SAFER AUTONOMOUS DRIVING IN A STOCHASTIC, PARTIALLY-OBSERVABLE ENVIRONMENT BY HIERARCHICAL CONTINGENCY PLANNING

Ugo Lecerf *
Renault Software Labs
EURECOM

Christelle Yemdji Tchassi
Renault Software Labs

Pietro Michiardi
EURECOM

ABSTRACT

When learning to act in a stochastic, partially observable environment, an intelligent agent should be prepared to anticipate a change in its belief of the environment state, and be capable of adapting its actions on-the-fly to changing conditions. As humans, we are able to form contingency plans when learning a task with the explicit aim of being able to correct errors in the initial control, and hence prove useful if ever there is a sudden change in our perception of the environment which requires immediate corrective action.

This is especially the case for autonomous vehicles (AVs) navigating real-world situations where safety is paramount, and a strong ability to react to a changing belief about the environment is truly needed.

In this paper we explore an end-to-end approach, from training to execution, for learning robust contingency plans and combining them with a hierarchical planner to obtain a robust agent policy in an autonomous navigation task where other vehicles' behaviours are unknown, and the agent's belief about these behaviours is subject to sudden, last-second change. We show that our approach results in robust, safe behaviour in a partially observable, stochastic environment, generalizing well over environment dynamics not seen during training.

1 INTRODUCTION

Developing a controller for AVs to be used in a real-life navigation environment poses a number of challenges including perception, and modeling environment dynamics (Michelmore et al., 2020; Yurtsever et al., 2020). The limitations in vehicle sensors, as well as the stochastic nature of inter-vehicle interactions, introduce a level of uncertainty in autonomous navigation tasks which hinders the ability to control an agent from the standpoint of both safety, and adequate performance. Sensor information, as well as other drivers' intentions, are subject to sudden change and a robust navigation algorithm must be able to safely adapt on-the-fly to these unforeseen changes.

Stochastic environments can be modeled as Partially-Observable Markov Decision Processes (POMDPs) (Kochenderfer et al., 2015). Solving POMDPs is challenging, yet possible, for example using methods combining learning and planning such as Brechtel et al. (2014); Hoel et al. (2020a). Having access to a model of the environment dynamics allows us to use planning algorithms, such as tree-search methods (Browne et al., 2012), alongside learning to both increase sample efficiency, and have access to a better representation of the environment's state-space structure (Machado et al., 2017). In cases where planning is possible it is much easier for an agent to predict the outcome of its actions and hence better adapt to eventual changes in the state-space (McAllister and Rasmussen, 2017). However, a change in the values of the stochastic model parameters (e.g. a change in a vehicle's sensor accuracy, unplanned scene obstruction, or simply an unforeseen behaviour from another vehicle) may induce a sharp drop in the agent's performance due to its inability to generalize well to new environment parameters.

An approach for tackling this issue has been to design controllers to have a high capacity for generalization: instead of attempting to learn over the entire space of possible environments, we design

*correspondance to: ugo.lecerf@renault.fr

an algorithm to act well enough over the whole space, having trained only on a tractable subset of environment configurations. One such example is known as contingency planning, whereby contingency plans are put in place to specifically counter stochastic environments in which failing to prepare for possible problems in advance can be an expensive mistake (Pryor and Collins, 1996).

Our contribution. In this work we make the following contributions:

- We introduce a method for learning a contingency policy concurrently to the optimal policy during training, such that the former is well-adapted to serve as a contingency plan.
- We combine learned policies with a high-level model based controller, and experimentally show through an autonomous navigation task that our approach is able to achieve a much safer agent performance in the case of a stochastic environment, while sacrificing a minimal amount of performance.

2 BACKGROUND

Deep reinforcement learning (RL) is a method for learning control algorithms in a weakly-supervised manner (by means of a reward signal). Implementing an RL algorithm requires us to model the environment as a Markov Decision Process (MDP), defined by the following elements: A finite set of states $s \in \mathcal{S}$, indexed by the timestep at which they are encountered: s_t. A finite set of actions $a \in \mathcal{A}$, also indexed by their respective timesteps: a_t. A transition model $\mathcal{T}(s, a, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, representing the probability of passing from s to s' after taking action a_t, $P(s_{t+1} = s' | s_t = s, a_t = a)$. An immediate reward function $R(s, a, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ and discount factor $\gamma \in (0, 1]$, controlling the weight in value of states further along the Markov chain. A POMDP is further augmented by an observation model $\mathcal{O}$, when we no longer have access to the true state s_t, but to an observation thereof $o_t \sim \mathcal{O}(s_t)$.

Actions in the POMDP are taken by a policy $\pi : \mathcal{O} \to \mathcal{A}$ mapping observations to actions. The value of an observation under a policy π, is given by the value function $V^\pi : \mathcal{O} \to \mathbb{R}$, which represents the expected future discounted sum of rewards, if policy π is followed from the true state s, of which we have an observation $o \sim O(s)$:

$$V^\pi(o) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})\right], \tag{1}$$
$$s_0 = s, o_t \sim O(s_t), a_t = \pi(o_t), s_{t+1} \sim \mathcal{T}(s_t, a_t).$$

Actions are chosen by the policy π, so as to maximize the action-value function $Q^\pi : \mathcal{O} \times \mathcal{A} \to \mathbb{R}$ which assigns values to actions according to the value of the states that are reached under π:

$$Q^\pi(o_t, a_t) := \mathbb{E}_{s_{t+1}}\left[R(s_t, a_t, s_{t+1}) + \gamma \cdot V^\pi(o_{t+1})\right], \tag{2}$$
$$s_{t+1} \sim \mathcal{T}(s_t, a_t), o_{t+1} \sim O(s_{t+1}).$$

We use Q^π in order to define the optimal policy, which we denote π^*, as the policy taking actions that maximizes Q^π: $\pi^*(o) := \arg\max_{a \in \mathcal{A}} Q^{\pi^*}(o, a)$. Equation (2) highlights that the action-value function is a sort of one-step look-ahead of the value of the next possible state s_{t+1}, in order to determine the value of actions in the current observation o_t. We denote the sequence of observations (trajectory through observation-space) visited by a policy π during an episode as:

$$\tau_\pi := \{o_t\}_{t \in [0,T]}, \tag{3}$$
$$s_0 \in \mathcal{S}, o_t \sim O(s_t), a_t = \pi(o_t), s_{t+1} \sim \mathcal{T}(s_t, a_t).$$

2.1 TRAINING AN OFF-POLICY DEEP Q NETWORK

The off-policy Q-learning algorithm (Mnih et al., 2015) is well-suited to learning a policy's Q-function in the discrete action space which we consider. Training off-policy allows us to train on past transitions stored in a replay buffer (Schaul et al., 2016) which both stabilizes training, and increases sample efficiency since a single transition sample may be used during multiple training steps. We perform a stochastic gradient descent on the MSE between the Q-estimate for the current step, and the discounted 1-step 'lookahead' Q-estimate summed with the transition reward:

$$\mathcal{L}(o_t, a_t, r_t, o_{t+1}) = \Big(Q_\theta(o_t, a_t) - \Big[r_t + \gamma Q_{\theta'}(o_{t+1}, \arg\max_{a'} Q_\theta(o_{t+1}, a')\Big]\Big)^2,$$

where θ are the current parameters for the Q-network, and θ' are parameters that are 'frozen' for a certain amount of steps. This is known as double Q-learning using target networks (Van Hasselt et al., 2016) and reduces the variance of gradient updates.

3 Related Work

Hierarchical RL. Hierarchical reinforcement learning (Dayan and Hinton, 1993; Sutton et al., 1999) is a promising approach for helping RL algorithms generalize better in increasingly complex environments. Several works apply the hierarchical structure of control to navigation tasks, which lend themselves well to modular controllers (Fisac et al., 2018). Approaches such as Andreas et al. (2017); Nachum et al. (2018) use sub-task or goal labelling in order to explicitly learn policies that are able to generalize through goal-space. Our approach differs where we don't wish to learn different goals, rather striving to attain the same goal under modified environment conditions, moreover, we aim to learn alternative strategies without having to subtask or goal labels. Works such as Machado et al. (2017); Zhang et al. (2021) also have the same aim of self-discovering strategies to be used in a hierarchical controller, though our method differs by targeting the value function of the agent, through additional reward terms. Similar to our approach, Cunningham et al. (2015) uses a form of voting through policy simulation, though our work also integrates training the contingency policy with robustness to environment modifications in mind.

Contingency planning. Some approaches use an estimation of the confidence of the actions proposed by an agent's policy (Bouton et al., 2019; Clements et al., 2019; Hoel et al., 2020b) to determine whether or not an agent's policy is sufficiently good in the current environment state. When this is not the case, control is typically given to a separate, often open-loop controller looking to mitigate any possible negative behaviour if failure cannot be avoided otherwise (Dalal et al., 2018; Filos et al., 2020). One issue with open-loop contingency plans – or any open-loop policy in general – is that they do not take into account the closed-loop nature of most real-world environments, whose dynamics are dependent on the actions of the agent and may themselves fail if not implemented carefully. Rhinehart et al. (2021); Killing et al. (2021) tackle the problem of high environment uncertainty by prioritizing information gathering if the agent is too uncertain about its policy's outcome. These approaches choose to approach by default with caution, if ever there is missing or uncertain information in the agent's input space. Kumar et al. (2020) seeks to learn a set of policies which are collectively robust to changes in environment dynamics, through the use of latent-conditioned policies (Eysenbach et al., 2019). Our approach is similar, though we use an explicit metric on trajectories to ensure diversity in contingency behaviour rather than a learnt descriminator function. Furthurmore we train our contingency plan to perform well when environment parameters change during the execution phase, which couples well with an on-line high-level controller.

4 Learning Contingency Policies

Problem statement. We wish to learn, on one hand, π^*, the optimal policy in our given environment, and on the other, π_1, a contingency policy able to navigate more safely through the environment, at the cost of performance, if ever there is high uncertainty linked to following the optimal policy.

4.1 Contingency Policies

Given the nature of a contingency plan, it must aim to exploit trajectories through different areas of state-space than trajectories from the optimal policy. Since we consider cases where typically local uncertainties in the navigation task disrupt the performance of the optimal policy, in order for the contingency plan to stay viable it should have trajectories through sufficiently different areas of state-space to avoid areas of high uncertainty. For example a contingency plan to high-speed trajectories, will often be more conservative and favor low-speed policies.

Using reward augmentation. Our approach for learning π^* and π_1 concurrently, is to augment the reward function by a term which we will refer to as a *trajectory penalty*, R^{pen}, which is propor-

tional to the similarity in expected observation-space trajectories between two policies. The aim of augmenting the reward in this way is for the policy of the contingency agent to be incentivized to solve the environment through different areas of state-space than the optimal policy. With this aim in mind we define a metric on agents' observation trajectories:

$$\mathcal{M}(\tau_{\pi_1}, \mathbb{E}\left[\tau_{\pi^*}\right]) := \int \left| \nu\left(\phi\left(\tau_{\pi_1}\right)\right) - \nu\left(\phi\left(\mathbb{E}\left[\tau_{\pi^*}\right]\right)\right) \right| d\phi(o), \tag{4}$$

where ϕ is an observation-feature function, and ν is the density function over observation features. It is possible to remain the most general possible with $\phi(o) = o$, however with some domain knowledge we can modify ϕ to retain only the features which best define the 'distinctness' of an agent's trajectory.

Designing the penalty term R^{pen} to be inversely proportional to the trajectory metric with respect to a reference agent, we can define it as:

$$R^{pen}_{\pi^*}(\tau_{\pi_1}) := -\frac{\alpha}{\mathcal{M}(\tau_{\pi_1}, \mathbb{E}\left[\tau_{\pi^*}\right]) + \delta}, \tag{5}$$

where α and δ determine the relative weight of the trajectory penalty, with respect to the regular reward function R. A lower value for α will hardly penalize the contingency policy for having a similar observation distribution to the reference agent, whereas higher weighting will make it seek a highly different trajectory, disregarding the original objective of the task given by the regular reward function. The value function for π_1 becomes:

$$V^{\pi_1}(o) := \mathop{\mathbb{E}}_{\tau \sim \pi_1}\left[R(\tau) + R^{pen}_{\pi^*}(\tau)\right], \quad \tau|_{t=0} = o,$$

where $\tau \sim \pi_1$ is the expected observation-space trajectory under π_1. We denote Π, the set of all policies (optimal and contingency) learned in the POMDP. Stochastic parameters for the POMDP are stored in a vector $\boldsymbol{\mu}$. Appendix A.1 presents the pseudo-code for our proposed algorithm.

Contingency policy's domain. When building a hierarchical control framework, much of the generalization capabilities come from the correct identification of the environment parameters, and then executing the relevant policy (Pateria et al., 2021)). When a new objective is introduced to the agent or environment parameters change causing new environment dynamics, making the optimal policy unable to perform well, we must ensure that the contingency policy is able to compensate for the initial actions taken by the optimal policy. When the observation-space gets more complex, we may no longer assume that the learned contingency policy has sufficient knowledge over its domain such that it is able to correctly control the agent from any such 'hand-off' state.

This is linked to the exploration trade-off made during training, where we explicitly limit an agent's exploration of the available state-space in the interest of making learning tractable (Sutton and Barto, 2018)). Due to limited exploration by design, this limits the domain on which a policy's Q-function is well-learnt. This isn't an issue for navigating from the starting state to each objective, however there will be some areas of state-space, less sampled by the agent, where the Q-function will have a greater error. This may be equivalently seen as a lack of exploration and insufficient training of the RL agent, however in more complex environments time and computing constraints prevent us from training the Q-function over the policy's entire domain.

To encourage good contingency policy performance in so-called 'hand-off' states, our approach is to modify the initial state distribution for the contingency agent. We achieve this by sampling states from the optimal agent's replay buffer, and adding them to the contingency agent's initial state distribution. Given the initial state distribution $p(s_0)$, we define the new initial state distribution for the contingency agent as: $p^{\pi_1}(s_0) = (1-\beta)p(s_0) + \beta p(\tilde{s}_{\pi^*})$, where $p(\tilde{s}_{\pi^*})$ is the uniform distribution over states in π^*'s replay buffer, and $\beta \in [0, 1]$ is a parameter controlling the ratio of initial states sampled from the regular task intialization to ones sampled from the optimal's replay buffer.

Moreover, our aim with this replay buffer intialization is to help the robustness of the contingency agent over the most probable 'hand-off' points with the optimal agent when the latter is unable to handle the current environment parameters. To this end, we may further use domain knowledge to constrain the states sampled from $p(\tilde{s}_\pi)$ such that they best represent these possible 'hand-off' points. Appendix A.2 includes more details about the replay buffer initialization.

4.2 HIERARCHICAL CONTROLLER

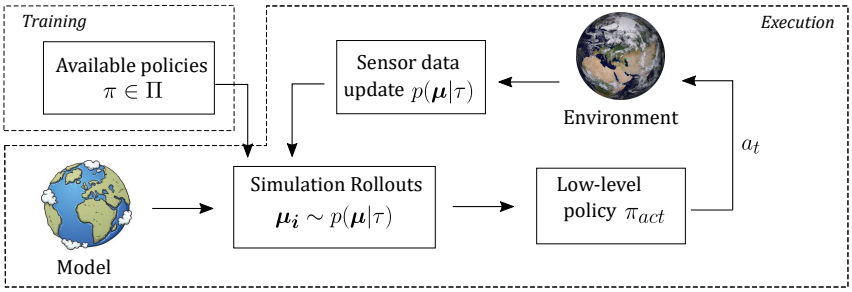


Figure 1: Structure of hierarchical controller composed of available policies and model-based planner (High-level policy selection).

Contingency plans alone are not sufficient: a higher-level controller is required, capable of selecting the best policy with respect to the current environment observation. In our approach, we combine the learned optimal and contingency policies with a model-based planner, in order to increase the robustness and safety of the acting agent with respect to environment uncertainty. Figure 1 shows the structure of the hierarchical controller, whose high-level policy selection is in fact the planner module responsible for estimating the safety of each of the available policies $\pi \in \Pi$, and selecting the one with the lowest estimated chance of failure. A pseudo-code description for the hierarchical controller is provided in appendix A.3.

4.3 PLANNER

The planner's role is to estimate the chance of failure for each $\pi \in \Pi$ available to the controller. The approach we retain for our purposes is straightforward: we perform roll-outs over the set of possible environment parameters $\boldsymbol{\mu}$ for each π, and return an estimated failure rate based on those roll-outs which will define which policy is selected by the planner. This process estimates the probability of policy failure given the current belief about environment stochastic parameters $\boldsymbol{\mu}$. Let $P(s_{fail}|\pi, \boldsymbol{\mu})$ denote the probability that policy π will fail for a given $\boldsymbol{\mu}$. The planner's goal is to select a policy according to:

$$\pi = \arg\min_{\pi \in \Pi} P(s_{fail}|\pi, \boldsymbol{\mu}^*),$$

where $\boldsymbol{\mu}^*$ are the true (unknown) parameter values. We may approximate it by sampling from a probability density function conditioned on the history of agent observations over the course of the episode $\boldsymbol{\mu}_i \sim p(\boldsymbol{\mu}|\tau)$. More details on sampling $\boldsymbol{\mu}_i$'s are given in appendix A.4. Let $C(\pi, \boldsymbol{\mu}_i) \in [0, 1]$ represent whether or not there is a failure (collision) from policy π, after roll-out. Then:

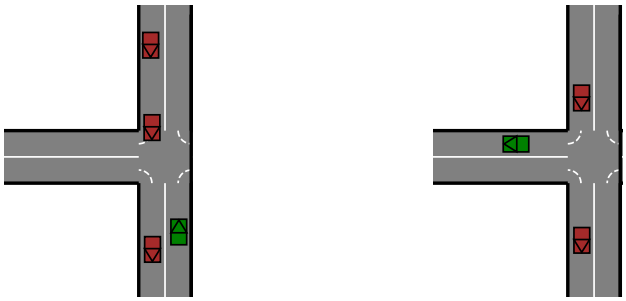
$$P(s_{fail}|\pi, \boldsymbol{\mu}^*) \approx \frac{1}{M} \sum_{i=1}^{M} C(\pi, \boldsymbol{\mu}_i),$$

for M samples of $\boldsymbol{\mu}_i$. We note that the quality of the approximation relies on how well the $\boldsymbol{\mu}_i$'s are sampled. The closer they are to $\boldsymbol{\mu}^*$, the better the planner will be able to estimate the true probability of failure for each policy. We use a simple approach consisting in eliminating $\boldsymbol{\mu}_i$'s from the sampling pool, if targets are observed behaving in contradiction to the considered environment parameters (based on vehicle speed, in our navigation task).

5 EXPERIMENTS

5.1 SIMULATION ENVIRONMENT

We evaluate our approach in a common autonomous navigation task (Brechtel et al., 2014; Hubmann et al., 2018; Bouton et al., 2019; Bernhard et al., 2019; Rhinehart et al., 2021)). Figure 2 shows two frames of the environment with oncoming vehicles in the intersection. In navigation tasks, the controllable agent is usually referred to as the ego whereas the other vehicles are referred



(a) $t = 3$s (b) $t = 5$s

Figure 2: Navigation task: ego makes a left turn across the intersection with oncoming traffic. Target vehicles may either be aggressive (i.e. disregarding presence of ego in intersection) or cooperative (i.e. slowing down if ego is close to intersection point).

to as targets. In this task the ego must adjust its speed in order to pass through the intersection without colliding with any of the oncoming targets. The optimal policy with respect to the reward function (6) is to pass through the intersection as fast as possible while avoiding collisions. We expect a contingency policy to be useful in this scenario, when there is a sudden change in estimated target behaviour by the planner and the ego will have to either slow down to let an aggressive target through, or speed up if it seems the target is slowing down too much which may also cause a collision.

Ego and target behaviours. The ego's action space is a set of longitudinal acceleration values: $a \in \mathcal{A} = \{-4, -2, -1, 0, 1, 2\}$ m/s^2. Target vehicles may have either a cooperative, or aggressive style of driving, this behaviour being unobservable by the ego. An aggressive target will speed through the intersection disregarding the presence of the ego vehicle, whereas a cooperative target will slow down if ever the ego vehicle approaches the intersection, although it will not come to a complete stop if ever ego halts in the middle of the road.

Reward function. To penalize collisions and encourage faster episode termination, the step-reward r_t is set-up as follows per time step t:

$$r_t = \begin{cases} -5 & \text{if collision} \\ -0.1 & \text{otherwise} \end{cases}. \tag{6}$$

Stochastic environment parameters. The environment dynamics depend on the targets' behaviours. The behaviour for each target vehicle is a random variable B_i representing degree of aggressiveness, which are collected in $\boldsymbol{\mu}$ where $\boldsymbol{\mu}|_i = B_i$ for $i \in [1, N]$ for N targets. In practice we use either $b_i = 0$ for a cooperative target, or $b_i = 1$ for an aggressive target.

5.2 Implementation Details

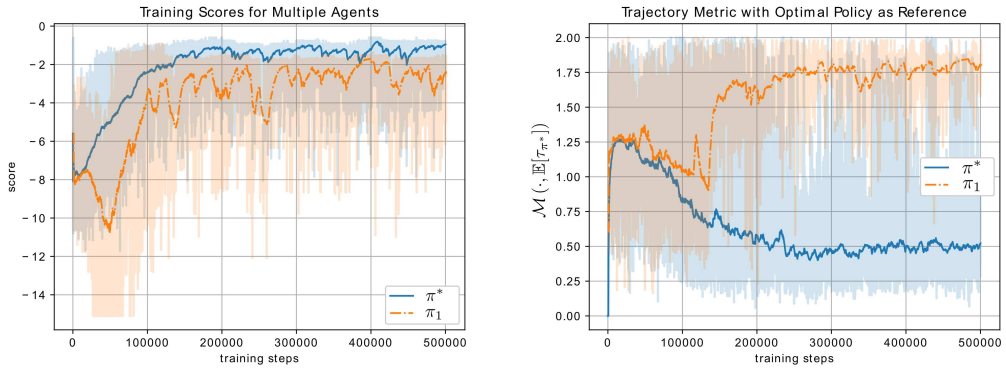
Target behaviours. During training we set $b_i = 1, \ \forall i \in [1, N]$. Both optimal and contingency policies are trained on these environment parameters.

Trajectory penalty. In practice we replace the expectation operator $\mathbb{E}[\tau_\pi]$ in (4) by the mean value over the last 100 samples of π^*'s replay buffer. In this environment we use the speed of the ego vehicle as an observation feature $\phi(o) = \dot{x}_{ego}$. Trajectory penalty scaling factors used are: $\alpha = 3, \delta = 0.1$, which are fixed by a rough initial sweep.

Adjusting the return values for different initial states. We use a modified initialization for the contingency agent to increase its capabilities for compensating erroneous behaviour from a previous policy. We initialize 50% of the contingency agent's episodes in this way, during the entire training phase. Forcing the contingency policy to start the episode in states sampled from the optimal policy's replay buffer will incur greater trajectory penalties due to the higher trajectory proximity. However, we do not need to to add an explicit compensation term: the contingency agent is still learning its true action-value function Q^{π_1}. We take into consideration that the trajectory penalty term R^{pen} will reduce the mean computed performance score, so it is important to look at the training score

and trajectory penalties separately when comparing performance from both policies. To compensate for varying episode length due to a different initial state, the contingency policy is attributed the cumulated rewards corresponding to the sampled initial state from the replay buffer.

6 RESULTS



(a) (b)

Figure 3: (a) Training scores for both agents during training phase. This figure does not take into account the trajectory penalties R^{pen} for π_1, only the regular rewards R. (b) Evolution of computed trajectory metric term $\mathcal{M}(\cdot, \mathbb{E}[\tau_{\pi^*}])$ for optimal and contingency policies. Although computed for both policies, the resulting trajectory penalty is only attributed to π_1.

Training a contingency policy with replay buffer. Figure 3a shows the scores achieved by both the optimal and contingency policies during training. We can see π^* learning until reaching its optimal performance score after approximately 200k training steps. π_1's optimization objective depends on an additional R^{pen} term, which is not stable until the observation-density in the replay buffer of π^* becomes approximately static. This is why we see the initial training curve of π_1 lag behind that of π^*. Once π^*'s expected observation trajectory, $\mathbb{E}[\tau_{\pi^*}]$, has a low enough variance then π_1 can make some meaningful progress towards its contingency objective. Looking at π_1's training curve in our case this appears to happen around the 50k step mark. Even once π_1 has reached its top performance, it continues to be more jittery than π^*. This is due to the sampled replay buffer initializations.

Trajectory metric during training. Figure 3b shows the evolution of the trajectory metric (4). The values for π^* are shown for comparison, but not actually included in its reward during training. The shape for π^* is to be expected, as the latest trajectory will initially be different from the mean trajectory sampled from its own replay buffer. However as its parameters θ converge and exploration is weaned away, the episodic trajectory will get closer to the mean trajectory sampled from the replay buffer.

We notice that initially both policies have similar trajectories τ. This is due to the high degree of initial random exploration. There is a sharp rise in the metric for π_1 around the 150k mark, and we may deduce that at this point the most recent trajectories τ_{π_1} are sufficiently different to the observations present in π^*'s replay buffer such that the metric between the two increases. We may conclude that this is where the contingency policy is converging to a contingency plan that has a different trajectory from that of the optimal policy (notably lower-speed trajectories).

Hierarchical planner safety performance. To evaluate the overall performance of our hierarchical controller, we compute the number of successes vs. failures on the navigation task, over a range of environment parameter values $\boldsymbol{\mu}$ for different agents. Table 1 compiles the results for each tested controller, over an environment parameter sample size of $M = 200$.

From our results we clearly deduce that a single agent has a hard time generalizing to new target behaviours, even though it may have achieved optimal performance within its training environment. Although π^* has the highest average score in cases when it does not fail, it does not generalize well

Table 1: Controller performances for a range of agents. Success rate gives ratio of number of successes vs. failures over all sampled environment configurations. Average score gives us the mean of scores obtained in cases where the agent does not fail. Results are obtained by averaging 4 runs over the same random seed.

Controllers	**Success rate**	**Average Score**
π^*	0.496	-1.200
$\pi_\mathbf{1}$ without replay buffer init.	0.885	-1.500
$\pi_\mathbf{1}$ with replay buffer init.	0.954	-1.466
H-control without replay buffer init.	0.890	-1.380
H-control with replay buffer init.	$\mathbf{1.000}$	$\mathbf{-1.238}$

and performs very poorly in unseen environment instances. In our navigation task, the contingency policy that is learnt corresponds to the ego having a more conservative driving attitude; this explains why the success rate is higher for π_1 with respect to π^*, although the average score decreases due to sacrificing performance for safer behaviour. When adding the replay buffer initialization to π_1, the success rate further increases due to the increased ability of the contingency agent to generalize to greater areas of observation-space.

Compared to the individual policies, we expect the hierarchical controller to perform better, due to its access to a model-based planner combined with both policies. Interestingly, we find that the contingency policy with replay buffer initialization has a higher success rate than the hierarchical controller using π_1 without the initialization. This highlights the importance of generalization when designing safe, robust algorithms. Finally, we see that the highest success rate is achieved by our proposed approach of combining optimal, and robust contingency policies. More importantly, though success rates are similar with the single π_1 with replay buffer initialization, we are able to obtain a good performance score close to the optimal policy, due to π^* being available to the hierarchical controller. This demonstrates how our approach is able to ensure much safer behaviour in unseen environment configurations than a single policy, without sacrificing performance by being overly cautious.

7 Conclusion

In conclusion, we have presented an approach for learning multiple policies in an autonomous navigation task and adapting the approach to specifically learn a robust contingency policy, which when combined with a model-based planner, is able to increase the robustness of the agent with respect to stochastic environment parameters. In our intersection use-case, we are able to reach a rate of no collisions for any (sampled) environment configuration, even when we only had access to a single one of these configurations during training. We acknowledge that the planner module has a simple behavioural prediction for target vehicles, and although it is sufficient in our simulation environment to obtain good performance, better detection of the environment's stochastic parameters $\boldsymbol{\mu}$ will increase the robustness of the overall agent, and ultimately the effectiveness of having an available contingency policy.

One main advantage of this hybrid approach is the ability to separate performance and safety in an RL framework. Whereas using a single reward function and relying on reward engineering to obtain the correct behaviour can be arbitrary, in this case we are able to optimize for performance in a complex environment, and ensure a high level safety in unseen instances of environment dynamics without having to tweak performance and safety terms in a single reward function.

REFERENCES

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 166–175, 06–11 Aug 2017.

Julian Bernhard, Stefan Pollok, and Alois Knoll. Addressing inherent uncertainty: Risk-sensitive behavior generation for automated driving using distributional reinforcement learning. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 2148–2155, 2019.

M. Bouton, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer. Safe reinforcement learning with scene decomposition for navigating complex urban environments. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 1469–1476, 2019.

Sebastian Brechtel, Tobias Gindele, and Rüdiger Dillmann. Probabilistic decision-making under uncertainty for autonomous driving using continuous pomdps. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 392–399, 2014.

Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, pages 1–43, 2012.

William R. Clements, Benoît-Marie Robaglia, Bastien Van Delft, Reda Bahi Slaoui, and Sébastien Toth. Estimating risk and uncertainty in deep reinforcement learning. *arXiv Preprint*, 2019.

Alexander G. Cunningham, Enric Galceran, Ryan M. Eustice, and Edwin Olson. Mpdm: Multipolicy decision-making in dynamic, uncertain environments for autonomous driving. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1670–1677, 2015.

Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint*, 2018.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 5, pages 271–278. Morgan-Kaufmann, 1993.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *Proceedings of the International Conference on Learning Representations*, 2019.

Angelos Filos, Panagiotis Tigkas, Rowan Mcallister, Nicholas Rhinehart, Sergey Levine, and Yarin Gal. Can autonomous vehicles identify, recover from, and adapt to distribution shifts? In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 3145–3153, 2020.

Jaime F. Fisac, Eli Bronstein, Elis Stefansson, Dorsa Sadigh, S. Shankar Sastry, and Anca D. Dragan. Hierarchical game-theoretic planning for autonomous vehicles. In *2019 IEEE Internation Conference on Robotics and Automation (ICRA)*, pages 9590–9596, 2018.

Carl-Johan Hoel, Katherine Driggs-Campbell, Krister Wolff, Leo Laine, and Mykel J. Kochenderfer. Combining planning and deep reinforcement learning in tactical decision making for autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 5:294–305, 2020a.

Carl-Johan Hoel, Krister Wolff, and Leo Laine. Tactical decision-making in autonomous driving by reinforcement learning with uncertainty estimation. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 1563–1569, 2020b.

Constantin Hubmann, Jens Schulz, Marvin Becker, Daniel Althoff, and Christoph Stiller. Automated driving in uncertain environments: Planning with interaction and uncertain maneuver prediction. *IEEE Transactions on Intelligent Vehicles*, 3:5–17, 2018.

Christoph Killing, Adam Villaflor, and John M. Dolan. Learning to robustly negotiate bi-directional lane usage in high-conflict driving scenarios. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 8090–8096, 2021.

Mykel J. Kochenderfer, Christopher Amato, Girish Chowdhary, Jonathan P. How, Hayley J. Davison Reynolds, Jason R. Thornton, Pedro A. Torres-Carrasquillo, N. Kemal Üre, and John Vian. *Decision Making Under Uncertainty: Theory and Application*. The MIT Press, 1st edition, 2015.

Saurabh Kumar, Aviral Kumar, Sergey Levine, and Chelsea Finn. One solution is not all you need: Few-shot extrapolation via structured maxent rl. In *Advances in Neural Information Processing Systems*, volume 33, pages 8198–8210, 2020.

Marlos C. Machado, Marc G. Bellemare, and Michael Bowling. A Laplacian framework for option discovery in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 2295–2304, 2017.

Rowan McAllister and Carl Edward Rasmussen. Data-efficient reinforcement learning in continuous state-action gaussian-pomdps. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Rhiannon Michelmore, Matthew Wicker, Luca Laurenti, Luca Cardelli, Yarin Gal, and Marta Kwiatkowska. Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 7344–7350, 2020.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

Ofir Nachum, Shixiang (Shane) Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys*, 54, 2021.

Louise Pryor and Gregg C. Collins. Planning for contingencies: A decision-based approach. *Journal for Artificial Intelligence Research*, 4:287–339, 1996.

Nicholas Rhinehart, Jeff He, Charles Packer, Matthew A. Wright, Rowan McAllister, Joseph E. Gonzalez, and Sergey Levine. Contingencies from observations: Tractable contingency planning with learned behavior models. In *2021 IEEE International Conference on Robotics and Automation*, pages 13663–13669, 2021.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *Proceedings of the International Conference on Learning Representations*, 2016.

Richard Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstaction in reinforcement learning. *Artificial Intelligence*, 1999.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access*, 8:58443–58469, 2020.

Jesse Zhang, Haonan Yu, and Wei Xu. Hierarchical reinforcement learning by discovering intrinsic options. In *International Conference on Learning Representations*, 2021.

A Implementation Details

A.1 Training Algorithm

Algorithm 1 gives an overview of the training regime used with the reward penalty, in order to learn contingency policies alongside the optimal. Notably, we attribute r^{pen} to the final sample at the end of the episode played by the contingency agent. $\mathbb{1}_{[\pi==\pi_1]}$ stands for the indicator function, indicating whether the current training agent is the contingency agent π_1, or not.

Both agents train concurrently, alternating training episodes. However the contingency agent π_1 starts its training only when the replay buffer of π^* is full. This is reflected in the training curves, though it is negligible with respect to the order of magnitude of total training samples. Computing the value for $R^{pen}_{\pi^*}(\tau_{\pi_1})$ before π^*'s expected trajectory memory buffer has converged to a stable value means that the MDP being solved by π_1 is initially greatly changing, though we did not investigate the effects of different training scheduling for the contingency agent in this work.

In the initial state distribution $p(s_0)$ during training, the ego always starts at the same position, whereas the initial positions of the target vehicles are randomized.

Algorithm 1 Training Contingency Policies

Init $\Pi = \{\pi^*, \pi_1\}$
while not converged **do**
  for $\pi \in \Pi$ **do**
    $s_0 \sim p^{\pi}(s_0)$ ▷ init. episode state
    $o_0 \sim \mathcal{O}(s_0)$
    $\tau_\pi = \{o_0\}$
    while episode not terminated **do** ▷ play episode
      $a_t = \pi(o_t)$
      $s_{t+1} \sim \mathcal{T}(s_t, a_t)$ ▷ environment step
      $r_t = R(s_t, a_t, s_{t+1})$ ▷ step reward
      $o_{t+1} \sim O(s_{t+1})$
      $\tau_\pi = \tau_\pi \cup \{o_{t+1}\}$
    end while
    $r^{pen} = R^{pen}_{\pi^*}(\tau)$ ▷ compute reward penalty according to (5)
    $r_T = r_T + \mathbb{1}_{[\pi==\pi_1]} r^{pen}$ ▷ attribute r^{pen} only to contingency agent
    for $t \in [0, T]$ **do**
      Memory$(\pi) \leftarrow (o_t, a_t, r_t, o_{t+1})$ ▷ store samples in agent replay buffer
    end for
  end for
end while

A.2 Replay Buffer Initialization

We mention in section 4.1 the need for a contingency plan to function well at potential 'hand-off' points, if the high-level controller switches to the contingency plan late in the episode. To increase the likelihood that $p(\tilde{s}_{\pi^*})$ well represents these states, using domain knowledge, we constrain it to have a uniform density only over states where the ego vehicle has not yet passed the intersection. This avoids the contingency policy learning to act once the ego has passed the intersection which is not useful in our use-case:

$$p(\tilde{s}_{\pi^*}) = \mathcal{U}(\tilde{\mathcal{S}}), \quad \tilde{\mathcal{S}} = \{s \in \text{Memory}(\pi^*) \mid s|_{x_{ego}} < x_{int}\},$$

where $s|_{x_{ego}} < x_{int}$ represents all states in which the ego has not yet crossed the intersection. In practice we only have access to observations of states in the memory buffer hence we map the sampled observations, which contain the ego's position, back onto environment states which would result in the sampled observation. Even though we're not assured to map back onto the exact same environment state as was encountered in the optimal agent's state-space trajectory, this nevertheless increases the contingency agent's ability to be robust to initial states sampled from the optimal agent's trajectory. The optimal agent π^* uses the standard environment initial state distribution: $p^{\pi^*}(s_0) = p(s_0)$.

In our experiments we found that a ratio of 50% between regular and optimal-replay-buffer episode initializations gave the best results. lower values tended to decrease the contingency agent's ability to function well at the 'hand-off' points, whereas higher values tended to overly impact the convergence of the contingency agent's parameters; $p(\tilde{s}_{\pi^*})$ adds a lot of variance in the state-space encountered by the contingency agent, and hence increases the complexity of correctly learning Q^{π_1} over this larger domain. Thus we use the value $\beta = 0.5$:

$$p^{\pi_1}(s_0) = \frac{1}{2}p(s_0) + \frac{1}{2}p(\tilde{s}_{\pi^*})$$

A.3 Hierarchical Controller Algorithm

Algorithm 2 shows how the hierarchical controller chooses between available policies. The idea is for it to choose the policy with the highest estimated safety in the environment, using estimates of the environment dynamics.

Algorithm 2 Executing Hierarchical Controller

$\Pi = \{\pi^*, \pi_1\}$ ▷ available policies
Init o_0 ▷ initial env observation
$\tau = \{o_0\}$
while episode not terminated **do**
  for $\pi \in \Pi$ **do**
    for i in simulation budget M **do**
      $\boldsymbol{\mu}_i \sim p(\boldsymbol{\mu}|\tau)$ ▷ sample dynamics given observation history
      $C^{\pi}_{\boldsymbol{\mu}_i} = \texttt{Simulate}(\pi, \boldsymbol{\mu}_i)$ ▷ simulate env using $\boldsymbol{\mu}_i$, and env model
    end for
  end for
  $\pi_{chosen} = \arg\min_{\pi \in \Pi} \frac{1}{M} \sum_{i=1}^{M} C^{\pi}_{\boldsymbol{\mu}_i}$ ▷ choose estimated safest policy
  $a_t = \pi_{chosen}(o_t)$
  $o_{t+1} \sim \texttt{Environment}(a_t)$ ▷ policy acts, env returns new observation
  $\tau = \tau \cup \{o_{t+1}\}$
  Update $p(\boldsymbol{\mu}|\tau)$ ▷ Update conditional probability on $\boldsymbol{\mu}$
end while

A.4 Sampling Environment Dynamics

The quality of our estimation of collision probability depends on the quality of estimation of environment parameters. The estimation of the possible parameters is updated using the history of observations from the agent: $p(\boldsymbol{\mu}|\tau)$. We start with a uniform density on $\boldsymbol{\mu}$ at the start of each episode, and with every new observation o_{t+1}, we compare the actions taken by target vehicles, with the actions according to either possible target behaviour model ($b_i \in \{0, 1\}$). If the observed target speed v^{obs} is within a certain threshold ε_v of the simulated behaviour speeds $v^{sim}_{b_i}$, then that target behaviour is retained in $p(\boldsymbol{\mu}|\tau)$. Using a stricter notation we can write $\boldsymbol{\mu}|_i = b_i$, where:

$$p(b_i) = \mathcal{U}(B), \quad B = \{b \mid \left|v^{sim}_b - v^{obs}\right| < \varepsilon_v, b \in \{0, 1\}\},$$

where v^{sim}_0, v^{sim}_1 are the simulated speeds for a cooperative and aggressive target, respectfully. All b_i's are independent, meaning there is no correlation between target behaviours.

To obtain our success rate and average score results, we sample without replacement $M = 200$ values for $\boldsymbol{\mu}$, and average out both the success rate (i.e. how many times the ego successfully navigated the intersection without crashing), and the score (i.e. regular reward function in the environment).