{\text{slack}} & \text{otherwise} \end{cases}$$

where $d_t - d_{t+1}$ is a dense reward for progressing towards the target position and $c^{\text{slack}} = 0.003$ is the slack reward encouraging shorter episodes.

Dataset. In `PointGoalNav`, we use the Matterport3D dataset [2] for training and testing. The training data (`train` split) contains 61 scenes with around 80k episodes per scene. The testing data (`test` split) contains 18 scenes *unseen* during training with 56 episodes per scene (in total 1008 episodes).

Training Process. We train our navigation agents with Proximal Policy Optimization (PPO) [6] which optimizes the objective

$$L(\omega) = \mathbb{E}[\min(r_t(\omega)\hat{A}_t, \text{clip}(r_t(\omega), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)]$$

where ω parameterizes the control policy, $r_t(\omega)$ is the probability ratio between the current policy and the rollout policy, and $\hat{A}_t$ is the estimate of the advantage function through Generalized Advantage Estimation (GAE) [5]. In practice, we adopt the training mechanism in Decentralized Distributed PPO (DD-PPO) [9]

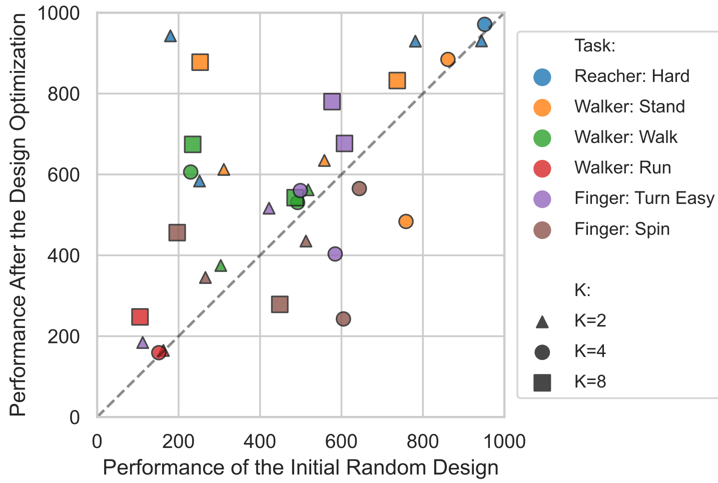

Fig. 12: Design optimization results for DMC showing results for each K. Difference points with the same color and shape indicate different random initialization.

to accelerate the training. We train the agent on 4 A100 or V100 GPUs while each GPU has 30 parallel environments collecting 64 steps of experience (simulation steps) per environment. We call collecting the rollouts above once as a *rollout collection step* ($\approx$ 7.6k simulation steps). With the collected rollouts, we perform 4 epochs of PPO update with 1 mini-batch per epoch. We use the Adam [4] optimizer with an initial learning rate of 2.5×10^{-4}. We set the clipping parameter ϵ to 0.2, the discount factor γ to 0.99, the GAE hyperparameter λ to 0.95. We train the agent for around 230 million (M) simulation steps.

Evaluation Process. We evaluate our agent based on the Success weighted by Path Length (SPL) metric [1]. One episode is counted as success only when the agent takes the `stop` action within 0.2 meters of the goal position within 500 steps. The SPL reported is the average across all episodes in the `test` split.

5.2 Target Navigation Setting

In `TargetNav`, the agent is spawned at a random ground location in an environment and needs to navigate to a target green sphere placed randomly in the scene. The radius of the target sphere is 0.5 m. The agent succeeds in the episode if it enters a circle of 0.8-meter radius around the target sphere center.

Observation Space. The agent receives its current position and rotation *relative* to the starting point and orientation from an idealized `GPS+Compass` sensor. Besides, the agent also observes egocentric RGB views through its photoreceptors. Compared to `PointGoalNav`, the agent does not receive the target position information. Thus, `TargetNav` focuses more on evaluating the agent's ability to explore the environment.

Action Space. The agent has access to 3 actions: `move_forward` (0.25m), `turn_left` (30°), and `turn_right` (30°).

Dataset. In `TargetNav`, we construct our training and testing data from the `PointGoalNav` dataset in Matterport3D scenes [2]. We randomly sample 10 scenes from the `train` split of the `PointGoalNav` setting. We use the same 18 test scenes as in the `test` split of the `PointGoalNav` dataset. We use all the episodes of the `PointGoalNav` dataset for the scenes chosen above. For each episode, we add the green sphere at a height of 1.5 meters above the ground at the goal position.

Reward and Training Process. `TargetNav` uses the same reward design and training process as in the `PointGoalNav` setting described in Sec. 5.1.

Evaluation Process. In `TargetNav`, we evaluate our agent based on Success. One episode is successful only when the agent enters the circle of 0.8-meter radius around the target sphere center within 1500 steps (we restrict the maximum number of steps to 500 during training). We do not require the agent to call the `stop` action because we want to focus more on evaluating the PR agent's exploration ability using its onboard PRs. The Success reported is the average across all episodes in the testing data.

5.3 Continuous Control in DeepMind Control Suite

We use six continuous control tasks from the DeepMind Control Suite [7] from the following three domains:

– `Reacher.` We use the difficulty level `Hard`, which requires controlling the two-legged actuator to reach the target ball with the tip of the actuator.
– `Walker` requires controlling a planar walker. The `Stand` task requires keeping the torso upright at some minimal height. `Walk` and `Run` tasks require, in addition, to have a specific forward velocity.
– `Finger` requires controlling a simple manipulator to manipulate an unactuated spinner. In the `Spin` task, the manipulator needs to spin the spinner with a specific angular velocity. In the `Turn Easy` task, one tip the of spinner needs to align with the target position specified visually.

We refer the reader to the original work [7] for a more detailed description of action space and reward definition. For the `Reacher: Hard` task, we added another green target object inside the original one. We do this because the MuJoCo renderer does not render the target ball (or any object) when the camera is inside it, and it would be impossible to realize that the camera is inside the target ball. Therefore, we add a smaller object inside the target, which gets rendered even when the camera is inside the target and provides a visual cue for success. For all tasks, we use a common practice and repeat the same action twice (and four times for the `Finger: Turn Easy`).

Observation Space. The agent only receives egocentric views from its onboard photoreceptors. Since visual observation does not provide full information about the state (e.g., velocities), we use the standard practice of stacking three consecutive frames and using them as input to the control policy.

Table 1: The total number of environment training steps for DMC tasks. The first and second rows show the number of environment training steps (in million) done during design-specific training and design optimization, respectively.

Number of steps, millions	Reacher: Hard	Walker: Stand	Walker: Walk	Walker: Run	Finger: Spin	Finger: Turn Easy
Design-specific training	10	20	20	20	20	30
Design optimization	15	80	80	80	100	100

Training Process. To maintain consistency with the navigation experiments, we use the PPO [6] learning algorithm with the following hyperparameters. We use $\gamma = 0.99$ for reward discounting, GAE $\lambda = 0.95$, and $\epsilon = 0.2$ for the PPO clipping loss. We train the control policies using half of a V100 or A100 GPU. During training, we have 10 parallel environments while each environment collects 10000 steps of experience per rollout. Here each rollout collection step is equivalent to 0.1M simulation steps. We split the collected rollouts into mini-batches of 1000 and perform 4 epochs of PPO updates. We show the specific number of simulation steps for each task in Tab. 1. We use Adam [4] optimization method with a learning rate 0.0001.

5.4 Design Optimization

Navigation Tasks. In `PointGoalNav` and `TargetNav`, we use a Gaussian distribution as the *design policy* $\pi_\phi(\theta) = \mathcal{N}(\theta \,|\, \mu, \mathrm{diag}(\sigma))$, where $\phi = (\mu, \sigma)$, $\mu, \sigma \in \mathbb{R}^{K\times 7}$ is the mean and standard deviation, θ is the design parameter, and K is the number of PRs. We initialize the mean to be a zero vector $\mu = \mathbf{0}^{K\times 7}$ and set the initial standard deviation to be 0.2, i.e., $\sigma = 0.2 \times \mathbf{1}^{K\times 7}$. We separate the design optimization into two stages: *Frozen Stage* and *Update Stage*.

Frozen Stage: In this phase, the design policy is "frozen", and we only train the control policy to act as a *local generalist*. At the beginning of each episode, the design parameter θ is sampled from the frozen design policy, $\theta \sim \pi_\phi(\cdot)$, thereby altering the robot design. As outlined in Section 5.2 of the main paper, the local generalist policy is optimized to manage control within a specific range of design parameters centered around the mean μ. The scope of this range is determined by the standard deviation σ; a larger σ allows the policy to handle a wider variety of design parameters, while a smaller σ limits it to a narrower range. During this stage, the control policy undergoes training for 20k rollout collection steps (153M simulation steps).

Update Stage: During this phase, both the design policy and the control policy undergo training simultaneously. We initiate updates to the design policy every 100 rollout collection steps (6M simulation steps) following each update of the control policy every 400 rollout collection steps (3.1M simulation steps).

When updating the design policy, we maintain the control policy in a frozen state. The objective is to align the design policy with the distribution of returns across the design parameter space. Instead of using returns as the primary objective, which tends to favor longer episodes due to accumulated rewards, we adopt SoftSPL (Soft Success Weighted by Path Length) [3] as the objective function. SoftSPL balances episode efficiency and success more effectively by considering the minimum distance achieved to the target, thus providing a denser and smoother reward landscape for optimizing the design policy.

Concurrently, when updating the control policy, we freeze the design policy. This approach ensures that the control policy adapts to manage the agent within the local parameters defined by the updated design policy. Each rollout consists of 64 steps to facilitate more frequent updates of the policies.

This dual updating strategy allows for comprehensive refinement of both the design and control policies, ensuring robust performance across various task scenarios.

DeepMind Control Suite. We use the same Gaussian design policy as in navigation setting. The total number of simulation steps dedicated to design optimization for each task is detailed in Tab. 1. During the Update Stage, we utilize return as the objective function for both the control and design policies. To adapt the control policy to the changing design policy, we update the design policy every single rollout collection step (0.1M simulation steps), following each training session of the control policy every 8 rollout collection steps (0.8 million simulation steps).

5.5 Network Architecture Details

Each photoreceptor input token to the Transformer consists of the RGB triple, the position embedding based on the position of the photoreceptor on the grid and the design parameters of the grid. This forms the encoder input to the control policy π. In the `PointGoalNav` and `TargetNav` navigation tasks, the policy is a 2-layer LSTM, while in DeepMindControl (DMC), we stack the last 3 frames' encodings as input to the policy.

6 Human Study for the Intuitive Designs

6.1 Visual Navigation Tasks

To compare the performance of our design optimization algorithm with that of human engineers, we conducted a survey. Participants were asked to optimize the position, orientation, and field of view (FoV) of visual sensors on a robot, aiming to enable it to complete the `TargetNav` task as quickly as possible.

The survey asks each participant to design photoreceptors for an agent for both K=2 and K=4 configurations. To aid visualization, we provide an interactive 3D render of the robot and its sensors within a sample environment (top-down view shown in Fig. 13). The target is a green ball with a radius of 0.5 meters, positioned 1.5 meters above the ground.

The sample environment used is a true-to-scale mesh of a home environment, along with a mesh of the target positioned 1.5 meters above the floor. The home environment serves as a simplified stand-in for a Matterport3D mesh. Additionally, the following information about the robot is provided to the participants:

1. At every step, the robot can take one of four possible actions: move forward 0.25 meters, turn left 30 degrees, or turn right 30 degrees.
2. The robot is controlled by a reinforcement learning (RL) trained policy that uses the visual output from the robot's sensors to navigate to the target. The RL policy rewards the robot for navigating to the ball with the least distance traveled. The policy has memory of the robot's past actions and visual inputs through an LSTM. Familiarity with RL is not required to complete the survey.
3. The robot's forward direction is along the positive Z axis.
4. The robot has a height of 2.5 meters, which is also the maximum height for sensor placement.

Fig. 13: A bird's eye view of the environment in Level 2. The target is circled in red, and the robot is circled in orange.

6.2 Continuous Control Tasks from the DMC Suite

The DMC benchmark uses the MuJoCo simulator [8], which is challenging to deploy within a browser due to the specialized format used to define scenes and the agent's morphology. Therefore, we ask participants to sketch their placement design based on a rendered image depicting the agent's morphology and the environment. Given a sketch, we implement the design inside the simulator, show it to the participant, and update it based on their feedback until convergence, i.e., when the participant agrees that the design corresponds to their intended placement. We collect ten intuitive designs for two tasks within one domain. Each participant provides one design for both tasks (`Walker:Walk` and `Walker:Stand`), which share the `Walker` domain. We provide a description of these two tasks similar to that in the original paper introducing the DMC benchmark [7].

References

1. Anderson, P., Chang, A., Chaplot, D.S., Dosovitskiy, A., Gupta, S., Koltun, V., Kosecka, J., Malik, J., Mottaghi, R., Savva, M., Zamir, A.R.: On Evaluation of

Embodied Navigation Agents (Jul 2018), http://arxiv.org/abs/1807.06757, arXiv:1807.06757 [cs] 10
2. Chang, A., Dai, A., Funkhouser, T., Halber, M., Nießner, M., Savva, M., Song, S., Zeng, A., Zhang, Y.: Matterport3D: Learning from RGB-D Data in Indoor Environments (Sep 2017), http://arxiv.org/abs/1709.06158, arXiv:1709.06158 [cs] 9, 11
3. Datta, S., Maksymets, O., Hoffman, J., Lee, S., Batra, D., Parikh, D.: Integrating egocentric localization for more realistic point-goal navigation agents. In: Conference on Robot Learning. pp. 313–328. PMLR (2021) 13
4. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. Tech. Rep. arXiv:1412.6980, arXiv (Jan 2017). https://doi.org/10.48550/arXiv.1412.6980, http://arxiv.org/abs/1412.6980 10, 12
5. Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P.: High-Dimensional Continuous Control Using Generalized Advantage Estimation (Oct 2018), http://arxiv.org/abs/1506.02438, arXiv:1506.02438 [cs] 9
6. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. arXiv pp. 1–12 (2017), http://arxiv.org/abs/1707.06347, arXiv: 1707.06347 9, 12
7. Tassa, Y., Tunyasuvunakool, S., Muldal, A., Doron, Y., Trochim, P., Liu, S., Bohez, S., Merel, J., Erez, T., Lillicrap, T., Heess, N.: dm_control: Software and Tasks for Continuous Control. Software Impacts **6**, 100022 (Nov 2020). https://doi.org/10.1016/j.simpa.2020.100022, http://arxiv.org/abs/2006.12983, arXiv:2006.12983 [cs] 11, 14
8. Todorov, E., Erez, T., Tassa, Y.: MuJoCo: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 5026–5033. IEEE (2012). https://doi.org/10.1109/IROS.2012.6386109 14
9. Wijmans, E., Kadian, A., Morcos, A., Lee, S., Essa, I., Parikh, D., Savva, M., Batra, D.: DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames (Jan 2020). https://doi.org/10.48550/arXiv.1911.00357, http://arxiv.org/abs/1911.00357, arXiv:1911.00357 [cs] 9