MAP CAP RALLY

Hybrid Architecture for Planning and Control of UGV in Stochastic Environments

NATHANIEL SNYDER*, BRIAN WANG* AND SWAPNIL SAYAN SAHA*

NESL Networked & Embedded Systems Laboratory

THE UNIVERSITY OF CALIFORNIA · UCLA · ET THERE BE LIGHT

Overview

○ An end-to-end hybrid unmanned ground vehicle (**UGV**) controller uses **M**odel **P**redictive **C**ontrol (**MPC**) for ***optimal*** trajectory projection and **R**einforcment **L**earning (**RL**) node for ***safely*** handling stochastic obstacles in real time.

○ Exploits ***imitation learning*** via stochastic ***online oracles*** with agent in the loop to achieve near-optimal trajectories for fast algorithmic convergence using ***Deep Q Networks***.

○ Full-stack open-source framework for Ackermann drive UGV in ROS-Gazebo using finite horizon error-tracking LQR ***mimics real-world*** environmental and mechanical constraints.

Experimental Framework

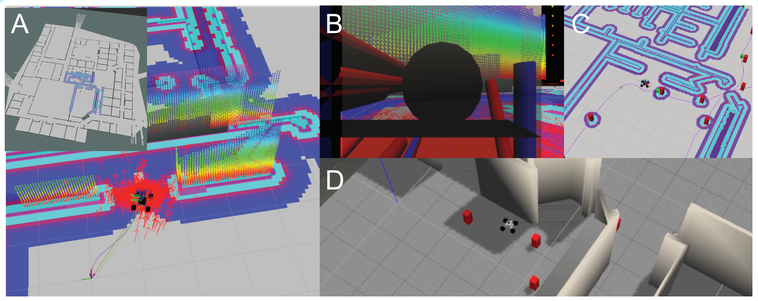


Fig. 1. (A): Implemented ROS-Gazebo testbed showing a UGV traversing through a warehouse using LiDAR SLAM and MPC, inset shows part of warehouse as viewed through real-time LiDAR **(B):** Camera Feed **(C):** Training data collection with stochastic obstacles, with the world seen through real-time LiDAR and SLAM **(D):** Human view of training data collection.

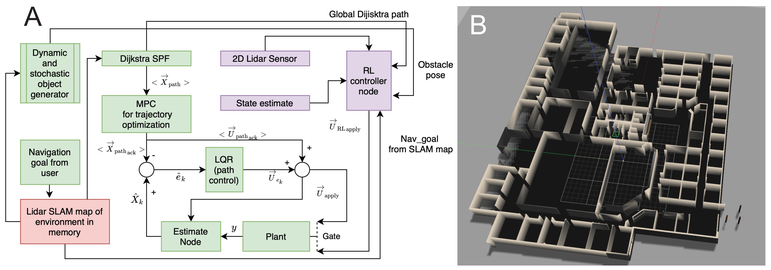


Fig. 2. (A): Overview of the UGV control system (red: priori belief states from SLAM; green: deployment phase; purple: RL agent (during deployment). **(B):** Full environmental view.

○ **Custom *MPC trajectory optimizer*** node tunes the output of classic path-planning algorithms (e.g. Dijiskstra) to a ***near-optimal yet feasible path***.

○ Dynamic ***obstacles handled safely using RL node*** exerting controlled disturbance signals, attempting to safely reduce L2 norm ***between safe and optimal projections (trade-off)*** without significant latency.

○ ***Error-tracking finite horizon LQR*** takes the vehicle to the goal point along the planned path in real-time.

○ The ***online training oracles*** for bootstrapping imitation learning (training data collection) include the ***LQR tracker*** for trajectory projection (near-optimal) and an ***user with a joystick*** for obstacle avoidance (safe).

○ *Observation space:* current location of 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) , upcoming lookahead points from oracle projections $\{\mathbf{x}, \mathbf{y}, \theta, \mathbf{x}, \mathbf{y} \in \mathbb{R}^{15}\}$; *action space:* steering $(\frac{-\pi^c}{6}, \frac{\pi^c}{6})$ and throttle (-0.25 m/s to 1.25 m/s) discretized into buckets.

○ Obstacles placed by oracle and stochastcally moved using ***Gaussian process.***

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

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	512		(89,128)
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

Fig. 3. (A): Implemented DQN algorithm **(B):** Sample target network architecture (CNN), candidates include CNN, CNN-LSTM and Hybrid***

○ RL algorithm: ***Deep Q-network***; goal: minimize the L2 norm between measured and predicted state-action value function.

○ Target network candidates contain ***convolutional and LSTM layers*** to extract differential patterns and ***features*** in high dimensions of space and time. Batch normalization and dropout helps prevent overfitting.

○ ***Convergence strategies*** applied: 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%), batch RL (minibatch size: 20).

○ State space: 95X1, Action space output: 147; throttle = [-0.25, 0.0, 0.25, 0.50, 0.75, 1.0, 1.25], steering is binned at ~3 degree intervals.

○ Other parameters: Optimizer: Adam (adaptive learning rate), γ = 0.95, ε = 1.0 (decay = 0.995), discount factor = 0.99.

○ Training data: ~ 70 minutes of obstacle and ~30 minutes of non-obstacle driving data, sampled at ~ 15 Hz.

Conclusion and Future Work

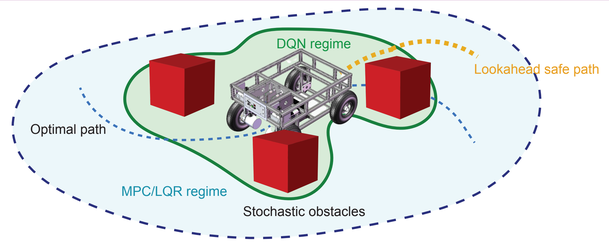


Fig. 5. Summary of hybrid Ackermann UGV control using RL and MPC

○ ***Outcome***: ***Low-latency*** real-time ***hybrid UGV controller*** exploiting ***Q-Learning and MPC*** able to ***handle dynamic obstacles*** in ***complex environments*** while following near-optimal path to goal points with streamlined, open-source and ***fast*** training framework using ***imitation learning***.

○ ***Application:*** Indoor service robots such as vacuum bots (e.g. Roomba), guard robots, waiter bots, assistive bots etc.

○ ***Future Work:***

○ Unstable target network in edge cases - requires more training examples, alternate network architectures and sophisticated reward function design.

○ RL-node tends to diverge from optimal path in absence of obstacles - 2D convolution (windowed approach) coupled with LSTM in target network for better future lookahead states; weighted feature maps for ranking observation modalities.

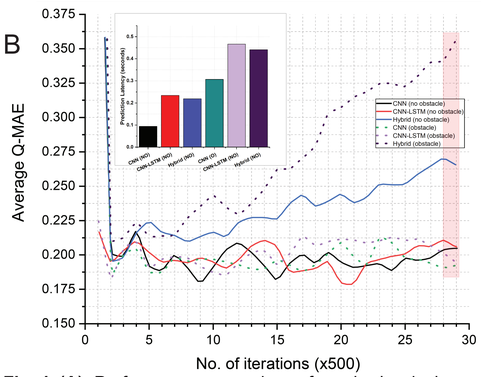
○ Occasional disagreement (conflicting actions) between RL node and MPC - Tertiary reflex agent for breaking ties based on priority assignments or adaptive controller regime based on LiDAR scan.

○ Deployment on physical robotic systems.

Results

A

Parameters (normalized over sections)	MPC	DQN	Hybrid
Avg. trajectory length (m)	**4.2 (NO)**, F (O)	4.85 (O), 6.1 (NO)	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 (NO)	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)



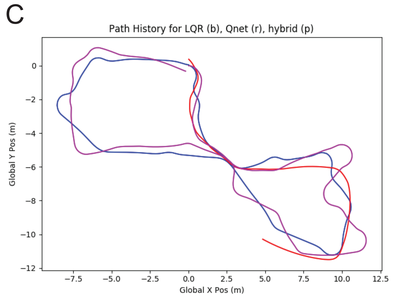


Fig. 4. (A): Performance comparison of purely classical, purely RL and our (MPC+RL) algorithm. **(B):** Average Q-MAE for candidate network architectures (inset: theoretical latency) **(C):** Sample trajectory of MPC, RL and hybrid algorithms for path with no obstacles.

○ Optimal target network architecture (metric: average Q-mean absolute error (MAE), lower is better): ***CNN***.

○ Setting: MPC, DQN and Hybrid (MPC+DQN) setup for two environments: environment with obstacle and no obstacles; path contains set of goalpoints (~11-12 per path) to follow

○ ***Fast convergence*** due to near-optimal training data (no exploration required) from online oracles.

○ Theoretical latency during training: 0.1-0.45 sec; ***prediction latency*** during deployment: ***< 2 mS*** @ 15 Hz system sampling time.

○ Solo ***MPC provides optimal trajectory for environment with no obstacles*** against solo RL and hybrid approach.

○ ***The hybrid controller provides optimal results for environment with stochastic obstacles*** against solo RL and solo MPC.

○ Target network validation accuracies: ~ 84% in no-obstacle, ~ 64% in stochastic obstacle.

○ Deployment of hybrid architecture: static tertiary reflex logic with fixed control regime based on LiDAR and MPC optimal path (priority assignment based on distance from MPC path).