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 (2267 collisions), a Clearpath Jackal in an urban environment (560 collisions), and a person recording video with a GoPro camera in an industrial environment using correlated random walk control policies (204 collisions).

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.

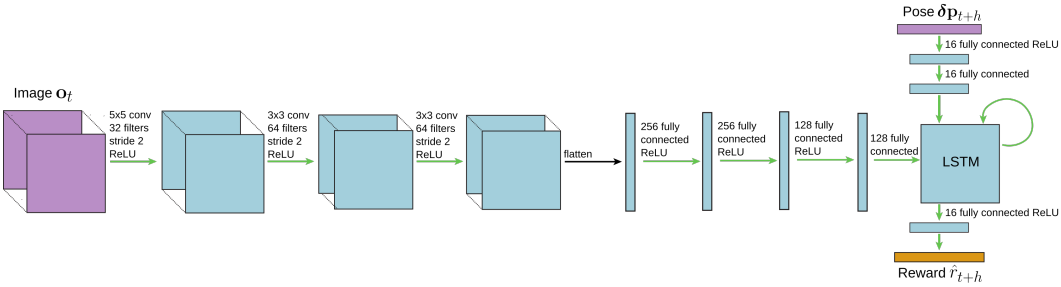

Figure S7: The neural network perception model takes as input the current image observation $\mathbf{o}_t$ and a sequence of H future changes in poses $\boldsymbol{\delta}\mathbf{p}_{t:t+H}$, and predicts H future rewards $\hat{r}_{t:t+H}$. This model is trained using data gathered by a variety of robots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

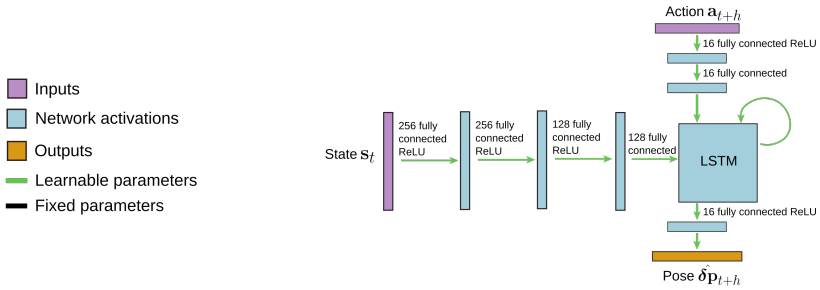

Figure S8: The neural network dynamics model takes as input the current robot state $\mathbf{s}_t$ and a sequence of H future actions $\mathbf{a}_{t:t+H}$, and predicts H future changes in poses $\hat{\boldsymbol{\delta}}\hat{\mathbf{p}}_{t:t+H}$. This model is trained using data gathered by a single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

Model Type	Optimizer	Learning Rate	Weight Decay	Grad Clip Norm	Collision Data Rebalancing Ratio	Environment Data Rebalancing Ratio	Horizon
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

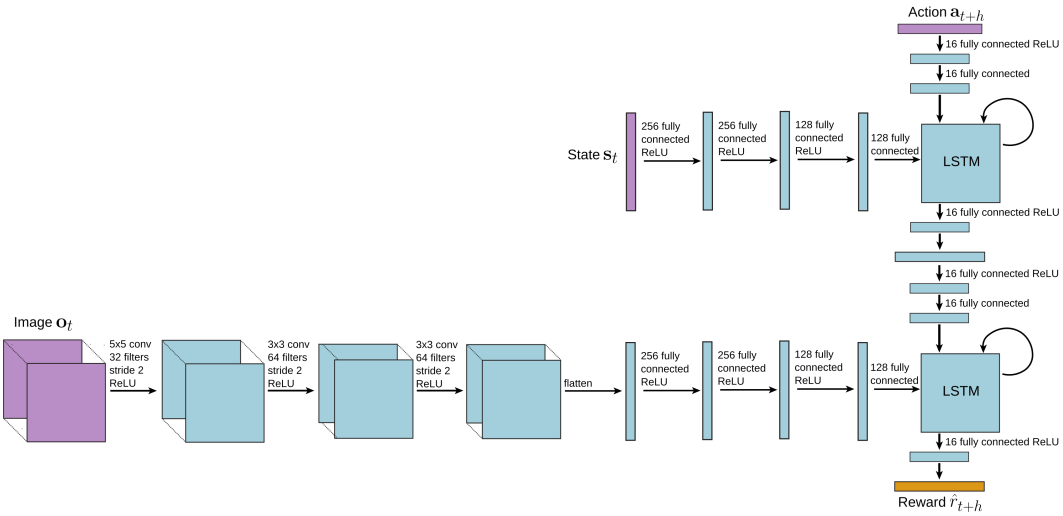

Figure S9: The combined neural network model is a concatenation of the dynamics model (a) and perception model (b). The model takes as input the current image observation $\mathbf{o}_t$, robot state $\mathbf{s}_t$, and a sequence of H future actions $\mathbf{a}_{t:t+H}$, and predicts H future rewards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

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

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.

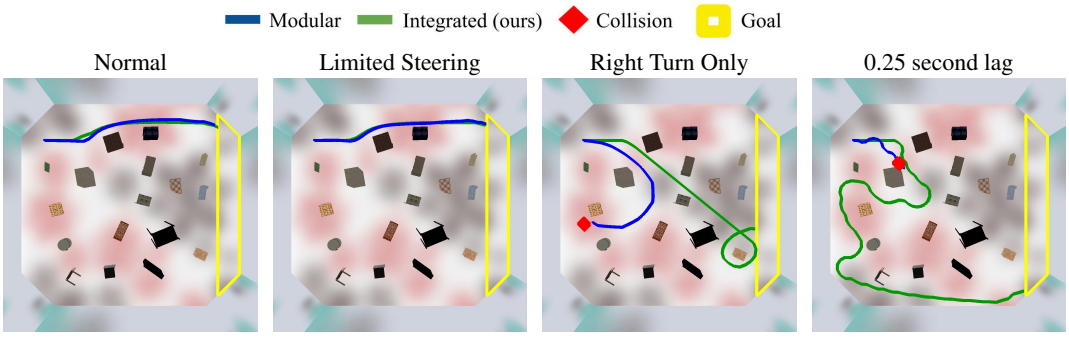

Figure S10: Top down view of example trajectories of both the conventional hierarchy (blue) and our HInt (green) approaches in a simulated environment for the task of driving towards a goal region (yellow) while avoiding collisions. The visualized trajectories are the median performing trajectory of 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.

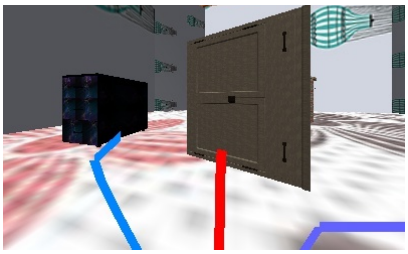

Figure S11: FPV visualization of the right turn only simulated experiment. Here, HInt (purple) successfully avoided the obstacle, while the conventional hierarchy approach failed, because the high-level policy outputted waypoints that avoided the obstacle by turning left (blue), but the dynamics model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