Multi-Robot Deep Reinforcement Learning for Mobile Navigation

Anonymous Author(s)
Affiliation
Address
email

Abstract: Deep reinforcement learning algorithms require large and diverse datasets in order to learn successful policies for perception-based mobile navigation. However, gathering such datasets with a single robot can be prohibitively expensive. Collecting data with multiple different robotic platforms with possibly different dynamics is a more scalable approach to large-scale data collection. But how can deep reinforcement learning algorithms leverage such heterogeneous datasets? In this work, we propose a deep reinforcement learning algorithm with hierarchically integrated models (HInt). At training time, HInt learns separate perception and dynamics models, and at test time, HInt integrates the two models in a hierarchical manner and plans actions with the integrated model. This method of planning with hierarchically integrated models allows the algorithm to train on datasets gathered by a variety of different platforms, while respecting the physical capabilities of the deployment robot at test time. Our mobile navigation experiments show that HInt outperforms conventional hierarchical policies and single-source approaches.

Keywords: Deep reinforcement learning, Multi-robot learning, Autonomous navigation

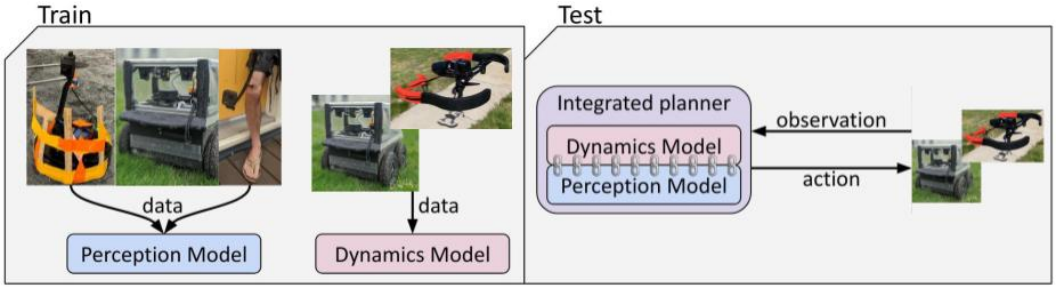

Figure 1: Overview of our hierarchically integrated models (HInt) algorithm. At training time, HInt separately trains a perception model and a dynamics model, then at test time, HInt combines the perception and dynamics model into a single model for integrated planning and execution. Our modular training procedure enables HInt to train the perception model using data gathered by multiple platforms, such as ground robots and even people recording video with a hand-held camera, while our integrated model at test time ensures the perception model only considers trajectories which are dynamically feasible.

1 Introduction

Machine learning provides a powerful tool for enabling robots to acquire control policies by learning directly from experience. One of the key guiding principles behind recent advances in machine learning is to leverage large datasets. In previous works in deep reinforcement learning for robotic control, the most common approach is to collect data from a single robot, train a policy in an end-to-end fashion, and deploy the policy on the same data-collection platform. This approach necessitates collecting a large and diverse dataset for every model of robot we wish to deploy on, which can present a significant practical obstacle, since there exists a plethora of different platforms in the

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.

real world. What if we could instead leverage datasets collected by a variety of *different* robots in our training procedure? An ideal method could use data from *any* platform that provides useful knowledge about the problem. As an example, it is expensive and time-consuming to gather a large dataset for visual navigation on a drone, due to on-board battery constraints. In comparison, it is much easier to collect data on a robot car. Because knowledge about the visual features of obstacles can be shared across vision-based mobile robots, making use of visual data collected by a car to train a control policy for a drone could significantly reduce the amount of data needed from the drone. Unfortunately, data from such heterogeneous platforms presents a challenge: different platforms have different physical capabilities and action spaces. In order to leverage such heterogeneous data, we must properly account for the underlying dynamics of the data collection platform.

To learn from multiple sources of data, previous works have utilized hierarchical policies [1, 2, 3]. In this type of method, a high-level and a low-level policy are trained separately. At test time, the actions generated by the high-level policy are used as waypoints for the low-level policy. By separating the policy into two parts, these algorithm are able to utilize data from multiple sources in the high-level policy, while representing robot-specific information such as the dynamics in the low-level policy. One drawback of these methods, however, is that the low-level policy is unable to communicate any robot-specific information to the high-level policy. This makes it possible for the high-level policy to command waypoints that are impossible for the robot to physically achieve, leading to poor performance and possibly dangerous outcomes.

The key idea in our approach, illustrated in Fig. 1, is to instead learn hierarchical *models*, and to integrate the hierarchical models for planning. At training time, we learn a perception model that reasons about interactions with the world, using our entire multi-robot dataset, and a dynamics model specific to the deployment robot, using only data from that robot. At test time, the perception and dynamics models are combined to form a single integrated model, which a planner uses to choose the actions. Such hierarchically integrated models can leverage data from multiple robots for the perception layer, while also reasoning about the physical capabilities of the deployment robot during planning. This is because when the algorithm uses the hierarchically integrated model to plan, the dynamics model can "hint" to the perception model about the robot-specific physical capabilities of the deployment robot, and the planner can only select behaviors that the deployment robot can actually execute (according to the dynamics model). In contrast to hierarchical policies, which may produce waypoint commands that are dynamically infeasible for the deployment robot to achieve, our hierarchically integrated models take the robot's physical capabilities into account, and only permit physically feasible plans.

The primary contribution of this work is HInt— hierarchically integrated models for acquiring image-based control policies from heterogeneous datasets. We demonstrate that HInt successfully learns policies from heterogeneous datasets in real-world navigation tasks, and outperforms methods that use only one data source or use conventional hierarchical policies.

2 Related Work

Prior work demonstrated end-to-end learning for vision-based control for a wide variety of applications, from video games [4, 5] and manipulation [6, 7] to navigation [8, 9]. However, these approaches typically require a large amount of diverse data [10], which hinders the adoption of these algorithms for real-world robot learning. One approach for overcoming these data constraints is to combine data from multiple robots; While prior methods have addressed collective learning, they typically assume that the robots are the same [11], have similar underlying dynamics [12], or the data is from expert demonstrations [13, 14, 15]. Our approach learns from off-policy data gathered by robots with a variety of underlying dynamics. Prior methods have also transferred skills from simulation, where data is more plentiful [16, 17, 18, 19, 20, 21, 22]. In contrast, our method does not require any simulation, and instead is aimed at leveraging heterogeneous real-world data sources, though it could be extended to incorporate simulated data as well.

Prior work has also investigated learning hierarchical vision-based control policies for applications such as autonomous flight [23, 2] and driving [1, 24, 3, 25]. One advantage of these conventional hierarchy approaches is that many can leverage heterogeneous datasets [26, 12, 23]. However, *even if each module is perfectly accurate*, these conventional hierarchy approaches can still fail because the low-level policy cannot communicate the robot's capabilities and limitations to the high-level policy. In contrast, because HInt learns hierarchical models, and performs planning on the integrated

models at test time, HInt is able to jointly reason about the capabilities of the robot and the perceptual features of the environment.

End-to-end algorithms that can leverage datasets from heterogeneous platforms have also been investigated by prior work; however, these methods typically require on-policy data, are evaluated in visually simplistic domains, or have only been demonstrated in simulation [27, 28, 29]. In contrast, HInt works with real-world off-policy datasets because at the core of HInt are predictive models, which can be trained using standard supervised learning.

3 HInt: Hierarchically Integrated Models

Our goal is to learn image-based mobile navigation policies that can leverage data from heterogeneous robotic platforms. The key contribution of our approach, shown in Fig. 2, is to learn separate hierarchical models at training time, and combine these models into a single integrated model for planning at test time. The two hierarchical models include a high-level, shared perception model, and a low-level, robot-specific dynamics model. The perception model can be trained using data gathered by a variety of different robots, all with possibly different dynamics, while the robot-specific dynamics model is trained only using data from the deployment robot. At test time, because the output predictions of the dynamics model —which are kinematic poses —are the input actions for the perception model, the dynamics and perception models can be combined into a single integrated model. The separate hierarchical model training allows our method to leverage datasets gathered by heterogeneous platforms, while the integrated planning enables our approach to directly map raw sensory inputs to robot actions that are dynamically feasible for the deployment robot. In the following sections, we will describe how HInt trains a perception model and a dynamics model, combines these models into a single integrated model in order to perform planning, and conclude with an algorithm summary.

3.1 Perception Model

The perception model is a neural network predictive model $f_\theta^{\text{PER}}(\mathbf{o}_t, \boldsymbol{\delta}\mathbf{p}_{t:t+H}) \rightarrow \hat{r}_{t:t+H}$—shown in Fig. 2— parameterized by model parameters θ. f_θ^{PER} takes as input the current image observation $\mathbf{o}_t$ and a sequence of H future changes in poses $\boldsymbol{\delta}\mathbf{p}_{t:t+H} = (\boldsymbol{\delta}\mathbf{p}_t, \boldsymbol{\delta}\mathbf{p}_{t+1}, ..., \boldsymbol{\delta}\mathbf{p}_{t+H-1})$, and predicts H future rewards $\hat{r}_{t:t+H} = (\hat{r}_t, \hat{r}_{t+1}, ..., \hat{r}_{t+H-1})$. The pose $\mathbf{p}$ characterizes the kinematic configuration of the robot. In our experiments, we used the robot's x position, y position, and yaw orientation as the pose, though 3D configurations that include height, roll, and pitch are also possible. This kinematic configuration provides a dynamics-agnostic interface between the perception model and the dynamics model. The perception model is trained using the heterogeneous dataset $\mathcal{D}^{\text{PER}}$ by minimizing the objective:

$$\mathcal{L}^{\text{PER}}(\theta, \mathcal{D}^{\text{PER}}) = \sum_{(\mathbf{o}_t, \boldsymbol{\delta}\mathbf{p}_{t:t+H}, r_{t:t+H})} \|\hat{r}_{t:t+H} - r_{t:t+H}\|_2^2 \tag{1}$$
$$\hat{r}_{t:t+H} = f_\theta^{\text{PER}}(\mathbf{o}_t, \boldsymbol{\delta}\mathbf{p}_{t:t+H})$$

The perception model can be trained with a heterogeneous dataset consisting of data gathered by a variety of robots, all with possibly different underlying dynamics. The only requirement for the data is that the recorded sensors (e.g., camera) are approximately the same, and that we can calculate the

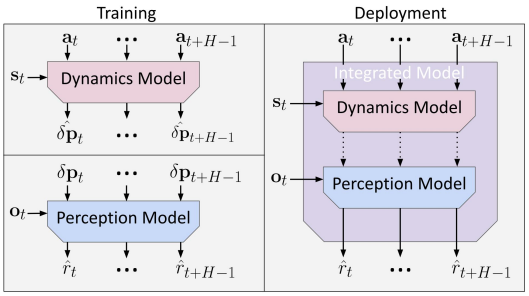

Figure 2: During training, we learn two separate neural network models. The dynamics model takes as input the current robot state and a sequence of future actions, and predicts future changes in poses. This model is trained using data gathered by a single robot. The perception model takes as input the current image observation and a sequence of future changes in poses, and predicts future rewards. This model is trained using data gathered by a variety of robots that have the same image observations, but potentially different dynamics. When deploying, the dynamics and perception models are combined into a single integrated model that is used for planning and executing actions that maximize reward.

change in pose—position and orientation—between each sequential datapoint, which could be done using visual or mechanical odometry. This ability to train using datasets gathered by heterogeneous platforms is crucial because the perception model is an image-based neural network, which requires large amounts of data in order to effectively generalize.

3.2 Dynamics Models

The dynamics model is a neural network predictive model $f_\phi^{\text{DYN}}(\mathbf{s}_t, \mathbf{a}_{t:t+H}) \rightarrow \hat{\boldsymbol{\delta}\mathbf{p}}_{t:t+H}$—shown in Fig. 2— parameterized by model parameters ϕ. f_ϕ^{DYN} takes as input the current robot state $\mathbf{s}_t$ and a sequence of H future actions $\mathbf{a}_{t:t+H} = (\mathbf{a}_t, \mathbf{a}_{t+1}, ..., \mathbf{a}_{t+H-1})$, and predicts H future changes in poses $\hat{\boldsymbol{\delta}\mathbf{p}}_{t:t+H} = (\hat{\boldsymbol{\delta}\mathbf{p}}_t, \hat{\boldsymbol{\delta}\mathbf{p}}_{t+1}, ..., \hat{\boldsymbol{\delta}\mathbf{p}}_{t+H-1})$. The state $\mathbf{s}$ can include any sensor information, such as the robot's battery charge or velocity, that may help the model to make more accurate predictions of future poses. The dynamics model is trained using the respective robot-specific dataset $\mathcal{D}^{\text{DYN}}$ by minimizing the objective:

$$\mathcal{L}^{\text{DYN}}(\phi, \mathcal{D}^{\text{DYN}}) = \sum_{(\mathbf{s}_t, \mathbf{a}_{t:t+H}, \boldsymbol{\delta}\mathbf{p}_{t:t+H})} \|\hat{\boldsymbol{\delta}\mathbf{p}}_{t:t+H} - \boldsymbol{\delta}\mathbf{p}_{t:t+H}\|_2^2 \tag{2}$$
$$\hat{\boldsymbol{\delta}\mathbf{p}}_{t:t+H} = f_\phi^{\text{DYN}}(\mathbf{s}_t, \mathbf{a}_{t:t+H}).$$

Although the dynamics model must only be trained using data collected by the deployment robot, the dynamics model dataset can be significantly smaller compared to the perception model dataset, because the dynamics model inputs are lower dimensional by orders of magnitude. Furthermore, the dynamics model dataset does not need a reward signal, allowing for easier data collection.

3.3 Planning and Control

In order to perform planning and control at test time, we first combine the perception and dynamics models into a single integrated model $f_{\theta,\phi}^{\text{COM}}(\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_{t:t+H}) = f_\theta^{\text{PER}}(\mathbf{o}_t, f_\phi^{\text{DYN}}(\mathbf{s}_t, \mathbf{a}_{t:t+H})) \rightarrow \hat{r}_{t:t+H}$, shown in Fig. 2. These models can be combined without any additional training because the output of the dynamics model—changes in kinematic poses—is also the input to the perception model. This integrated model is essential because it enables the planner to holistically reason about the entire system. In contrast, in conventional hierarchical control methods, where the high-level policy outputs a goal for the low-level policy, the high-level policy could output a dynamically infeasible reference trajectory for the low-level controller. Our approach would not suffer from this failure case because, with integrated planning, the perception model can only take as input dynamically-feasible trajectories that are output by the dynamics model.

We then plan at each time step for the action sequence that maximizes reward according to the integrated model by solving the following optimization:

$$\mathbf{a}_{t:t+H}^* = \arg\max_{\mathbf{a}_{t:t+H}} R(\hat{r}_{t:t+H}, \hat{\boldsymbol{\delta}\mathbf{p}}_{t:t+H}, \mathbf{a}_{t:t+H}, \mathbf{g}_t) \tag{3}$$
$$\hat{r}_{t:t+H} = f_{\theta,\phi}^{\text{COM}}(\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_{t:t+H}).$$

Here, R is a user-defined task-specific reward function, and $\mathbf{g}_t$ is a user-specified goal. We follow the framework of model predictive control, where at each time step, the robot calculates the best sequence of actions for the next H steps, executes the first action in the sequence, and then repeats the process at the next time step. The action sequence that approximately maximizes the objective in Eqn. 3 can be computed using any optimization method. In our implementation, we employ stochastic zeroth-order optimization, as is common in model-based RL [30, 31]. Specifically, we used either the cross-entropy method (CEM) [32] or MPPI [33], depending on the computational constraints of the platform.

3.4 Algorithm Summary

We now briefly summarize how our HInt system operates during training and deployment, as shown in Fig. 1 and Fig. 2.

During training, we first gather perception data using a number of platforms. For each of these platforms, we save the onboard observations $\mathbf{o}$, change in poses $\boldsymbol{\delta}\mathbf{p}$, and rewards r into a shared

dataset $\mathcal{D}^{\text{PER}}$ (see §B.1), and use this dataset to train the perception model f_θ^{PER} (Eqn. 1) (see §B.2). Then, using the robot we will deploy at test time, we gather dynamics data by having the robot act in the real world and recording the robot's onboard states $\mathbf{s}$, actions $\mathbf{a}$, and change in poses $\boldsymbol{\delta}\mathbf{p}$ into the dataset $\mathcal{D}^{\text{DYN}}$ (see §C.1); we use this dataset to train the dynamics model f_ϕ^{DYN} (Eqn. 2) (see §C.2).

When deploying HInt, we first combine the perception model f_θ^{PER} and dynamics model f_ϕ^{DYN} into a single model $f_{\theta,\phi}^{\text{COM}}$. The robot then plans a sequence of actions that maximizes reward (Eqn. 3), executes the first action, and repeat this planning process at each time step until the task is complete, as is standard in model-based RL with model-predictive control [30, 31] (see §D).

4 Experiments

We now present an experimental evaluation of our hierarchically integrated models algorithm in the context of real world mobile robot navigation tasks. Videos and additional details about the data sources, models, training procedures, and planning can be found in the Appendix section or on the project website: https://sites.google.com/view/multi-robot-drl

In our experiments, we aim to answer the following questions:

Q1: Does leveraging data collected by other robots, in addition to data from the deployment robot, improve performance compared to only learning from data collected by the deployment robot?

Q2: Does HInt's integrated model planning approach result in better performance compared to conventional hierarchy approaches?

In order to separately examine these questions, we investigate **Q1** by training the perception module with multiple different real-world data sources, including data from different environments and different platforms, and evaluating on a single real-world robot. To examine **Q2**, we deploy a shared perception module to systems with different low-level dynamics.

4.1 Q1: Comparison to Single-Source Models

In this experiment, we deployed a robot in a number of visually diverse environments, including ones in which *that robot itself has not collected any data*. Our hypothesis is that HInt, which can make use of heterogeneous datasets gathered by multiple robots, will outperform methods that can only train using data gathered by the deployment robot.

Perception data was collected in three different environments using three different platforms (Fig. 3): a Yujin Kobuki robot in an indoor **office** environment (3.7 hours), a Clearpath Jackal in an outdoor **urban** environment (3.5 hours), and a person recording video with a hand-held camera in an **industrial** environment (1.2 hours). The deployment robot is the Clearpath Jackal. The same dataset that provided the Jackal perception data as described above was also used for the Jackal dynamics data.

The robot's objective is to drive towards a goal location while avoiding collisions and minimizing action magnitudes. More specifically, the reward r is -1 for a collision and 0 otherwise, the goal $\mathbf{g}$ specifies a GPS location, and the reward function used for planning is

$$\begin{aligned} &R(\hat{r}_{t:t+H}, \hat{\boldsymbol{\delta}\mathbf{p}}_{t:t+H}, \mathbf{a}_{t:t+H}, \mathbf{g}_t) \\ &= \sum_{h=0}^{H-1} \hat{r}_{t+h} - 0.3 \cdot \angle(\hat{\boldsymbol{\delta}\mathbf{p}}_{t+h}, \mathbf{g}_t) - 0.05 \cdot \|\hat{\boldsymbol{\delta}\mathbf{p}}_{t+h}\|_1. \end{aligned} \tag{4}$$

We compared HInt, which learns from all the data sources, with the single data source approach from Kahn et al [34]. Similar to HInt, this method also uses a vision-based predictive model to perform planning, but differs from HInt in the following ways: (i) only data gathered by the deployment robot can be used for training and (ii) the integrated perception and dynamics model is trained end-to-end. In our implementation, HInt and the single data source approach use the same training parameters and neural network architecture for the integrated model in order to make the most fair comparison.

Fig. 4 shows the results comparing HInt to the single data source approach. In all environments[1], our approach is more successful in reaching the goal. Note that even when the single data source method

[1] We could not run experiments in the office environment due to COVID-related closures.

Figure 3: Training data was gathered by an indoor Yujin Kobuki robot in an office, an outdoor Clearpath Jackal robot in an urban environment, and a person with a video camera in an industrial area. HInt is able to jointly train on this heterogeneous data, which leads to improved navigation performance. HInt was deployed on the ourdoor Jackal robot as well as a Parrot Bebop drone in the urban environment. Because HInt is able to reason about the dynamics of the deployment robot during planning, it is able to successfully control robots that are not included in the perception dataset, such as the drone.

		Perception Data Sources				
		Single-Source [34]			HInt (ours)	
		Kobuki (Office)	Jackal (Urban)	Human (Industrial)	Kobuki (Office) + Jackal (Urban)	Kobuki (Office) + Jackal (Urban) + Human (Industrial)
Test Env	Urban	0%	13%	N/A	**100%**	**100%**
	Industrial	0%	N/A	0%	33%	**87%**

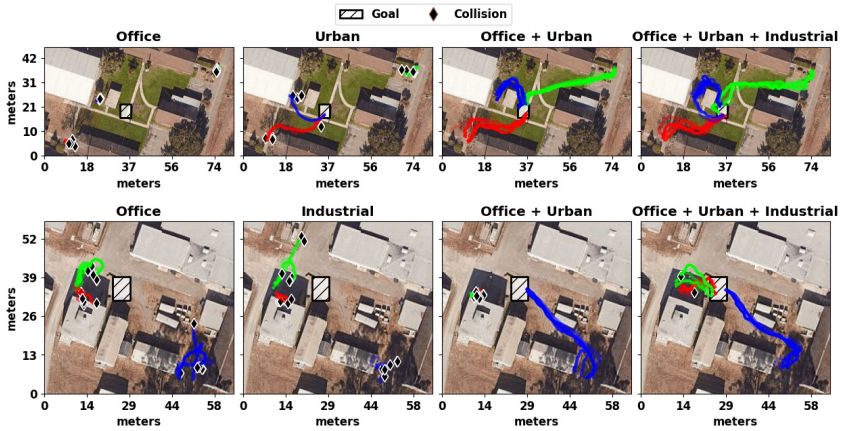

Figure 4: Comparison of single data source models [34] versus our HInt approach in an urban and industrial environment for the task of reaching a goal location while avoiding collisions using the Clearpath Jackal robot. Note that HInt was trained with more datapoints than the single source approach, because HInt is able to learn from data collected by other platforms in addition to the Jackal deployment robot. Each approach was evaluated from the 3 same start locations in each environment (corresponding to the red, green, and blue lines), and was ran 5 times from each start location. The quantitative results show what percentage of the 15 trials successfully reached the goal.

is trained and deployed in the same environment, HInt still performs better, because learning-based methods benefit from large and diverse datasets. Furthermore, the row labeled "Industrial" illustrates well how HInt can benefit from data collected with other platforms: although the Jackal robot had never seen the industrial setting during training, the training set did include data collected by a person with a video camera in this setting. The increase in performance from including this data ("Kobuki + Jackal + Human") shows that the Jackal robot was able to effectively integrate this data into its visual navigation strategy. Fig. 5 shows first-person images of HInt successfully navigating in both the urban and industrial environments.

4.2 Q2: Comparison to Conventional Hierarchy

In this experiment, we deployed the perception module onto robots with different low-level dynamics, including ones that were not seen during the collection of the perception data. Our hypothesis is that our integrated HInt approach will outperform conventional hierarchy approaches, because it is able to jointly reason about perception and dynamics.

We compared our integrated approach with the most commonly used hierarchical approach, in which the perception model is used to output desired waypoints that are then passed to a low-level con-

Figure 5: Visualization of HInt at test time successfully reaching the goal location while avoiding collisions in urban (top) and industrial (bottom) environments.

troller [1, 23, 24, 2, 3]. In our prediction- and planning-based framework, this modular approach is implemented by first planning over a sequence of positions $\boldsymbol{\delta}\mathbf{p}^*_{t:t+H}$ by using a kinematic vision-based model, and then planning over a sequence of actions $\mathbf{a}^*_{t:t+H}$ so as to minimize the tracking error against these planned positions using a robot-specific dynamics model. This baseline represents a clean "apples-to-apples" comparison between our approach – which directly combines both the dynamic and kinematic models into a single model – and a conventional pipelined approach that separates vision-based kinematic planning with low-level trajectory tracking. Both our method and this baseline employ the same neural network architectures for the vision and dynamics models, and train them on the same data, thus providing for a controlled comparison that isolates the question of whether our end-to-end approach improves over a conventional pipelined hierarchical approach. The planning process for the hierarchical baseline can be expressed as the following two-step optimization:

$$\boldsymbol{\delta}\mathbf{p}^*_{t:t+H} = \arg\max_{\boldsymbol{\delta}\mathbf{p}_{t:t+H}} R(\hat{r}_{t:t+H}, \boldsymbol{\delta}\mathbf{p}_{t:t+H}, \mathbf{g}_t) \tag{5}$$

$$\hat{r}_{t:t+H} = f^{\text{PER}}_{\theta}(\mathbf{o}_t, \boldsymbol{\delta}\mathbf{p}_{t:t+H})$$

$$\mathbf{a}^*_{t:t+H} = \arg\min_{\mathbf{a}_{t:t+H}} \sum_{h=0}^{H-1} \|\hat{\boldsymbol{\delta}\mathbf{p}}_{t+h} - \boldsymbol{\delta}\mathbf{p}^*_{t+h}\|_2^2 \tag{6}$$

$$\hat{\boldsymbol{\delta}\mathbf{p}}_{t:t+H} = f^{\text{DYN}}_{\phi}(\mathbf{s}_t, \mathbf{a}_{t:t+H}).$$

We trained the perception model with three data sources from different robots ("Kobuki + Jackal + Human" in Fig. 4), one of which is a Clearpath Jackal robot. We then deployed this perception model on the Jackal robot. The robot's objective is to go towards a goal location while avoiding collisions, using the same reward function for planning as Eqn. 5.

We also evaluated our method on a Parrot Bebop drone, shown in Fig. 3, which was not included in the data collection process for the perception model. The drone's objective is to avoid collisions with minimal turning. The reward function for planning is

$$\begin{aligned} &R(\hat{r}_{t:t+H}, \hat{\boldsymbol{\delta}\mathbf{p}}_{t:t+H}, \mathbf{a}_{t:t+H}, \mathbf{g}_t) \\ &= \sum_{h=0}^{H-1} \hat{r}_{t+h} - \|\hat{\boldsymbol{\delta}\mathbf{p}}_{t+h}\|_1. \end{aligned} \tag{7}$$

The results in Fig. 6 compare HInt against the conventional hierarchy approach on a Jackal robot with its normal dynamics and with its steering limited to 40% of its full steering range, as well as on a Bebop drone flying at low speed (8 degrees forward tilt) and high speed (16 degrees forward tilt). While the two approaches were able to achieve similar performance when evaluated on the normal Jackal and the low speed drone, our approach outperformed conventional hierarchy on the limited steering Jackal and on the high speed drone drone. The poor performance of the conventional hierarchy baseline in these experiments can be explained by the inability of the conventional higher-level planner – which plans kinematic paths based on visual observations – to account for the different dynamics limitations of each platform. This can be particularly problematic near obstacles, where the kinematic model might decide that a last-minute turn to avoid an obstacle may be feasible, when in fact the dynamics of the current robot make it impossible. This is not an issue for the standard

	Normal Jackal	Limited Steering Jackal	Low Speed Drone	High Speed Drone
Conventional Hierarchy	**80%**	0%	**100%**	20%
HInt (ours)	**80%**	**100%**	**100%**	**100%**

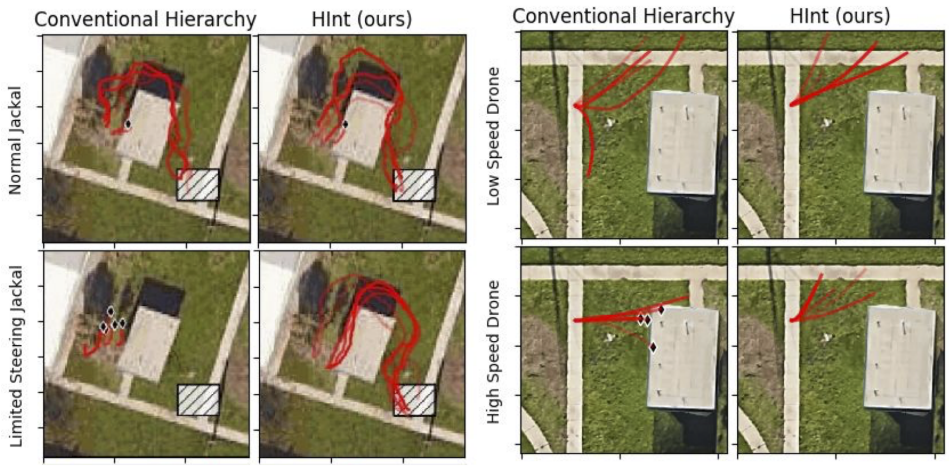

Figure 6: Comparison of conventional hierarchy vs. HInt (ours) approaches in a real world experiment, on Jackal with normal dynamics and with its steering limited to 40%, as well as on a drone with low speed (8 degrees forward tilt) and high speed (16 degrees forward tilt). Both approaches were evaluated with 5 trials each from the same starting position. Our approach is able to achieve higher success rates for reaching the goal region when deployed on the limited steering Jackal and the drone, because the perception model was able to reason about its dynamical limitations.

Jackal robot and the low speed drone, which are both able to make sharp turns. However, the limited steering Jackal was unable to physically achieve the sharp turns commanded by the higher-level model in the baseline method. Similarly, due to the aerodynamics of high-speed flight, the Bebop drone was also unable to achieve the sharp turns commanded by the baseline higher-level planner, leading to collision (see, e.g., the lower-right plot in Fig. 6). In contrast, in HInt, because the low-level model is able to inform the high-level model about the physical capabilities of the robot, the integrated model could correctly deduce that avoiding the obstacles required starting the turn earlier when controlling a less maneuverable robot, which allowed for successful collision avoidance.

We also evaluated HInt and conventional hierarchy on a visual navigation task in simulation involving robot cars with more drastic dynamical differences, such as limited steering, right turn only, and 0.25 seconds lag. More details can be found in the Appendix (see §E). In both the real world and simulation, we showed that conventional hierarchy can fail when deployed on dynamical systems that are not a part of the perception dataset. This is because the higher level perception-based policy can set waypoints for the lower level dynamics-based policy that are outside the physical capabilities of the robot. In contrast, HInt's integrated planning approach enables the dynamics model to inform the perception model about which maneuvers are feasible.

5 Discussion

We presented HInt, a deep reinforcement learning algorithm with hierarchically integrated models. The hierarchical training procedure enables the perception model to be trained on heterogeneous datasets, which is crucial for learning image-based neural network control policies, while the integrated model at test time ensures the perception model only considers trajectories that are dynamically feasible. Our experiments show that HInt can outperform both single-source and conventional hierarchy methods on real-world navigation tasks.

One of the key algorithmic ideas of HInt is that planning at test time enables us to directly use the integrated model without any additional training. However, one of the main limitations is that our approach can still suffer if the outputs of one model are out-of-distribution for the inputs of the subsequent model. Bayesian neural networks and other approaches could provide a principled way to cope with these intermediate out-of-distribution activations. We believe that solving these and other challenges is crucial for enabling robot learning platforms to learn from large visual datasets while still combining perception and control, and that HInt is a promising step towards this goal.

References

[1] W. Gao, D. Hsu, W. S. Lee, S. Shen, and K. Subramanian. Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation. In *CoRL*, 2017.

[2] E. Kaufmann, M. Gehrig, P. Foehn, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza. Beauty and the beast: Optimal methods meet learning for drone racing. In *ICRA*, 2019.

[3] S. Bansal, V. Tolani, S. Gupta, J. Malik, and C. Tomlin. Combining optimal control and learning for visual navigation in novel environments. In *CoRL*, 2019.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[5] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[6] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 2016.

[7] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *CoRL*, 2018.

[8] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert. Learning monocular reactive uav control in cluttered natural environments. In *ICRA*, 2013.

[9] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah. Learning to drive in a day. In *ICRA*, 2019.

[10] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018.

[11] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *IJRR*, 2018.

[12] S. Dasari, F. Ebert, S. Tian, S. Nair, B. Bucher, K. Schmeckpeper, S. Singh, S. Levine, and C. Finn. RoboNet: Large-scale multi-robot learning. In *CoRL*, 2019.

[13] A. Edwards, H. Sahni, Y. Schroecker, and C. Isbell. Imitating latent policies from observation. In *ICML*, 2019.

[14] F. Torabi, G. Warnell, and P. Stone. Generative adversarial imitation from observation. In *ICML Workshop on Imitation and Intent*, 2018.

[15] W. Sun, A. Vemula, B. Boots, and J. A. Bagnell. Provably efficient imitation learning from observation alone. In *ICML*, 2019.

[16] F. Sadeghi and S. Levine. (CAD)2 RL: Real Single-Image Flight without a Single Real Image. *RSS*, 2017.

[17] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *CoRL*, 2017.

[18] F. Zhang, J. Leitner, M. Milford, and P. Corke. Modular deep q networks for sim-to-real transfer of visuo-motor policies, 2017.

[19] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *ICRA*, 2018.

[20] X. Meng, N. Ratliff, Y. Xiang, and D. Fox. Neural Autonomous Navigation with Riemannian Motion Policy. In *ICRA*, 2019.

[21] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis. Learning navigation behaviors end-to-end with autorl. *RA-L*, 2019.

[22] J. Zhang, L. Tai, P. Yun, Y. Xiong, M. Liu, J. Boedecker, and W. Burgard. VR-goggles for robots: Real-to-sim domain adaptation for visual control. *RA-L*, 2019.

[23] A. Loquercio, A. I. Maqueda, C. R. Del-Blanco, and D. Scaramuzza. Dronet: Learning to fly by driving. In *RA-L*, 2018.

[24] M. Müller, A. Dosovitskiy, B. Ghanem, and V. Koltun. Driving policy transfer via modularity and abstraction. In *CoRL*, 2018.

[25] X. Meng, N. Ratliff, Y. Xiang, and D. Fox. Scaling Local Control to Large-Scale Topological Navigation. *arXiv preprint arXiv:1909.12329*, 2019.

[26] D. Hejna, P. Abbeel, and L. Pinto. Hierarchically decoupled imitation for morphological transfer. In *ICML*, 2020.

[27] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *ICRA)*, 2017.

[28] O. Nachum, S. S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. In *NeurIPS*, 2018.

[29] M. Wulfmeier, A. Abdolmaleki, R. Hafner, J. T. Springenberg, M. Neunert, T. Hertweck, T. Lampe, N. Siegel, N. Heess, and M. Riedmiller. Compositional Transfer in Hierarchical Reinforcement Learning. In *RSS*, 2020.

[30] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[31] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

[32] R. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1999.

[33] G. Williams, A. Aldrich, and E. Theodorou. Model predictive path integral control using covariance variable importance sampling. *arXiv:1509.01149*, 2015.

[34] G. Kahn, P. Abbeel, and S. Levine. BADGR: An autonomous self-supervised learning-based navigation system. *arXiv:2002.05700*, 2020.

[35] M. Goslin and M. Mine. The panda3d graphics engine. In *IEEE Computer*, 2004.

[36] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

A Experiment Evaluation Environments

First, we introduce the environments and robots we used for our experiments.

The real world evaluation environments used in the comparison to single-source models (§4.1) consisted of an urban and an industrial environment. The robot is a Clearpath Jackal with a monocular RGB camera sensor. The robot's task is to drive towards a goal GPS location while avoiding collisions. We note that this single GPS coordinate is insufficient for successful navigation, and therefore the robot must use the camera sensor in order to accomplish the task. Further details are provided in Tab. S1.

Variable	Dimension	Description
$\mathbf{o}_t$	15,552	Image at time t, shape $54 \times 96 \times 3$.
$\boldsymbol{\delta}\mathbf{p}_t$	3	Change in x position, y position, and yaw orientation.
$\mathbf{s}_t$	0	There is no inputted state.
$\mathbf{a}_t$	2	Linear and angular velocity.
r_t	1	$r = -1$ if the robot collides, otherwise $r = 0$. The reward is calculated using the onboard collision detection sensors.
$\mathbf{g}_t$	2	GPS coordinate of goal location.

Table S1: Clearpath Jackal and task.

The real world evaluation environments used in the comparison to conventional hierarchy (§4.2) consisted of an urban environment. The robots used in this experiment include the Clearpath Jackal and a Parrot Bebop drone. The settings used for the Jackal experiment are the same as the ones in the comparison to single-source models (Tab. S1). The Parrot Bebop drone also has a monocular RGB camera, and it's task is to avoid collisions while minimizing unnecessary maneuvers. Further details are provided in Tab. S2.

Variable	Dimension	Description
$\mathbf{o}_t$	15,552	Image at time t, shape $54 \times 96 \times 3$.
$\boldsymbol{\delta}\mathbf{p}_t$	3	Change in x position, y position, and yaw orientation.
$\mathbf{s}_t$	0	There is no inputted state.
$\mathbf{a}_t$	1	Angular velocity.
r_t	1	$r = -1$ if the robot collides, otherwise $r = 0$. The reward is calculated using the onboard collision detection sensors.
$\mathbf{g}_t$	0	There is no inputted goal.

Table S2: Parrot Bebop drone and task.

The simulated evaluation environment used in the comparison to conventional hierarchy (§E) was created using Panda3d [35]. The environment is a cluttered enclosed room consisting of randomly textured objects randomly placed in a semi-uniform grid. The robot is a car with a monocular RGB camera sensor. The robot's task is to drive towards a goal region while avoiding collisions. Further details are provided in Tab. S3.

[†]For the lag experiment only, the state consisted of the past 0.25 seconds of executed actions $\mathbf{a}_{t-1}$.

Variable	Dimension	Description
$\mathbf{o}_t$	31,104	Images at time t and $t-1$, each of shape $54 \times 96 \times 3$.
$\boldsymbol{\delta}\mathbf{p}_t$	1	The robot is at a fixed height and travels at a constant speed. Therefore, the change in pose only includes the change in yaw orientation.
$\mathbf{s}_t$	$0^{\dagger}$	There is no inputted state.
$\mathbf{a}_t$	1	Angular velocity (the robot travels at a constant speed).
r_t	1	$r = -1$ if the robot collides, otherwise $r = 0$. The reward is calculated using the onboard collision detection sensors.
$\mathbf{g}_t$	1	Relative angle to goal region.

Table S3: Simulated robot and task.

B Training the Perception Model

The perception model, as discussed in Sec. 3.1, takes as input an image observation and a sequence of future changes in poses, and predicts future rewards. Here, we discuss the perception model training data and training procedure in more detail.

B.1 Training Data

In order to train the perception model, perception data must be collected. For our simulated experiments (§E), perception data was collected by running the reinforcement learning algorithm from [34] in the simulated cluttered environment.

For our real world experiments (§4.1), perception datasets were collected by a Yujin Kobuki robot in an office environment, a Clearpath Jackal in an urban environment, and a person recording video with a GoPro camera in an industrial environment using correlated random walk control policies.

Additional details of the perception data sources is provided in Tab. S4.

Platform	Environment	Sim or Real?	Time Discretization	Data Collection Policy	Size (datapoints / hours)	Reward Label Generation	Pose Label Generation
car	cluttered room	sim	0.25	on-policy	500,000 / 34	physics engine	physics engine
Yujin Kobuki	office	real	0.4	correlated random walk	32,884 / 3.7	collision sensor	odometry from wheel encoders
Clearpath Jackal	urban	real	0.25	correlated random walk	50,000 / 3.5	collision sensor	IMU
Person with GoPro camera	industrial	real	0.33	correlated random walk	12,523 / 1.2	manual labeling	IMU from GoPro

Table S4: Perception model data sources.

B.2 Neural Network Training

The perception model is represented by a deep neural network, with architecture depicted in Fig. S7. It is trained by minimizing the loss in Eqn. 1 using minibatch gradient descent. In Tab. S5, we provide the training and model parameters.

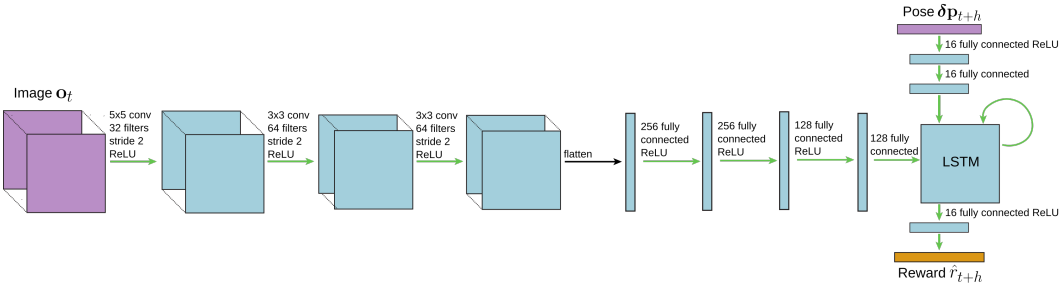

Figure S7: The neural network perception model takes as input the current image observation $\mathbf{o}_t$ and a sequence of H future changes in poses $\boldsymbol{\delta}\mathbf{p}_{t:t+H}$, and predicts H future rewards $\hat{r}_{t:t+H}$. This model is trained using data gathered by a variety of robots that have the same image observations, but different dynamics.

Model Type	Optimizer	Learning Rate	Weight Decay	Grad Clip Norm	Collision Data Rebalancing Ratio	Environment Data Rebalancing Ratio	Hori-zon
simulation	Adam [36]	10^{-4}	0.5	10	No rebalancing	N/A	6
real world: single source	Adam	10^{-4}	0.5	10	50:50	N/A	10
real world: office + urban	Adam	10^{-4}	0.5	10	50:50	25:70	10
real world: office + urban + industrial	Adam	10^{-4}	0.5	10	50:50	33:33:33	10

Table S5: Perception model parameters.

C Training the Dynamics Model

The dynamics model, as discussed in Sec. 3.2, takes as input the robot state and future actions, and predicts future changes in poses. Here, we discuss the dynamics model training data and training procedure in more detail.

C.1 Training Data

In order to train the dynamics model, dynamics data must be collected. For our simulated experiments, the basic simulated car dynamics are based on a single integrator model:

$$\begin{aligned} \theta_{t+1} &= \theta_t + \Delta \cdot \mathbf{a}_t \\ x_{t+1} &= x_t + \Delta \cdot \text{SPEED} \cdot -\sin(\theta_{t+1}) \\ y_{t+1} &= y_t + \Delta \cdot \text{SPEED} \cdot \cos(\theta_{t+1}) \\ \mathbf{a} &\in (\frac{-2\pi}{3}, \frac{2\pi}{3}). \end{aligned} \tag{8}$$

However, we note that this dynamics model was never provided to the learning algorithm; the learning algorithm only ever received samples from the dynamics model. Dynamics data was collected using a random walk control policy for each robot dynamics variation—normal, limited steering, right turn only, and 0.25 second lag.

For our real world experiments (§4.1), the both the Clearpath Jackal and the Parrot Bebop drone collected dynamics data in the urban environment using a correlated random walk control policy.

Additional details of the dynamics data sources is provided in Tab. S6.

Robot	Sim or Real?	Dynamics	Time Discretization	Data Collection Policy	Size (datapoints / hours)
normal car	sim	Eqn. 8	0.25	random walk	5,000 / 0.4
limited steering car	sim	Eqn. 8, but with $\mathbf{a} \in (\frac{-\pi}{3}, \frac{\pi}{3})$	0.25	random walk	5,000 / 0.4
right turn only car	sim	Eqn. 8, but with $\mathbf{a} \in (0, \frac{2\pi}{3})$	0.25	random walk	5,000 / 0.4
0.25 second lag car	sim	Eqn. 8, but with $\theta_{t+1} = \theta_t + \Delta \cdot \mathbf{a}_{t-3}$	0.25	random walk	5,000 / 0.4
Clearpath Jackal	real	Unmodeled	0.25	correlated random walk	50,000 / 3.5
Parrot Bebop drone	real	Unmodeled	0.25	correlated random walk	4977 / 0.35

Table S6: Dynamics model data sources.

C.2 Neural Network Training

The dynamics model is represented by a deep neural network, with architecture depicted in Fig. S8. It is trained by minimizing the loss in Eqn. 2 using minibatch gradient descent. In Tab. S7, we provide the training and model parameters.

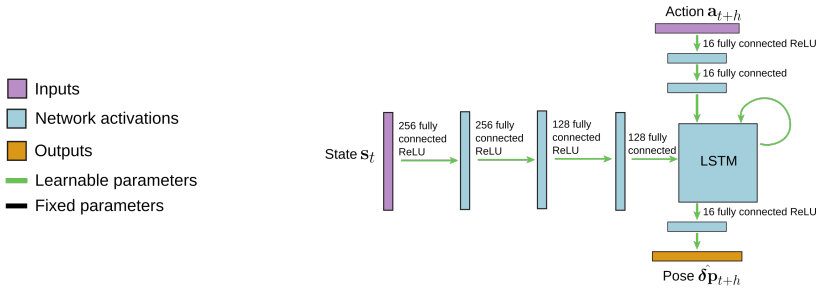

Figure S8: The neural network dynamics model takes as input the current robot state $\mathbf{s}_t$ and a sequence of H future actions $\mathbf{a}_{t:t+H}$, and predicts H future changes in poses $\hat{\boldsymbol{\delta}\mathbf{p}}_{t:t+H}$. This model is trained using data gathered by a single robot.

Model Type	Optimizer	Learning Rate	Weight Decay	Grad Clip Norm	Collision Data Rebalancing Ratio	Environment Data Rebalancing Ratio	Hori-zon
simulation	Adam	10^{-4}	0.5	10	N/A	N/A	6
real world: Jackal	Adam	10^{-4}	0.5	10	N/A	N/A	10
real world: drone	Adam	10^{-4}	0.5	10	N/A	N/A	10

Table S7: Dynamics model parameters.

D Planning

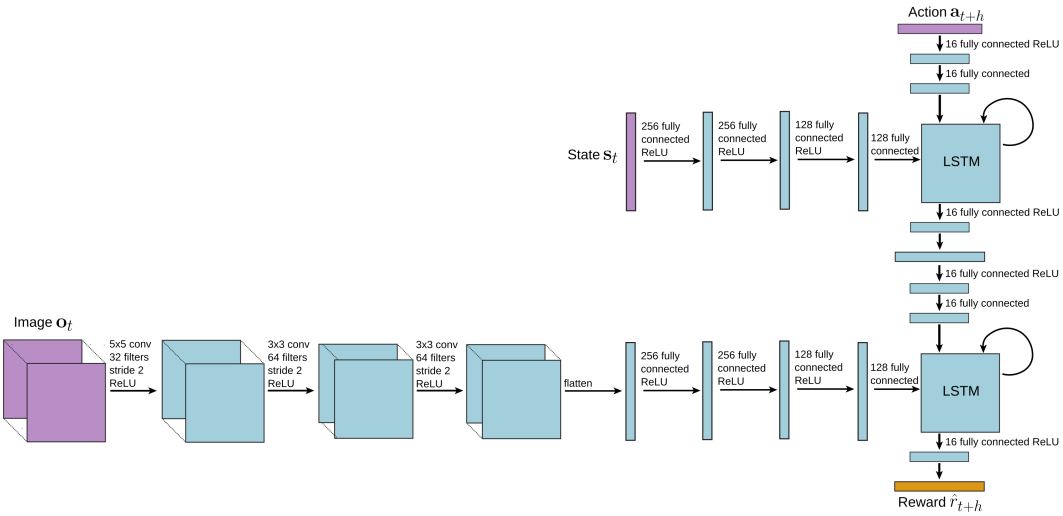

Figure S9: The combined neural network model is a concatenation of the dynamics model (a) and perception model (b). The model takes as input the current image observation $\mathbf{o}_t$, robot state $\mathbf{s}_t$, and a sequence of H future actions $\mathbf{a}_{t:t+H}$, and predicts H future rewards $\hat{r}_{t:t+H}$. This integrated model can then be used for planning and executing actions that maximize reward.

At test time, we perform planning and control using the integrated model (§3.3). The neural network architecture of the integrated model is shown in Fig. S9. At each time step, we plan for the action sequence that maximizes Eqn. 3 using a stochastic zeroth-order optimizer, execute the first action, and continue planning and executing following the framework of model predictive control.

For the simulated experiments (§E), the reward function used for planning is

$$R(\hat{r}_{t:t+H}, \hat{\boldsymbol{\delta}\mathbf{p}}_{t:t+H}, \mathbf{a}_{t:t+H}, \mathbf{g}_t) = \sum_{h=0}^{H-1} \hat{r}_{t+h} - 0.01 \cdot \|\hat{\boldsymbol{\delta}\mathbf{p}}_{t+h}^{\text{HEADING}} - \mathbf{g}_t\|_1, \tag{9}$$

which encourages the robot to drive in the direction of the goal while avoiding collisions. We solve Eqn. 3 using the Cross-Entropy Method (CEM) [32], and replan at a rate of 4 Hz.

For the real world Jackal experiments (§4.1), the reward function used for planning is

$$R(\hat{r}_{t:t+H}, \hat{\boldsymbol{\delta}\mathbf{p}}_{t:t+H}, \mathbf{a}_{t:t+H}, \mathbf{g}_t) = \sum_{h=0}^{H-1} \hat{r}_{t+h} - 0.3 \cdot \angle(\hat{\boldsymbol{\delta}\mathbf{p}}_{t+h}, \mathbf{g}_t) - 0.05 \cdot \|\hat{\boldsymbol{\delta}\mathbf{p}}_{t+h}\|_1, \tag{10}$$

which encourages the robot to avoid collisions, drive towards the goal, and minimize action magnitudes. For the real world drone experiments, the reward function used for planning is

$$R(\hat{r}_{t:t+H}, \hat{\boldsymbol{\delta}\mathbf{p}}_{t:t+H}, \mathbf{a}_{t:t+H}, \mathbf{g}_t) = \sum_{h=0}^{H-1} \hat{r}_{t+h} - \|\hat{\boldsymbol{\delta}\mathbf{p}}_{t+h}\|_1, \tag{11}$$

which encourages the robot to avoid collisions and minimize action magnitudes. We solve Eqn. 3 using the MPPI [33], and replan at a rate of 6 Hz.

E Simulation Experiments in Comparison to Conventional Hierarchy

In addition to the real world experiments in Sec. 4.2, we also performed experiments in simulation to further evaluate the performance of HInt in comparison to conventional hierarchy. We trained a perception model using data collected by a simulated car with a monocular RGB camera. The data was collected by running an on-policy reinforcement learning algorithm inside a cluttered room environment using the Panda3D simulator. Similar to the real world experiments, the rewards here also denote collision, where $r = -1$ if collision and $r = 0$ otherwise.

We deployed the perception model onto variants of the base car's dynamics model, in which the dynamics are modified by: constraining the angular velocity control limits, only allowing the robot to turn right, or inducing a 0.25 second control execution lag. These modifications are very drastic and correspond to extreme versions of the kinds of dynamical variations exhibited by real-world robots.

The robot's objective is to drive to a goal region while avoiding collisions. More specifically, the reward function used for planning is

$$\begin{aligned} &R(\hat{r}_{t:t+H}, \hat{\boldsymbol{\delta}\mathbf{p}}_{t:t+H}, \mathbf{a}_{t:t+H}, \mathbf{g}_t) \\ &= \sum_{h=0}^{H-1} \hat{r}_{t+h} - 0.01 \cdot \|\hat{\boldsymbol{\delta}\mathbf{p}}_{t+h}^{\text{HEADING}} - \mathbf{g}_t\|_1, \end{aligned} \tag{12}$$

which encourages the robot to drive in the direction of the goal while avoiding collisions.

Tab. S8 and Fig. S10 show the results comparing HInt versus the conventional hierarchy approach. While HInt and conventional hierarchy achieved similar performance when deployed on the platform that gathered the perception training data, HInt greatly outperformed the conventional hierarchy approach when deployed on dynamical systems that were not present in the perception training data.

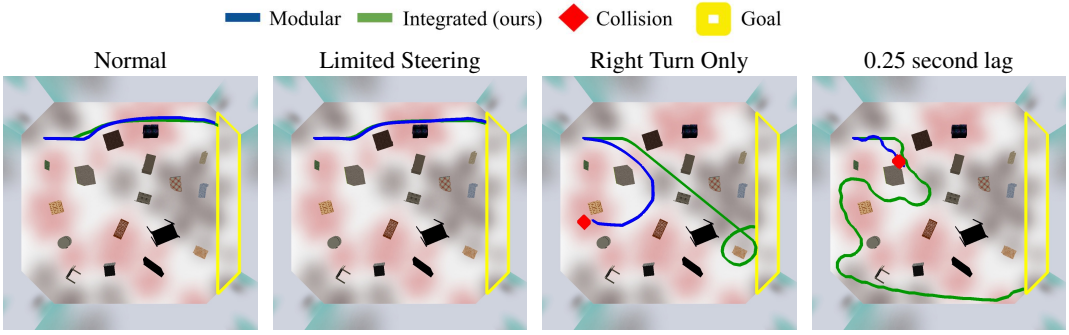

Figure S10: Top down view of example trajectories of both the conventional hierarchy (blue) and our HInt (green) approaches in a simulated environment for the task of driving towards a goal region (yellow) while avoiding collisions. The visualized trajectories are the median performing trajectory of the conventional hierarchy and our HInt approaches for one example starting position. The red diamond indicates a collision. Our integrated approach is able to reach the goal region at a higher rate compared to the modular approach.

	Normal	Limited Steering	Right turn only	0.25 second lag
Conventional Hierarchy	96%	68%	0%	0%
HInt (ours)	**96%**	**84%**	**56%**	**40%**

Table S8: Comparison of conventional hierarchy (e.g., [1, 2, 3]) versus HInt (ours) at deployment time in a simulated environment for the task of driving towards a goal region while avoiding collisions. Four different dynamics models were evaluated—normal, limited steering, right turn only, and 0.25 second lag. Both approaches were evaluated from the same 5 starting positions, with 5 trials for each starting position. Our approach is able to achieve higher success rates for reaching the goal region because the perception model is only able to consider trajectories that are dynamically feasible.

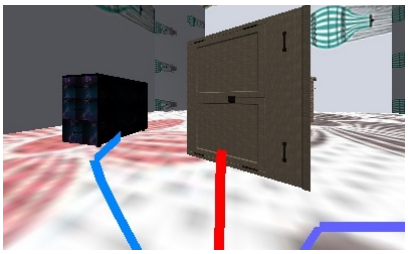

Figure S11: FPV visualization of the right turn only simulated experiment. Here, HInt (purple) successfully avoided the obstacle, while the conventional hierarchy approach failed, because the high-level policy outputted waypoints that avoided the obstacle by turning left (blue), but the dynamics model is unable to turn left, and therefore drove straight into the obstacle (red).

An illustrating example of why hierarchical models are advantageous is shown in Fig. S11. Here, the robot can only turn right, and cannot make left turns. For conventional hierarchical policies, the high-level perception policy—which is unaware of the robot's dynamics—outputs desired waypoints that attempts to avoid the obstacle by turning left. The low-level controller then attempts to follow these left-turning waypoints, but because the robot can only turn right, the best the controller can do is drive straight, which causes the robot to collide with the obstacle. In contrast, our integrated models approach knows that turning right is correct because the dynamics model is able to convey to the perception model that the robot can only turn right, while the perception model is able to confirm that turning right does indeed avoid a collision.