A Appendix

A.1 Pursuer capture time as a function of train vs. test evader

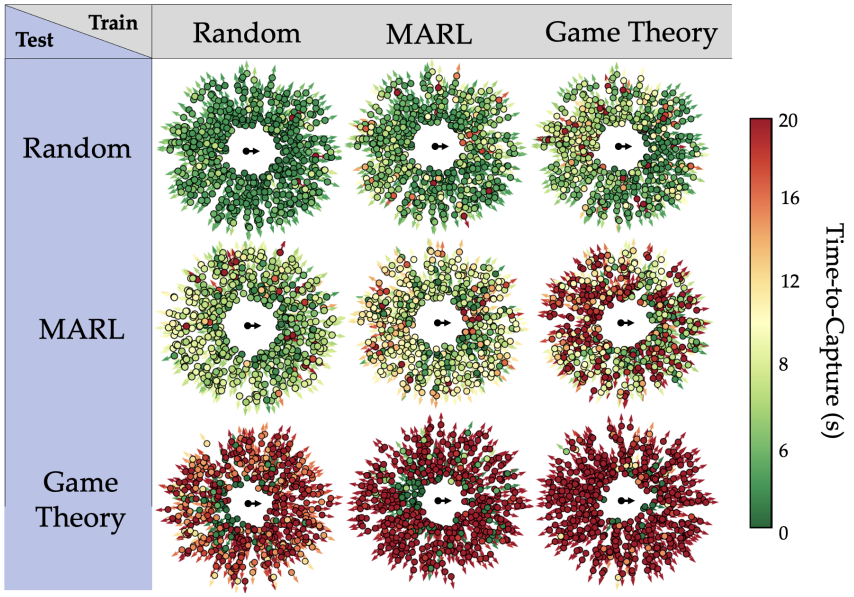

Figure 7: Each pursuer policy is trained on a unique distribution of evader behavior—random, MARL, or zero-sum game-theoretic—and deployed to interact with each other possible evader. The plots reveal how pursuer policies trained on more optimal but less diverse behavior "overfit" to those types of maneuvers. For example, consider the top-right where the pursuer policy is trained against a game theory evader but tested against the random evader. The random evader does not explicitly strategize, but the capture time distribution of the game-theory policy mirrors the distribution when interacting with the game-theoretic evader (Fig. 4).

A.2 Simulation & training details

Quadruped High & Low-level Control. The quadruped uses a pre-computed low-level controller for walking, and is just learning a high-level strategic policy. In this way, our method is agnostic to the specific low-level controller. We use an off-the-shelf RL-based approach to train a walking policy [46]. We use this policy during all simulation experiments and during training the high-level strategic policy. The high-level policy we present in this work controls linear and angular velocity, $u^{\mathrm{P}} = [v_x^{\mathrm{P}}, v_\theta^R] \in \mathcal{U}^{\mathrm{P}}$. The control inputs are bounded: $v_x^{\mathrm{P}} \in [0, 3]\frac{m}{s}$ and $v_\theta^{\mathrm{P}} \in [-2, 2]\frac{rad}{s}$. Note that the pursuer has a $\sim 17\%$ linear speed advantage over the evader (detailed in Appendix A.3).

Optimization Details. The episode length for interactions in the Isaac Gym simulator is always $T = 20s$. The high-level quadruped policy runs at $5\ Hz$, and simulates a new control applied every $\delta t = 0.2s$. The low-level quadruped policy runs at $50\ Hz$. We use the Adam optimizer for both PPO and for DAGGER. We use Exponential Linear Unit (ELU) activations for the policy network and for the encoders. The future state trajectory of the evader is always $N = 4s = 8$ steps. We train the teacher policies with $3,000$ environments, and the student policy with $8,000$ environments. We always train the high level policy on flat terrain in an infinite plane. Otherwise, we use all the default parameters from [46] but remove the cross-entropy loss during PPO.

A.3 Evader simulation & policy design

We simulate the evader as a unicycle dynamical system, controlling linear and angular velocity, $u^{\mathrm{e}} = [v_x^{\mathrm{e}}, v_\theta^{\mathrm{e}}] \in \mathcal{U}^{\mathrm{e}}$. The control inputs are bounded: $v_x^{\mathrm{e}} \in [0, 2.5]\frac{m}{s}$ and $v_\theta^{\mathrm{e}} \in [-2, 2]\frac{rad}{s}$. The discrete-time dynamics model we use is:

$$\begin{bmatrix} p_x^{\mathrm{e}} \\ p_y^{\mathrm{e}} \\ p_z^{\mathrm{e}} \\ \theta^{\mathrm{e}} \end{bmatrix}^{t+\delta t} = \begin{bmatrix} p_x^{\mathrm{e}} \\ p_y^{\mathrm{e}} \\ p_z^{\mathrm{e}} \\ \theta^{\mathrm{e}} \end{bmatrix}^{t} + \delta t \begin{bmatrix} v_x^{\mathrm{e}} \cos(\theta^{\mathrm{e}}) \\ v_x^{\mathrm{e}} \cos(\theta^{\mathrm{e}}) \\ 0 \\ v_\theta^{\mathrm{e}} \end{bmatrix}. \tag{3}$$

The evader's initial condition is randomly initialized at the start of each episode and after each environment reset. Specifically, the initial xy-position of the evader is offset from the pursuer initial xy-position via

$$\begin{bmatrix} p_x^{\mathrm{e}} \\ p_y^{\mathrm{e}} \\ p_z^{\mathrm{e}} \\ \theta^{\mathrm{e}} \end{bmatrix}^{0} = \begin{bmatrix} p_x^{\mathrm{p}} \\ p_y^{\mathrm{p}} \\ 0 \\ 0 \end{bmatrix}^{0} + \begin{bmatrix} r^0 \cos(\psi^0) \\ r^0 \sin(\psi^0) \\ 0.001 \\ \tan^{-1}(p_y^{\mathrm{e},0}/p_x^{\mathrm{e},0}) \end{bmatrix},$$

where $r^0 \sim \mathrm{Unif}[2, 6]$ is drawn uniformly at random between 2 and 6 meters, and $\psi^0 \sim \mathrm{Unif}[-\pi, \pi]$ so that the evader spawns at random angles relative to the pursuer. The z-position is always fixed at 0.001 and the yaw angle θ^{e} is chosen so the evader is facing away from the pursuer at the start of the episode.

A.3.1 Dubins' policy

This evader always moves in Dubins' paths [47] with a fixed time duration for turning or going straight determined randomly upon the start of the episode. Let $v_x^{\mathrm{e}} \in [\underline{v}_x, \bar{v}_x]$ be the min and max linear velocity commands and $v_\theta^{\mathrm{e}} \in [\underline{v}_\theta, \bar{v}_\theta]$ be the min and max angular velocity commands. Let τ_{fwd} be the number of seconds for which the evader goes forward, and τ_{turn} be the number of seconds to turn. At the start of the episode, each parameter is sampled from the corresponding uniform distribution:

$$\tau_{fwd} \sim \mathrm{Unif}[2, 4], \quad \tau_{turn} \sim \mathrm{Unif}[0.6, 1.4].$$

The Dubins' policy switches between two maneuvers depending on if the current time t matches the time interval τ_{fwd} or τ_{turn}:

- τ_{fwd}: evader goes forward at max linear velocity, applying $u^{\mathrm{e}} = [\bar{v}_x, 0]$
- τ_{turn}: evader turns while moving at max linear velocity. With 70% probability the evader turns the *opposite* direction as the last turn, and with 30% turning the *same* direction as last time): $u^{\mathrm{e}} = [\bar{v}_x, \bar{v}_\theta]$ or $u^{\mathrm{e}} = [\bar{v}_x, \underline{v}_\theta]$

At the start of the episode, the evader uniformly at random chooses to turn left or right.

A.3.2 Random policy

This evader, $\pi^{\mathrm{e}}_{\mathrm{rand}}(t)$, randomly samples a motion primitive to apply for a fixed number of seconds, before re-sampling another motion primitive. The set of motion primitives $\mathcal{MP}$ are the Cartesian product of regularly discretized linear and angular velocities:

$$V_x = [\underline{v}_x, 0.5\underline{v}_x, 0, 0.5\bar{v}_x, \bar{v}_x], \quad V_\theta = [\underline{v}_\theta, 0.5\underline{v}_\theta, 0, 0.5\bar{v}_\theta, \bar{v}_\theta], \quad \mathcal{MP} := V_x \times V_\theta$$

At the start of each episode, the duration for applying a motion primitive is randomly sampled to be between $1 - 3$ seconds. When it is time to switch motion primitives, $\pi^{\mathrm{e}}_{\mathrm{rand}}(t) \equiv (v_x, v_\theta) \sim \mathrm{Unif}[\mathcal{MP}]$.

A.3.3 MARL policy

This evader, $\pi^{\mathrm{e}}_{\mathrm{marl}}(x^{\mathrm{rel}}_t)$, is trained to evade a pre-trained pursuer policy, equivalent to a single iteration of [19]. The evader is rewarded for maximizing the distance between the two agents at each timestep, and obtains a termination penalty upon capture:

$$\textit{Evasion: } r_t = 2 \cdot ||x^{\mathrm{e}}_t - x^{\mathrm{p}}_t||^2_2, \quad \textit{Capture: } r_T = -80 \cdot \mathbb{1}\{||x^{\mathrm{e}}_T - x^{\mathrm{p}}_T||^2_2 \leq 0.8\}. \tag{4}$$

The evader trains against a pre-trained, fully-observable pursuer policy: $\pi^*(x^{\mathrm{rel}}, z_t)$. This pursuer was originally trained against a random evader, $\pi^{\mathrm{e}}_{\mathrm{rand}}(t)$. Since the pursuer is pre-trained, its policy is already highly capable of capturing the evader making it difficult for the MARL agent to learn. To tackle this, we use a curriculum set at where at each fixed iteration, the pursuer speed in increased by 20%. The curriculum is set to $[1800, 2000, 2400, 2800, 3100, 3800]$ iterations.

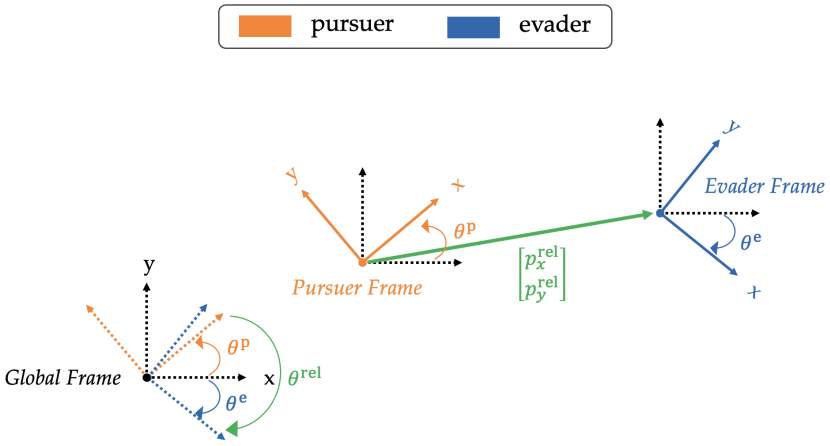

Figure 8: Global and relative coordinate system with respect to the pursuer's coordinate frame.

A.4 Zero-sum game policies: pursuer and evader

For one of our candidate fully-observable supervisor policies, we use an off-the-shelf zero sum differential game solver [17, 48]. It computes the optimal value of the two-player game, $V(x^{\mathrm{rel}}, t)$ at each time $t \in [0, T]$ via solving the Hamilton-Jacobi Isaacs Partial Differential Equation, whose solution yields optimal game-theoretic policies for both the pursuer and the evader. Each agents' state is their global position and their heading angle: $x^i := [p^i_x, p^i_y, \theta^i]^\top, \quad i \in \{\mathrm{p}, \mathrm{e}\}$ (see Fig. 8). Each agent controls their linear and angular yaw velocity: $u^i := [v^i_x, v^i_\theta] \in \mathcal{U}^i$ and the dynamics evolve via a unicycle model:

$$\begin{bmatrix} \dot{p}^i_x \\ \dot{p}^i_y \\ \dot{\theta}^i \end{bmatrix} = \begin{bmatrix} v^i_x \cos(\theta^i) \\ v^i_x \sin(\theta^i) \\ v^i_\theta \end{bmatrix}. \tag{5}$$

Each agent's policy is defined in their respective relative coordinate system (see Fig. 8 for relative coordinate system in pursuer's body frame). From the perspective of the pursuer, the relative state $x^{\mathrm{rel}} := [p^{\mathrm{rel}}_x, p^{\mathrm{rel}}_y, \theta^{\mathrm{rel}}]^\top$ consists of the relative xy-position and heading:

$$\theta^{\mathrm{rel}} := \theta^{\mathrm{e}} - \theta^{\mathrm{p}}, \quad \begin{bmatrix} p^{\mathrm{rel}}_x \\ p^{\mathrm{rel}}_y \end{bmatrix} := \begin{bmatrix} \cos(\theta^{\mathrm{p}}) & \sin(\theta^{\mathrm{p}}) \\ -\sin(\theta^{\mathrm{p}}) & \cos(\theta^{\mathrm{p}}) \end{bmatrix} \begin{bmatrix} p^{\mathrm{e}}_x - p^{\mathrm{p}}_x \\ p^{\mathrm{e}}_y - p^{\mathrm{p}}_y \end{bmatrix} \tag{6}$$

The continuous-time relative dynamics of the evader in the pursuer's body frame is defined by:

$$\begin{bmatrix} \dot{p}^{\mathrm{rel}}_x \\ \dot{p}^{\mathrm{rel}}_y \\ \dot{\theta}^{\mathrm{rel}} \end{bmatrix} = \begin{bmatrix} -v^{\mathrm{p}}_x + v^{\mathrm{e}}_x \cos(\theta^{\mathrm{rel}}) + v^{\mathrm{p}}_\theta p^{\mathrm{rel}}_y \\ v^{\mathrm{e}}_x \sin(\theta^{\mathrm{rel}}) - v^{\mathrm{p}}_\theta p^{\mathrm{rel}}_x \\ v^{\mathrm{e}}_\theta - v^{\mathrm{p}}_\theta \end{bmatrix} := f(x^{\mathrm{rel}}, u^{\mathrm{p}}, u^{\mathrm{e}}). \tag{7}$$

Note that when solving the game, the dynamics models are assumed to be deterministic.

The game is solved over the same time horizon as the episode length in the Isaac Gym simulator: $T = 20s$. The pursuer's optimal policy is defined as:

$$\pi^{\mathrm{p}}_{\mathrm{game}}(x^{\mathrm{rel}}, t) := \arg \min_{u^{\mathrm{p}} \in \mathcal{U}^{\mathrm{p}}} \max_{u^{\mathrm{e}} \in \mathcal{U}^{\mathrm{e}}} \nabla V(x^{\mathrm{rel}}, t)^{\top} f(x^{\mathrm{rel}}, u^{\mathrm{p}}, u^{\mathrm{e}}), \tag{8}$$

The evader's optimal policy is defined similarly with two changes: 1) the relative state x^{rel} is of the pursuer's state in the evader's body frame, and 2) the $\min$ and the $\max$ are swapped in Eq. 8.

$$\pi^{\mathrm{e}}_{\mathrm{game}}(x^{\mathrm{rel}}, t) := \arg \max_{u^{\mathrm{e}} \in \mathcal{U}^{\mathrm{e}}} \min_{u^{\mathrm{p}} \in \mathcal{U}^{\mathrm{p}}} \nabla V(x^{\mathrm{rel}}, t)^{\top} f(x^{\mathrm{rel}}, u^{\mathrm{p}}, u^{\mathrm{e}}). \tag{9}$$

Note that because the joint dynamics are affine in the pursuer and evader controls, the value of the game from the pursuer and evader's perspective is identical (also known as *Isaacs' condition* [14]).

Supervising the partially-observable policy. Recall that our approach uses supervision from the fully-observable policy. The partially-observable policy's intent estimate, $\hat{z}_t$, is supervised with the teacher's intent encoding z_t and also the pursuer actions, u^{p}_t, are supervised with the teacher policy actions, $u^{\mathrm{p},*}_t$. However, the game-theoretic policy does not generate an explicit latent state z_t the way that the MARL and Random policy architectures do. Thus, we only supervise partially observable policy's with the game theory policy $\pi^{\mathrm{p},*}(x^{\mathrm{rel}}_t, t)$ at each timestep and we update the latent intent encoder $\mathcal{E}$ and the policy π^{p} weight's. In the continual learning of Sec. 5.3 we follow this same paradigm: finetune the weights of the encoder and the policy from just action supervision. Otherwise the architecture of the partially observable student policy is identical (LSTM for encoding with 3-layer MLP for action network).

A.5 Kalman Filter

We use a linear state transition model,

$$x^{\mathrm{rel}}_{t+\delta t} = A x^{\mathrm{rel}}_t + B u^{\mathrm{p}}_t + w$$

which ignores the role of the evader (i.e., $u^{\mathrm{e}}_t \equiv 0$) during the the prediction step, and a linear observation model,

$$y_t = H x^{\mathrm{rel}}_t + v.$$

Note that while the Kalman filter ignores the motion of the evader, the student policy does not. Indeed, the student policy uses a history of relative states to predict the future evader's motion (Fig. 2). We intentionally ignore the evader's motion in the Kalman filter prediction step to waive the requirement for a velocity estimator (which can be cumbersome to obtain and noisy in the real world). While this makes filtering imperfect, we empirically found that the learned policy can cope with the resulting inaccuracies.

The state transition model assumes the relative state evolves via a single integrator model, assuming $u^{\mathrm{p}} := [v_x, v_y, v_\theta]$, which directly influence the time derivatives of the relative x, y and θ states. The quadruped we used in the experiments cannot control v_y directly, so at deployment time this entry of the control input is set to zero. The process covariance $w \sim \mathcal{N}(0, Q)$, the measurement covariance $v \sim \mathcal{N}(0, R)$, initial error covariance matrix P_0, observation matrix H, and state transition matricies are:

$$Q = 0.01 \cdot I_{3\times 3}, \quad R = \mathrm{diag}(0.2, 0.2, 0.1), \quad P_0 = I_{3\times 3}, \quad H = I_{3\times 3}, \tag{10}$$

$$A = I_{3\times 3}, \quad B = \mathrm{diag}(-\delta t, -\delta t, \delta t), \quad \delta t = 0.2 \tag{11}$$

A.6 Real-world experiments

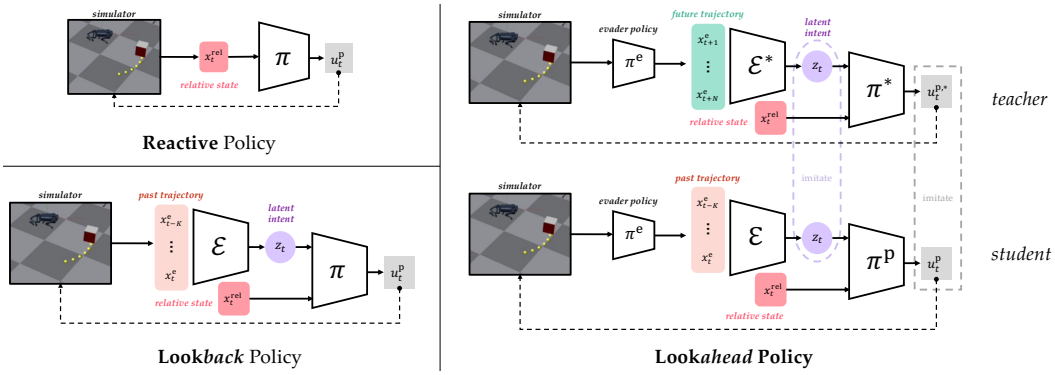

Figure 9: Policy architectures for reactive (up left), lookback (low left) and lookahead policy (right).

We use an Unitree A1 robot and a Unitree Go1 robot for experiments. The policy controls the linear speed of the robot on the x-axis in the body frame, i.e., in the forward direction (v_x), and the angular yaw velocity (v_θ). Such high-level commands are executed by an off-the-shelf low-level policy [4]. We use a Zed 2 camera for object detection in 3D. The object detection pipeline and the Kalman filter run onboard in a Jetson Tx2, while the low-level controller runs on an Intel UP Board. Communication happens through ROS using the implementation of [4].

Our policy only controls the pursuer Unitree A1 robot, while the evader is either a human or a Go1 robot teleoperated by a human. In both cases, the evader policy is neither pre-scripted nor explicitly strategic. The evader tries to avoid the pursuer with short-horizon evasion maneuvers, but it is not strategic in a long horizon to keep the interaction spatially constrained. We switch between tracking a person and another robot by changing the tracking class from PERSON to ANIMAL in the ZED 2 camera configs. We assume that only a single evader is present in the scene; therefore, we do not keep track of object identity. We empirically find that the ANIMAL class has more mis-detections than the PERSON class since the visual features of the quadruped differ from a real dog. Still, the detection performance was sufficient for continuous interaction.

A.7 Section 5.1 details: Reactive, Lookback, Lookahead policies

All policies use the same simulation setup as in Sec. A.2. The main differences are in the inputs and in the latent intent learning. Fig. 9 visualizes the policy architectures for the **reactive**, **lookback** and **lookahead** policies. We also compare training time of our **lookahead** approach–which uses future trajectory information as privileged info to learn the latent intent–to a **lookback** policy which must estimate the latent intent only from a history of relative states via an LSTM. We see that future trajectory supervision enables 10 times faster training compared to the **lookback** policy (Fig. 10) as well as higher overall reward.

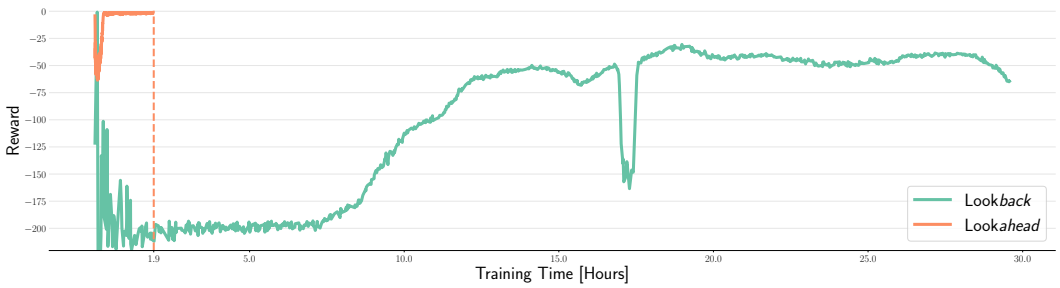

Figure 10: The lookahead policy (orange) converges to a higher reward 10 times faster than the lookback policy (green)

A.8 Note on directly learning a strategic pursuer policy with PPO

We tried to directly learn a policy with partial observations, $\pi(\hat{x}_t, \Sigma_t)$, using PPO. However, our policies never got a performance better than random. We believe the main reason to be the difficulty of the optimization problem. Indeed, such a policy needs to learn at the same time: (1) strategy, i.e., learning to predict where the evader will be, (2) control, i.e., learning to move towards a desired location on the x-y plane, and (3) estimation, i.e., learning to reason over the noisy states from the Kalman filter to get an estimate of the real position of the evader. We empirically found that PPO fails to do all these tasks simultaneously. However, we found that PPO could easily solve (1) and (2) if provided with a short-horizon future trajectory of the evader: this corresponds to our fully-observable training. To reduce sample complexity, we learn (3) using distillation learning, even though it could as well be trained with a further stage of reinforcement learning. An interesting venue for future work is to see how to learn such policy directly.