MaP, CaP, RaLly! Hybrid Architecture for Planning and Control of UGV in Stochastic Environments

Nathaniel Snyder
Dept. of MAE
UCLA
natsnyder1@gmail.com
UID: 205029016

Brian Wang
NESL, Dept. of CS
UCLA
wangbri1@g.ucla.edu
UID: 605229631

Swapnil Sayan Saha
NESL, Dept. of ECE
UCLA
swapnilsayan@g.ucla.edu
UID: 605353215

Abstract

With the advent and commercialization of unmanned ground vehicles (UGV), handling stochasticity safely but optimally in the operating environment in real-time entails computational and implementation complexities. In this paper, we benchmark the performance of a low-latency hybrid UGV controller that combines long-term goal convergence guarantees from control algorithms with the short-term robustness of reinforcement learning (RL) to safely guide an UGV to avoid dynamic obstacles while following near-optimal trajectories. Accelerated through imitation learning via stochastic online oracles, our evaluations show that the hybrid controller, using model-predictive-control (MPC) and deep Q-network (DQN), has superlative performance metrics over formal methods and RL frameworks working alone, with the potential of paving the way to Level-5 UGV controllers.

1 Introduction

Level 5 (SAE) autonomous vehicles have been touted as the upcoming digital disruption and a physical technological driver of the fourth industrial revolution [1][2]. While societal acceptance and confidence in self-driving technologies are commonplace [3], consumers manifest safety and reliability concerns [1][3]. Implanting real-time situational awareness in stochastic environments embracing probable edge cases entails computational and implementation complexities, exacerbated by the absence of truly dynamic training frameworks and datasets [4] and thus hurdling the advent of level 5 UGV. In particular, real-time trajectory management in UGV in the presence of a large number of stochastic and dynamic obstacles is NP-Hard and in PSPACE [5]. The computational elements within any UGV framework fulfill three main purposes: mapping, trajectory planning, and control / state estimation. Traditional approaches of mapping, planning and control of UGV in stochastic environments result in human engineered object-detector functions dealing with high-dimensional observation-space without probabilistic guarantees of fulfilling the end goal [5]. On one side, reactive navigation strategies, which infer the local costmap of obstacles from sensor information in real-time to continuously update waypoints, suffer from sub-optimal and understated trajectory projections. On the other hand, deliberate navigation strategies (pre-planned) are more likely to converge to global optima but are unable to handle dynamic deviations in a priori environmental belief states [6].

An intuitive solution to the problem is to design goal-driven intelligent agents capable of learning directly from experiential and exploratory observations while performing time-critical sequential decision making based on current and past state-action pairs. Several RL techniques (populated in [7]) exploiting Q-Learning [7][8] and actor-critic methods [5][6][9][10][11] have emerged and have been shown to exhibit superior and promising performance characteristics over formal methods [5][8][9][10] with smaller dependence on belief states in stochastic environments. However, these end-to-end agents are often unable to guarantee convergence to optimal trajectories in the long run

and are difficult to train, suffering from reward sparsity, tedious training phase, unpredictable / unsafe agent behavior, hyperparameter sensitivity, low generalizability and divergence in large and complex maps [8][10][12], constricting their superlative performance characteristics over the short and task-specific domain.

Keeping the challenges in mind and motivated from [8] and [10], we hypothesize **MaP**, **CaP**, **RaL**ly, an open-source end-to-end hybrid Ackermann UGV controller using a **M**odel **P**redictive **C**ontrol (MPC) oracle, which exploits a LiDAR Simulataneous Localization and Mapping (SLAM) to plan, optimize and fulfill user-defined navigation goals, while using a **R**einforcement **L**earning agent to handle dynamic obstacles in the environment. MaP, CaP RaLly combines the convergence guarantees of control theory with the ability to handle randomness from reinforcement learning for dynamically avoiding moving obstacles in an indoor warehouse environment, enabling the UGV to effectively and robustly navigate stochastic environments using RL under the guidance of a near-optimal MPC oracle. To facilitate the training process, we exploit imitation learning [8] with the notion of stochastic online oracles for near-optimal training data collection, resulting in fast algorithmic convergence. Evaluation in real-time environments characterize the hybrid approach as a low-latency controller able to handle stochastic obstacles and complete user-defined goals, with the potential of deployment in Level-5 indoor scenarios.

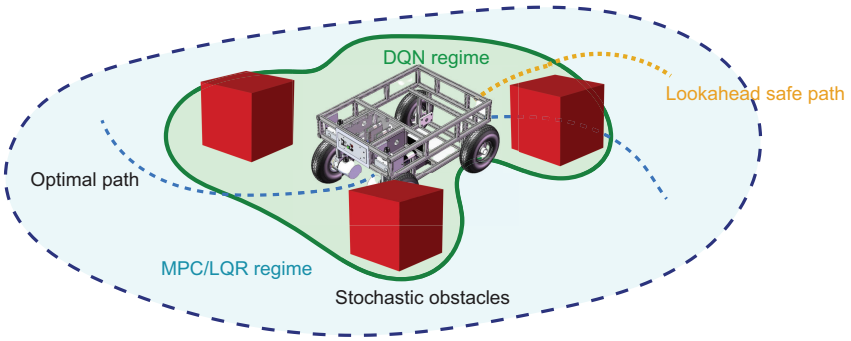

Figure 1: Summary of hybrid Ackermann UGV control using hybrid controller (MPC + RL). The RL node (using DQN) performs short term safe trajectory adjustments based on obstacle presence while the MPC keeps the robot on track for goal fulfillment.

2 Related Work

Propositions to autonomous trajectory management in the presence of stochastic obstacles consist of formal methods exploiting control theory, learning enabled systems (RL and Deep-Learning (DL)) and hybrid framework utilizing the aforementioned approaches. Our approach falls in the third category, borrowing long term stability and goal convergence tendencies from formal methods while using the short term safety nets generated by RL methods.

Formal Methods

Classical reactive and deliberate strategies for handling dynamic obstacles in real-time include traveling salesman / shortest path variants, velocity-obstacle, probabilistic / tree-search methods, non-linear optimization, integer programming, artificial potential fields, receding horizon optimization, dynamic window and stochastic reachability (SR) [5][6][12][13]. However, the application space of these methods are severely limited by the problem complexity and extent of stochasticity in the environment. Deliberate methods are more likely to converge to goal-points optimally as they have a broader receptive field than reactive methods, but fall short when the environment state alters. Although some of these methods (e.g. SR) provide probabilistic guarantees of whether the UGV will remain within a desired state-space, the probabilistic models assume obstacles with a predictable motion profile and are often deemed computationally expensive for time-critical decision making in the wild [5][6]. In fact, our evaluations show that formal methods provide superior performance

metrics in the absence of stochasticity in the environment, but fail to operate when the environment is dynamic, parallel to [5][6].

For our hybrid framework, we choose MPC as one of the nodes as it can act as a motion planner (deliberate) and real-time (reactive) controller within the stability, robustness and operating constraints [12]. We use MPC as the planner for optimal trajectory, while also keeping the UGV in course when the RL framework deviates it too far off course with assistance from tertiary reflex logic.

RL Frameworks

The state-of-the-art in reinforcement-learning controlled UGV is the use of DQN / Q-Learning [7][8], with the emergence of actor-critic methods [5][6][11] recently. However, as discussed in Section 1, since most of the proposed algorithms use non-linear function approximators, there is no guarantee whether the policy converges to the global optima [5]. Furthermore, these algorithms have regions in the state-space where they perform sub-optimally compared to formal methods [5] that are difficult to interpret. Our evaluations echo the findings in [5][9][14], showing that pure RL frameworks, although superlative in the short-term, are unable to operate in the long run to fulfill user-defined goals and diverge from the average MPC-guided path on its own, parallel to the findings in [5][10].

For this project, we chose Q-Learning/DQN trained via imitation learning as the RL node. The RL node helps avoid stochastic obstacles with the goal of minimizing the L2 norm between optimal trajectory and safe trajectory. For the target network, candidate architectures include convolutional neural networks (CNN) and recurrent architectures (e.g. CNN-LSTM).

Hybrid Architectures

A small number of articles attempt to illustrate the benefits of coupling the ability of RL to handle uncertainties with the convergence guarantees of formal methods (e.g. model predictive control (MPC) or probabilistic roadmaps (PRM)) in the context of intelligent transportation and robot locomotion, notably [9][10][12][15][16][17]. [9][12] and [16] assume obstacles are either static or have a predictable motion profile, [15] is task-specific (can only be applied in cruise-control scenarios) and [17] guarantees safety during exploration but no exploitation convergence guarantees. [10] fits the problem statement profile, coupling actor-critic methods from [11] with PRM as obstacle avoiding node and planning node respectively. However, the autonomous training framework requires considerable epochs to pass in order to obtain optimal training data. Compared to [10], the imitation learning approach [8] that we apply can ease the training process, with online oracles supplying useful examples covering edge cases without sacrificing performance metrics.

Table 1: Comparison of our approach with notable state-of-the-art

Methods	**Category***	**Algorithm**	**Observations**	**Actions****	**Expert**
[5]	F, R	SR, A3C	LiDAR (real-time)	Throttle, Steering (Disc.)	None
[6]	R	PPO	Rangefinder	Throttle, Steering (Cont.)	None
[8]	R	DQN	Camera, GPS, IMU and wheel speed	Throttle, Steering (Cont. and Disc.)	MPC
[9][10]	H	PRM and A2C [11]	LiDAR (SLAM, real-time), waypoints (goal, current and lookahead)	Throttle, Steering (Cont.)	None
[12]	H	MPC and TD0	Camera, motion-capture system	Yaw, Pitch, Roll (Cont.)	None
[13]	F	MPC	Camera	Throttle, Steering (Cont.)	None
Our method	**H**	**MPC and DQN**	**LiDAR (SLAM, real-time), waypoints (goal, current and lookahead)**	**Throttle, Steering (Cont. and Disc.)**	**MPC, human**

* F - Formal, R - RL, H - Hybrid (formal + RL)
** Cont. - Continuous action space, Disc. - Discrete action space

3 Materials and Methods

We implemented a full-stack Ackermann-drive UGV architecture (not available by default for ROS-Gazebo) equipped with LiDAR and camera and capable of trajectory planning and control in the ROS-Gazebo framework on a Linux machine, serving as our experimental testbed.

3.1 Training and Evaluation Framework

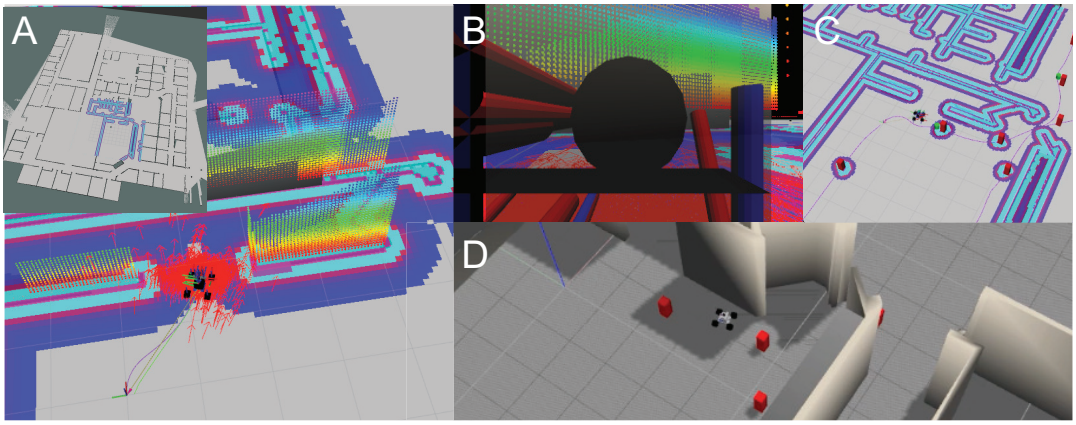

Figure 2: (A): Implemented ROS-Gazebo testbed showing a UGV traversing through a warehouse using LiDAR SLAM and MPC, inset shows part of warehouse as viewed through real-time LiDAR (B): Camera Feed (C): Training data collection with stochastic obstacles, with the world seen through real-time LiDAR and SLAM (D): Human view of training data collection.

Fig. 2 illustrates snapshots from the implemented evaluation framework. To achieve near-optimal path planning, a custom MPC trajectory optimizer node has been developed that tunes the output of classic path-planning algorithms (e.g. Dijisktra) to a near-optimal yet feasible path given the environmental and mechanical constraints. The path and desired controls are then fed into a custom error-tracking finite horizon LQR, which takes the vehicle to the goal point along the planned path in real-time. This framework is intended to bootstrap all phases for the RL-agent, including data collection, training and deployment phases. The implemented controller architecture, along with high-level view of the training grounds is shown in Fig. 3. To generate dynamic obstacles, the human marks average obstacle positions across P2P warehouse trajectories in RViz, while an obstacle manager stochastically moves those blocks using Gaussian process in Gazebo.

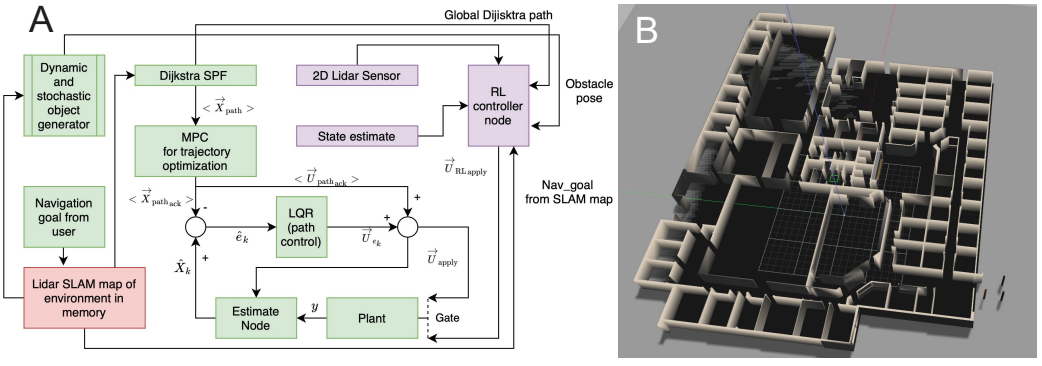

Figure 3: (A): Overview of the UGV control system (red: priori belief states from SLAM; green: deployment phase; purple: RL agent (during deployment). (B): Full environmental view.

For the obstacle node, we adopt imitation learning strategy from [8] to teach a RL agent how to follow projections from a stochastic online oracle [8][18] with agent in the loop. The agent is trained in the presence of obstacles, with the objective of minimizing the L2 norm between MPC projection (near-optimal) and RL projection (safe), allowing oracles to intervene in case of sub-optimal choices. The online training oracles include the LQR tracker for trajectory projection (near-optimal) and an user with a joystick for obstacle avoidance (safe). To initiate training for the RL agent, we recorded $\sim$ 70 minutes ($\sim$ 2 GB) of dynamic obstacle driving trajectories from human oracle and $\sim$ 30 minutes ($\sim$ 1 GB) of obstacle-less near-optimal trajectories from MPC-LQR controller along those same paths collected at 15 Hz. The goal of the agent is to successfully predict sufficiently safe control actions to navigate through the warehouse with obstacles with attempts to follow the near-optimal path mostly.

For switching between the RL and MPC nodes, we have designed a static tertiary reflex logic with fixed control regime based on LiDAR and MPC optimal path, with priority assignment based on distance from MPC path. In other words, the MPC node takes over control from RL node if the RL node deviates from the planned optimal path given a threshold radius.

The agent action space includes steering ($-\frac{\pi}{6}^c$ to $\frac{\pi}{6}^c$) and throttle (-0.25 m/s to 1.25 m/s), which has been discretized into $\sim$3 degree intervals and 7 bins respectively for the RL node. Note that the action space for the MPC node is still continuous. The observation space for the RL node includes the current location of the car $\{(x_t, y_t, \theta_t), x, y \in \mathbb{R}, \theta \in [-\pi, \pi]\}$, LiDAR scan $L_v \in \mathbb{R}^{1080}$, goal point (x_f, y_f, θ_f) and upcoming look-ahead points from the oracle projections $\{\mathbf{x}, \mathbf{y}, \theta, \mathbf{x}, \mathbf{y} \in \mathbb{R}^{15}\}$. The size of the state space is 95×1 and the size of the action space is 147×1 for the RL agent.

3.2 Algorithms and Architectures

Initialize replay memory D to capacity N;
Initialize action-value function Q with random weights θ ;
Initialize target action-value function $\hat{Q}$ with weights $\hat{\theta} = \theta$;
for *episode = 1, M* **do**
  for *steps = 1,T* **do**
    if *D.length* $>$ *threshold* **then**
      sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D ;
      if *path terminates at* $j + 1$ **then**
        $q_j = r_j$;
      else
        $q_j = r_j + \gamma \max_a \hat{Q}(\phi_{j+1}, a; \hat{\theta})$;
      end
      $y_j = \hat{Q}(\phi_j; \hat{\theta})$;
      $y_j[a_j] = q_j$;
      Perform gradient descent on $(y_j - Q(\phi_j; \hat{\theta}))^2$ w.r.t. θ ;
      Every C steps set $\hat{\theta} = \theta$;
    else
      store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D ;
    end
  end
end

Algorithm 1: Implemented DQN algorithm

Layer (type)	Param # *	Activation	Out. Shape *
Conv. 1	512	ReLu	(93, 128)
B. Norm 1	512		(93,128)
Dropout 1			(93,128)
Conv. 2	49280	ReLu	(91,128)
B. Norm 2	512		(91,128)
Dropout 2			(91,128)
Conv. 3	49280	ReLu	(89,128)
B. Norm 3	128		(137,32)
Dropout 3			(89,128)
Flatten **			(11392)
Dense	1674771	Softmax	(147)

* Values change depending on network architecture
** For CNN-LSTM, replace Flatten with LSTM
*** Hybrid architecture: Two CNNs, each dealing with pose and LiDAR data separately

Table 2: Sample target network architecture

For the RL node, we implemented a Deep Q-network with the goal of minimizing the L2 norm between measured and predicted state-action value function. The intuition behind the Q-network is the hypothesis that given a set of observations, the Q-network will choose that state-action pair that maximizes the Q function and hence choose the optimal action given a particular state from the training set. Since the reward function penalizes the agent for moving too far away from the optimal path or collide with walls or obstacles, we postulate that the resulting agent should be robust to uncertainties in the environment while achieving near-optimal trajectory projections with the help of the MPC oracle to the end goal. For the target network, we implemented three candidate architectures: CNN, CNN-LSTM and a hybrid architecture where two CNNs separately deal with pose and LiDAR data, motivated from [19]. Convolutional layers are able to extract differential patterns by enforcing a degree of local connectivity among adjacent samples and enabling scale invariance (account for noisy feature discrepancy among similar state-action pairs), making the architecture suitable as a high-dimensional feature extractor. Recurrent layers are able to learn temporal dynamics of feature activations, inferring temporal dependencies [19]. The batch normalization and dropout layers help minimize overfitting.

The reward increases linearly as the agent gets close to the optimal path with 60% dropout probability. We applied several strategies in order to obtain fast but accurate convergence metrics, including dense experience replay (replay memory size: 1000) and target network updates (update every 50 steps), gradient (within 0 to 1) and reward clipping (dropout: 60%) and the use of batches (minibatch size: 20). Other hyperparameters include the optimizer: Adam (adaptive learning rate), $\gamma = 0.95$, $\epsilon = 1.0$ (decay = 0.995) and discount factor = 0.99.

Due to shortcomings in ROS's path planning packages for optimal control of an Ackermann-drive vehicle, we implemented a custom MPC algorithm to plan safe vehicle paths for the robot. Our MPC takes the start pose of the robot, the goal pose of the robot, and the 'globally safe' (but non-dynamically constrained) Dijkstra path to follow. The MPC node implements a log-barrier interior point optimizer to solve sequential approximations of the non-linear program, which consists of an affine cost function coupled to non-linear dynamical constraints. Once converged, the robot is provided a set of feasible states and controls to reach the goal position while following as closely as it can to Dijkstra's path. To assert the robot follows the MPC trajectory in the wake of non-trivial process and measurement noises (and in real time), the states and controls were next fed into a finite horizon error tracking LQR by performing successive linearizations of the dynamics model about each MPC state and control pair. This allows the Ackermann-vehicle to follow a safe-yet-feasible path plan throughout the tightly cornered warehouse with a near 100% success rate for non-obstacle cases.

3.3 Implementation Specifics

The models and the code were implemented in Python, using Keras and Sklearn via a Tensorflow backend along with RosPy for interaction between the models and the experimental testbed. All models were trained on a GPU notebook with 12 GB RAM, 3.8 GHz Intel Core i7-7700 8-core CPU and 6 GB NVIDIA GeForce 1060.

4 Results and Discussion

Fig. 4 illustrates the performance of candidate target-network architectures, with average Q-Mean Absolute Error (Q-MAE) being the performance metric, while inset shows theoretical latency during training. We observe that CNN and CNN-LSTM performed closely in presence of obstacles. Since CNN has a lower theoretical latency than CNN-LSTM, we chose CNN as the final target-network architecture.

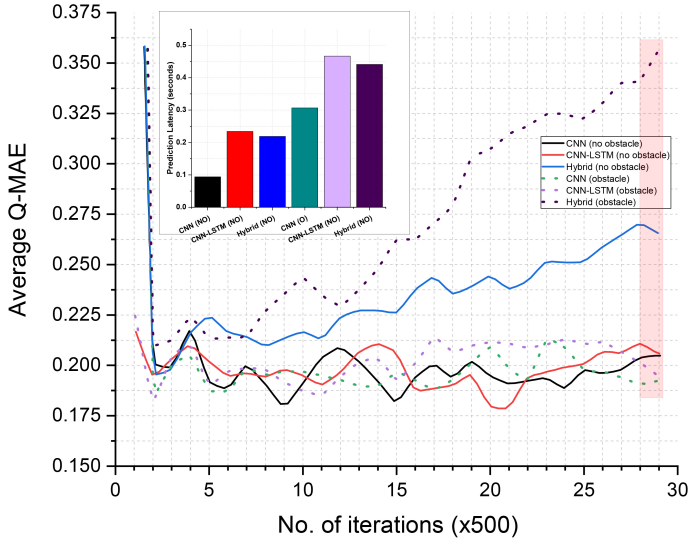

Figure 4: Average Q-MAE for candidate network architectures (inset: theoretical latency)

We also observed converge of Q-MAE within $\sim$ 30 episodes (500 iterations per episode) to around 0.2 for most network architectures. This is because DQN continuously updates the Q value for a particular action taken in a path, with the goal of minimizing the measured (optimal) vs predicted Q value. Since exploration is not necessary and the examples resemble near-optimal trajectories, we see fast convergence. In addition, we received an average target network validation accuracies of 84% for no-obstacle case and 64% for obstacle case.

We observed a performance degradation when we increased the number of layers (from 3 to 6) in the target network architecture. Although this is counter-intuitive, we postulate that deeper networks tend to memorize the Q-values calculated from the training-set and hence are unable to generalize on slightly different state-action pairs (which can occur as the MPC will influence the state-action pairs the RL node will observe if it gets too far from the optimal path) during deployment phase.

Table 3 illustrates the performance of our algorithm vs formal and purely RL-driven methods, set up for two environments: environment with obstacle and no obstacles, with each path containing a set of goalpoints ($\sim$ 11-12 per path) to fulfill and set by the user. The performance metrics are normalized over chunks within two P2P goalpoints except the success rate, which is defined as the ratio of fulfilled goalpoints to total number of goalpoints. The setup sets the formal and pure RL methods as baselines.

Table 3: Performance comparison of classical, RL and our (MPC+RL) algorithm.

Parameters (normalized)	**MPC**	**DQN**	**Hybrid (ours)**
Avg. trajectory length (m)	**4.2 (NO)**, F (O)	4.85 (NO), 6.1 (O)	4.43 (NO), **5.47 (O)**
Avg. trajectory duration (sec)	**3.92 (NO)**, F (O)	5.1 (NO), 7.1 (O)	4.35 (NO), **6.69 (O)**
Avg. Speed (m/s)	**1.07 (NO)**, F (O)	1.05 (NO), **0.85 (O)**	1.02 (NO), 0.81 (O)
Success rate (%)*	**100 (NO)**, 0 (O)	80 (NO), 20 (O)	90 (NO), **75 (O)**
Involvement ratio (RL:MPC)	N/A	N/A	82:18 (NO), 73:27 (O)

O - with obstacle, NO - without obstacle

* collision with wall or obstacle is a failure

From Table 3, we observe that solo MPC provides optimal trajectory for environment with no obstacles against solo RL and hybrid approach. However, the hybrid controller provides optimal results for environment with stochastic obstacles against solo RL and solo MPC. This is in line with the discussion in Section 2. With the environment a priori (non-obstacle case), the MPC can pre-plan the shortest and fastest P2P path (and adjust if the vehicle goes of course) mathematically possible using formal methods. However, since stochastic obstacles are not stored in SLAM map, the MPC fails to operate in the obstacle setting. Although the DQN is trained using the MPC oracle in case of no obstacles, there is a slight performance degradation for pure RL approach. In our setting, we observed that the RL node tends to diverge in the long run from the optimal given by MPC, in line with the findings in [5]. We postulate that the target network often fails to linearize the action that leads to the highest state-action value, giving rise to local regions of failure. We also observed the following cases:

- *Unstable target network in edge cases* - We observed that the DQN behavior becomes unpredictable when extreme cases are encountered, e.g. narrow corridors, small clearance around obstacles etc. This might be attributed to insufficient number of those samples appearing in the training set, as well the as the reward function and tertiary reflex logic having no external information about those situations. The MPC itself cannot save the UGV under such circumstances as the control regime is still within the threshold radius under most of these cases.
- *Occasional disagreement (conflicting actions) between RL node and MPC* - Conflicting actions occur when the RL node is completely out of phase with the MPC in equal authority, causing the robot to stop. This can be postulated as the inability of the reflex agent to properly break ties and prioritize assignments between RL and MPC node.

For the stochastic case, we see that the hybrid architecture has the highest success rate, with the RL agent in control of the UGV $\sim$ 70% of the time while the MPC chips in to prevent divergence. In our setting, the hybrid approach improves the success rate by 55% over RL methods. In addition, we

observed a prediction latency of < 2 mS @ 15 Hz system sampling time during deployment, which indicates that our hybrid controller is suitable for deployment without incurring noticeable system overhead.

Fig. 5 shows sample trajectory from one trial for all three methods. The trajectory reflects and summarizes the aforementioned discussions in this section. In particular, one can infer where the MPC node had to correct the RL node as it started to diverge, characterized by sharp turns.

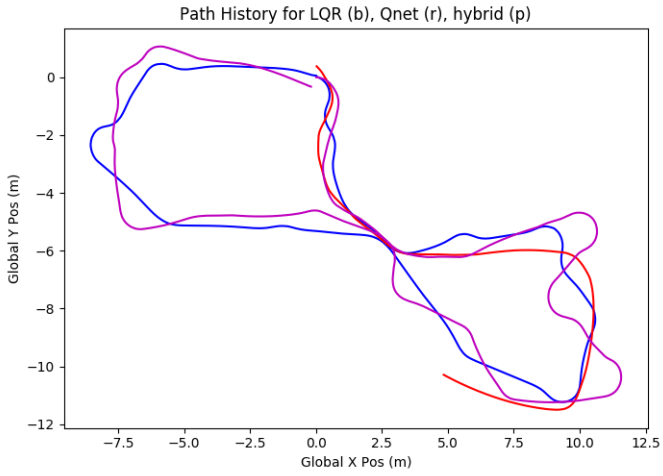

Figure 5: Sample trajectory of MPC (blue), RL (red) and hybrid (pink) algorithms for path with no obstacles. In this particular trial, the RL node failed to complete the entire path.

5 Conclusion and Future Work

In this paper, we introduced a low-latency real-time hybrid UGV controller exploiting Q-Learning and MPC, with the ability to handle dynamic obstacles in complex environments while following near-optimal path to goal points via a streamlined, open-source and fast training framework using imitation learning. Our evaluations indicate that coupling formal methods with learning-enabled systems offer significant performance improvement over the former operating alone, suitable for deployment in goal-oriented physical Level-5 UGV robots.

There are several future directions to our work in line with solving observed issues. These include designing an adaptive tertiary reflex agent that adaptively assigns control regime based on variable threshold radius to either the MPC or RL node based on real-time LiDAR scan, training the RL node with more training data covering extreme situations, using weighted feature maps to scale and prioritize certain input modalities (e.g. lookahead MPC path) and designing a better reward function. In addition, one may benchmark more stable RL algorithm choices such as A2C, A3C, PPO and DDPG.

Author Contributions

- Nathaniel Snyder (team leader) designed and coded the experimental testbed, the UGV control system, the tertiary reflex logic and the MPC node.
- Brian Wang implemented the DQN and the target network architecture, set up the training framework and coupled the Tensorflow models with ROS-Gazebo framework.
- Swapnil Sayan Saha designed the obstacle manager, the tertiary reflex logic, the target network architecture, the evaluation experiments and general project directions.

References

[1]. Garidis, Konstantin, et al. *Toward a User Acceptance Model of Autonomous Driving.* Proceedings of the 53rd Hawaii International Conference on System Sciences. 2020.

[2]. Li, Guoping, Yun Hou, and Aizhi Wu. *Fourth Industrial Revolution: Technological Drivers, Impacts and Coping Methods.* Chinese Geographical Science 27.4 (2017): 626-637.

[3]. Wintersberger, Sophie, Muhammad Azmat, and Sebastian Kummer. *Are We Ready to Ride Autonomous Vehicles? A Pilot Study on Austrian Consumers' Perspective*. Logistics 3.4 (2019): 20.

[4]. Hussain, Rasheed, and Sherali Zeadally. *Autonomous cars: Research results, issues, and future challenges.* IEEE Communications Surveys & Tutorials 21.2 (2018): 1275-1313.

[5]. A. Garg, H. L. Chiang, S. Sugaya, A. Faust and L. Tapia, *Comparison of Deep Reinforcement Learning Policies to Formal Methods for Moving Obstacle Avoidance,* 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau, China, 2019, pp. 3534-3541.

[6]. E. Meyer, H. Robinson, A. Rasheed and O. San, *Taming an Autonomous Surface Vehicle for Path Following and Collision Avoidance Using Deep Reinforcement Learning,* in IEEE Access, vol. 8, pp. 41466-41481, 2020.

[7]. M. S. Shim and P. Li, *Biologically inspired reinforcement learning for mobile robot collision avoidance,* 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, 2017, pp. 3098-3105.

[8]. Y. Pan et al. *Agile Autonomous Driving using End-to-End Deep Imitation Learning.* Robotics: Science and Systems (RSS). Pittsburgh, PA, USA, 2018.

[9]. A. Faust et al., *PRM-RL: Long-range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-Based Planning,* 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, 2018, pp. 5113-5120.

[10]. A. Francis et al., *Long-Range Indoor Navigation With PRM-RL,* in IEEE Transactions on Robotics (2020).

[11]. H. L. Chiang, A. Faust, M. Fiser and A. Francis, *Learning Navigation Behaviors End-to-End With AutoRL*, in IEEE Robotics and Automation Letters, vol. 4, no. 2, pp. 2007-2014, April 2019

[12]. C. Greatwood and A. G. Richards. *Reinforcement learning and model predictive control for robust embedded quadrotor guidance and control.* Autonomous Robots 43.7 (2019): 1681-1693.

[13]. S. R. Devaragudi and B. Chen. *MPC-Based Control of Autonomous Vehicles With Localized Path Planning for Obstacle Avoidance Under Uncertainties."*International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Vol. 59292. American Society of Mechanical Engineers, 2019.

[14]. B. Lutjens, M. Everett and J. P. How, *Safe Reinforcement Learning With Model Uncertainty Estimates,* 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 2019, pp. 8662-8668.

[15]. N. K. Ure, M. U. Yavas, A. Alizadeh and C. Kurtulus, *Enhancing Situational Awareness and Performance of Adaptive Cruise Control through Model Predictive Control and Deep Reinforcement Learning,* 2019 IEEE Intelligent Vehicles Symposium (IV), Paris, France, 2019, pp. 626-631.

[16]. T. Tram, I. Batkovic, M. Ali and J. Sjöberg, *Learning When to Drive in Intersections by Combining Reinforcement Learning and Model Predictive Control,* 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 2019, pp. 3263-3268.

[17]. T. Koller, F. Berkenkamp, M. Turchetta and A. Krause, *Learning-Based Model Predictive Control for Safe Exploration*, 2018 IEEE Conference on Decision and Control (CDC), Miami Beach, FL, 2018, pp. 6059-6066.

[18]. C-A Cheng et al. *Accelerating Imitation Learning with Predictive Models.* Proceedings of Machine Learning Research 89: 3187-3196 (2019).

[19]. S. S. Saha, S. S. Sandha and M. Srivastava, *Deep Convolutional Bidirectional LSTM for Complex Activity Recognition with Missing Data*, (to appear) in Activity and Behavior Computing - Smart Innovations, Systems and Technologies, Springer (2020).