Experimental Validation on Aerial Vehicles of Real-Time Motion Planning with Continuous-Time Q-Learning[⋆]

Christian Llanes[*], **Joshua Netter**[*], **Kyriakos G. Vamvoudakis**[*], **Samuel Coogan**[*]

[*] *Georgia Institute of Technology, Atlanta, GA 30332 (e-mails: christian.llanes@gatech.edu, jnetter6@gatech.edu, kyriakos@gatech.edu, sam.coogan@gatech.edu)*

Abstract: In this paper, we propose an algorithm and implementation for real-time optimized kinodynamic motion planning for aerial vehicles with unknown dynamics in crowded environments. A random-sampling space-filling tree is used for both planning and rapidly replanning a path through the environment. Then, continuous-time Q-learning is used to approximately solve the resulting finite-horizon optimal control problem online to optimally track the planned path. To facilitate the Q-learning, we propose an actor-critic structure with integral reinforcement learning to approximate the Hamilton-Jacobi-Bellman equation. The critic approximates the Q-function while the actor approximates the control policy. We demonstrate our approach on custom drone hardware in which all planning, learning, and control computations are conducted onboard in real-time.

Keywords: Mission planning and decision making, trajectory and path planning, optimal control, reinforcement learning, UAVs

1. INTRODUCTION

Over the past decade, unmanned aerial vehicles (UAVs) have become increasingly accessible to a wide variety of consumers and are available for a wide variety of purposes such as search and rescue, package delivery, aerial surveying and photography, and other tasks that cannot easily be accomplished on the ground. This has influenced a greater demand for autonomous capabilities, and perhaps the most critical component for these autonomous systems is planning and navigation for objective completion. This poses a significant challenge, as UAVs often operate in dynamic environments that make pre-flight path planning an impossibility. Instead, these vehicles must be able to plan and adjust their path while in flight to properly respond to a changing environment. Autonomous flight is further complicated by the problem of finding the optimal control to traverse these planned paths. UAVs are complicated systems, and any model constructed of the system dynamics will inevitably be simplified. Additionally, even if the system dynamics are known, solving the Hamilton-Jacobi-Bellman equation for the optimal control is a computationally intensive process as shown in Lewis et al. (2012), and UAVs may not be equipped to find a solution. Thus, it is crucial for UAVs to be capable of not only adjusting their planned path in real-time, but also be capable of quickly estimating their system dynamics and optimal control as well. To this end, we propose an algorithm for kinodynamic motion planning in real-time through a dynamic environment, as well as a method for estimating a solution for the finite-horizon optimal control problem in a system with unknown dynamics.

1.1 Related Work

Motion planning through dynamic environments is a challenging problem, requiring not only the ability to plan a path through high-dimensional spaces but also replan these paths as the obstacle space changes. To address these challenges, Otte and Frazzoli (2014, 2016) propose RRT^X, an asymptotically optimal random-sampling motion planning algorithm for dynamic environments. However, this algorithm may encounter difficulties when considering motion constraints on real systems such as UAVs. These constrained motion planning concerns are presented as the kinodynamic motion planning problem in Donald et al. (1993). In Kontoudis et al. (2020, 2022), the RRT^X algorithm is adjusted for kinodynamic motion planning by augmenting the obstacles in real-time in order to ensure that the agent following the path does not collide with the true obstacle space.

A UAV navigating a real environment requires not only a method to plan a safe path, but also a control framework to move it along this planned path. Unfortunately, there is no guarantee that the system dynamics of this UAV are known beforehand. To prepare for unknown dynamics, the motion planning approach must use adaptive control techniques from Ioannou and Sun (2012). In Vamvoudakis and Lewis (2009), the authors use reinforcement learning from Sutton and Barto (1998) to solve the continuous time infinite-horizon optimal control problem. This approach is later adapted to find the optimal control for

[⋆] This work is supported in part by NSF under grant Nos. NRI-1924978, CAREER CPS-1851588, CPS-2038589, CPS-2227185, and by NASA ULI under grant number 80NSSC20M0161.

the kinodynamic motion planning problem in Kontoudis and Vamvoudakis (2019). This work is further expanded in Netter et al. (2021), where the authors use a similar approach to examine the motions of other agents to predict their motion planning strategies.

1.2 Contributions

In this work we build upon the model-free Q-learning formulation from Kontoudis and Vamvoudakis (2019) along with RRT$^{\mathrm{X}}$ for motion planning to develop a novel software and hardware stack applying these methods to a quadrotor UAV. We therefore demonstrate for the first time the real-time capabilities of the proposed motion planning and model-free optimal control pipeline.

This paper is organized as follows. We introduce the model-free Q-learning control problem formulation in Section 2. In Section 3 we describe how the Q-learning formulation is applied to a UAV with an inner-loop rotational dynamics regulation controller. In Section 4.2, we first present the algorithm used in our implementation for the real hardware flight experiment. Then, we demonstrate our results with a real hardware flight experiment on a quadrotor UAV with a RRT$^{\mathrm{X}}$ path planning algorithm and Q-learning model-free controller.

1.3 Notation

We denote the symmetric Kronecker product for vectors U and V by $U \otimes_s V$. We also denote the symmetric half vectorization of a symmetric matrix $A \in \mathbb{R}^{n\times n}$ by $\mathrm{svec}(A) \in \mathbb{R}^{\frac{1}{2}n(n+1)}$ as in Schäcke (2013).

2. PROBLEM FORMULATION

Consider a linear time-invariant (LTI) and continuous-time system

$$\dot{x} = Ax(t) + Bu(t),\ x(0) = x_0,\ t \geq 0 \tag{1}$$

with state $x(t) \in \mathcal{X} \subseteq \mathbb{R}^n$, control $u(t) \in \mathcal{U} \subseteq \mathbb{R}^m$, plant matrix $A \in \mathbb{R}^{n\times n}$, and input matrix $B \in \mathbb{R}^{n\times m}$. We seek to drive the state to a desired reference x_r, therefore we define a new state variable $\bar{x}(t) = x(t) - x_r$. The control strategy is selected to minimize a finite horizon quadratic cost function that results in the value function

$$V^\star(\bar{x}; t_0, T) = \min_u \left(\phi(T) + \frac{1}{2}\int_{t_0}^{T} (\bar{x}^\top M\bar{x} + u^\top Ru)\mathrm{d}\tau \right) \tag{2}$$

for finite horizon T, terminal penalty function $\phi(T) = \frac{1}{2}\bar{x}^\top(T)P(T)\bar{x}(T)$, terminal Riccati matrix $P(T) \in \mathbb{R}^{n\times n} \succ 0$, state penalty matrix $M \in \mathbb{R}^{n\times n} \succeq 0$, and control penalty matrix $R \in \mathbb{R}^{m\times m} \succ 0$. The Hamiltonian with respect to (1) and (2) is defined as

$$\begin{aligned}\mathcal{H}\left(\bar{x}; u; \frac{\partial V^\star}{\partial \bar{x}}, \frac{\partial V^\star}{\partial t}\right) = \frac{\partial V^\star}{\partial t} + \frac{\partial V^\star}{\partial \bar{x}}^\top (A\bar{x} + Bu) \\ + \frac{1}{2}(\bar{x}^\top M\bar{x} + u^\top Ru).\end{aligned} \tag{3}$$

Since (1) is linear and the associated cost function is quadratic, then the solution to (3) is a quadratic cost-to-go defined as

$$V^\star(\bar{x}; t) = \frac{1}{2}\bar{x}^\top P(t)\bar{x} \tag{4}$$

for $P(t) \in \mathbb{R}^{n\times n} \succ 0$ as the solution to the differential Riccati equation

$$-\dot{P}(t) = P(t)A + A^\top P(t) + M - P(t)BR^{-1}B^\top P(t). \tag{5}$$

The optimal control is therefore defined as

$$u^\star(\bar{x}; t) = -R^{-1}B^\top P(t)\bar{x},\ \forall \bar{x}, t \geq 0. \tag{6}$$

We next propose an action-dependent quality function defined as

$$\begin{aligned}\mathcal{Q}(\bar{x}; u; t) &= V^\star(\bar{x}; t) + \mathcal{H}\left(\bar{x}; u; \frac{\partial V^\star}{\partial \bar{x}}, \frac{\partial V^\star}{\partial t}\right) \\ &= V^\star(\bar{x}; t) + \frac{1}{2}(\bar{x}^\top M\bar{x} + u^\top Ru) \\ &\quad + \bar{x}^\top P(t)(A\bar{x} + Bu) + \frac{1}{2}\bar{x}^\top \dot{P}(t)\bar{x}\end{aligned} \tag{7}$$

where $\mathcal{Q}(\bar{x}; u; t) \in \mathbb{R}$. We rewrite (7) in a compact matrix form by first constructing an augmented state $U := [\bar{x}^\top u^\top]^\top \in \mathbb{R}^{n+m}$ and therefore have

$$\mathcal{Q}(\bar{x}; u; t) = \frac{1}{2}U^\top \bar{\mathcal{Q}}(t)U := \frac{1}{2}U^\top \begin{bmatrix} Q_{\mathrm{xx}}(t) & Q_{\mathrm{xu}}(t) \\ Q_{\mathrm{ux}}(t) & Q_{\mathrm{uu}} \end{bmatrix} U \tag{8}$$

where $Q_{\mathrm{xx}}(t) = \dot{P}(t) + P(t) + M + P(t)A + A^\top P(t)$, $Q_{\mathrm{xu}}(t) = Q_{\mathrm{ux}}^\top = P(t)B$, and $Q_{\mathrm{uu}} = R$. By using the stationary condition $\partial\mathcal{Q}(\bar{x}; u; t/\partial u) = 0$ we find a model-free expression for the optimal control $u^\star$ in (6) as

$$u^\star(\bar{x}; t) = \arg\min_u \mathcal{Q}(\bar{x}; u; t) = -Q_{\mathrm{uu}}^{-1}Q_{\mathrm{ux}}(t)\bar{x} \tag{9}$$

We implement the Q-learning framework as an actor and critic network structure. The critic approximates the Q-function and the actor performs a policy improvement. The critic approximator is defined as

$$\begin{aligned}\mathcal{Q}^\star(\bar{x}; u^\star; t) &= \frac{1}{2}U^\top \begin{bmatrix} Q_{\mathrm{xx}}(t) & Q_{\mathrm{xu}}(t) \\ Q_{\mathrm{ux}}(t) & Q_{\mathrm{uu}} \end{bmatrix} U \\ \mathcal{Q}^\star(\bar{x}; u^\star; t) &= \frac{1}{2}\mathrm{svec}(\bar{\mathcal{Q}}(t))^\top (U \otimes_s U).\end{aligned} \tag{10}$$

where $\mathrm{svec}(\bar{\mathcal{Q}}(t))$ is the symmetric vectorization of the symmetric matrix $\bar{\mathcal{Q}}$ to reduce computational complexity of the algorithm. Then, we let $\nu(t)^\top W_{\mathrm{c}} := \frac{1}{2}\mathrm{svec}(\bar{\mathcal{Q}})$ for critic weight vector $W_{\mathrm{c}} \in \mathbb{R}^{\frac{(n+m)(n+m+1)}{2}}$ and radial basis function $\nu(t) \in \mathbb{R}^{\frac{(n+m)(n+m+1)}{2} \times \frac{(n+m)(n+m+1)}{2}}$. With this we rewrite the Q-function as

$$\mathcal{Q}^\star(\bar{x}; u^\star; t) = W_{\mathrm{c}}^\top \nu(t)(U \otimes_s U). \tag{11}$$

We use a similar approach for the actor and assign $\mu(t)^\top W_{\mathrm{a}} := -Q_{\mathrm{uu}}^{-1}Q_{\mathrm{ux}}(t)$ to rewrite (9) as

$$u^\star(\bar{x}; t) = W_{\mathrm{a}}^\top \mu(t)\bar{x} \tag{12}$$

for actor weight matrix $W_{\mathrm{a}} \in \mathbb{R}^{n\times m}$ and radial basis function $\mu(t) \in \mathbb{R}^{n\times n}$. Since the system is unknown and our formulation is model-free, we do not know the ideal critic weights W_{c} a priori. Therefore, we use estimated critic weights, $\hat{W}_{\mathrm{c}}$, and apply an adaptive parameter estimation technique from Ioannou and Sun (2012) to approximate the Q-function as

$$\hat{\mathcal{Q}}(\bar{x}; u; t) = \hat{W}_{\mathrm{c}}^\top \nu(t)(U \otimes_s U). \tag{13}$$

Using the same parameter estimation technique as in (13) we rewrite the actor from current estimates of the actor weights $\hat{W}_{\mathrm{a}}$ as

$$\hat{u}(\bar{x}; t) = \hat{W}_{\mathrm{a}}^\top \mu(t)\bar{x}. \tag{14}$$

Using an integral reinforcement learning (IRL) approach from Vrabie et al. (2012) we write the IRL Bellman equation as

$$
\begin{aligned}
\mathcal{Q}^\star(\bar{x}(t); u^\star(t); t) &= \mathcal{Q}^\star(\bar{x}(t-\Delta t); u^\star(t-\Delta t); t-\Delta t) \\
&\quad - \frac{1}{2}\int_{t-\Delta t}^{t} (\bar{x}^\top M \bar{x} + u^\top R u) \mathrm{d}\tau \\
\mathcal{Q}^\star(\bar{x}(T); u^\star(T); T) &= \frac{1}{2}\bar{x}^\top(T) P(T) \bar{x}
\end{aligned} \tag{15}
$$

for a small time interval $\Delta t \in \mathbb{R}^+$. We next aim to satisfy the Bellman equation through an iterative online formulation by defining the critic errors $e_{\mathrm{c}_1}, e_{\mathrm{c}_2} \in \mathbb{R}$. We define the first critic error e_{c_1} as

$$
\begin{aligned}
e_{\mathrm{c}_1} &= \hat{Q}(\bar{x}(t); \bar{u}(t); t) - \hat{Q}(\bar{x}(t-\Delta t); \bar{u}(t-\Delta t); t-\Delta t) \\
&\quad + \frac{1}{2}\int_{t-\Delta t}^{t} (\bar{x}^\top M \bar{x} + u^\top R u) \mathrm{d}\tau \\
&= \hat{W}_{\mathrm{c}}^\top [\nu(t)(U(t) \otimes_s U(t)) \\
&\quad - \nu(t-\Delta t)(U(t-\Delta t) \otimes_s U(t-\Delta t))] \\
&\quad + \frac{1}{2}\int_{t-\Delta t}^{t} (\bar{x}^\top M \bar{x} + u^\top R u) \mathrm{d}\tau.
\end{aligned} \tag{16}
$$

The second critic error is defined as

$$
e_{\mathrm{c}_2} = \frac{1}{2}\bar{x}^\top(t) P(T) \bar{x}(t) - \hat{W}_{\mathrm{c}}^\top \nu(t)(U(t) \otimes_s U(t)). \tag{17}
$$

The actor approximation error $e_a \in \mathbb{R}^m$ is defined as

$$
e_a = \hat{W}_{\mathrm{a}}^\top \mu(t)\bar{x} + \hat{Q}_{\mathrm{uu}}^{-1}\hat{Q}_{\mathrm{ux}}(t)\bar{x} \tag{18}
$$

for $\hat{Q}_{\mathrm{uu}}^{-1}$ and $\hat{Q}_{\mathrm{ux}}(t)$ from the critic approximation $\nu(t)^\top \hat{W}_{\mathrm{c}}$. Using adaptive control techniques from Ioannou and Sun (2012) we seek to minimize a cost function based on squared norm errors

$$
\begin{aligned}
K_1(\hat{W}_{\mathrm{c}}, \hat{W}_{\mathrm{c}}(T)) &= \frac{1}{2}||e_{\mathrm{c}_1}||^2 + \frac{1}{2}||e_{\mathrm{c}_2}||^2 \\
K_2(\hat{W}_{\mathrm{a}}) &= \frac{1}{2}||e_{\mathrm{a}}||^2.
\end{aligned} \tag{19}
$$

To minimize the cost function (19), we implement a normalized gradient descent algorithm for the critic and actor weight update law defined by

$$
\dot{\hat{W}}_{\mathrm{c}} = -\alpha_{\mathrm{c}} \frac{\partial K_1}{\partial \hat{W}_{\mathrm{c}}} = -\alpha_{\mathrm{c}} \left(\frac{\sigma}{(1+\sigma^\top\sigma)^2} e_{\mathrm{c}_1} + \frac{\sigma_{\mathrm{f}}}{(1+\sigma_{\mathrm{f}}^\top \sigma_{\mathrm{f}})^2} e_{\mathrm{c}_2} \right) \tag{20}
$$

$$
\dot{\hat{W}}_{\mathrm{a}} = -\alpha_{\mathrm{a}} \frac{\partial K_2}{\partial \hat{W}_{\mathrm{a}}} = -\alpha_{\mathrm{a}} \bar{x} e_{\mathrm{a}}^\top \tag{21}
$$

for $\sigma := \nu(t)(U(t) \otimes_s U(t)) - \nu(t-\Delta t)(U(t-\Delta t) \otimes_s U(t-\Delta t))$, $\sigma_{\mathrm{f}}(t) := \nu(t)(U(t) \otimes_s U(t))$, user-specified critic convergence rate $\alpha_c \in \mathbb{R}^+$, and actor convergence rate $\alpha_a \in \mathbb{R}^+$.

3. APPLICATION TO UAVS

In this section, we discuss how to apply the Q-learning framework to a multirotor UAV test platform used for experimental validation.

A major assumption made for the problem formulation is that (1) is LTI. Therefore, if we apply the proposed Q-learning framework to the full nonlinear dynamical quadrotor model we may not obtain adequate results. One approach is to obtain a linearization of the quadrotor model at the hover state. Another approach is to apply feedback linearization to obtain an equivalent linear system of the quadrotor and apply IRL on the equivalent linear system. This approach is more robust than a linearization at the hover state, which would introduce linearization errors at large orientation offsets from the hover state. However, we found that applying feedback linearization for a quadrotor requires adding auxiliary states because of the relative degree of the quadrotor dynamics and extra model parameters defined by the user while we are looking for a model-free approach.

Instead, we construct the model-free Q-learning formulation to output a mass-normalized force vector $f_{des} = [f_x, f_y, f_z]^\top$ from input position and velocity reference states. We assume that $\|f_{des}\| \neq 0$. We then use the Mellinger controller developed in Mellinger and Kumar (2011) to output attitude and thrust commands to the vehicle with the formulation summarized in the following. The desired mass normalized thrust for the quadrotor is defined as

$$
c = f_{\mathrm{des}}^\top z_B \tag{22}
$$

for vector z_B pointing towards the body z axis of the quadrotor. The desired body z axis for the quadrotor is defined as

$$
z_{B,\mathrm{des}} = \frac{f_{\mathrm{des}}}{\|f_{\mathrm{des}}\|}. \tag{23}
$$

From the desired yaw angle $\psi_{\mathrm{des}}(t)$ we compute the desired body axes $x_{B,\mathrm{des}}$ and $y_{B,\mathrm{des}}$ as

$$
\begin{aligned}
x_{C,\mathrm{des}} &= [\cos\psi_{\mathrm{des}}, \sin\psi_{\mathrm{des}}, 0]^\top \\
y_{B,\mathrm{des}} &= \frac{z_{B,\mathrm{des}} \times x_{C,\mathrm{des}}}{\|z_{B,\mathrm{des}} \times x_{C,\mathrm{des}}\|} \\
x_{B,\mathrm{des}} &= y_{B,\mathrm{des}} \times z_{B,\mathrm{des}}.
\end{aligned} \tag{24}
$$

We then define the desired orientation matrix $R_{\mathrm{des}} = [x_{B,\mathrm{des}}, y_{B,\mathrm{des}}, z_{B,\mathrm{des}}]$. The control commands to the vehicle are expressed as a desired quaternion q_{des} from R_{des} and collective mass normalized thrust c. For simplicity, in our experiments we command a constant desired yaw angle. This control strategy enables the Q-learning controller to learn the finite-horizon optimal control problem for the UAV with only position and velocity states and allowing the Mellinger controller to focus on orientation dynamics. This pipeline enables a UAV to learn the minimum energy to reach a goal pose through an obstacle environment as a model-free formulation because the Mellinger controller does not contain model parameters or vehicle dynamics.

4. EXPERIMENTS

In this section, we demonstrate the proposed model-free Q-learning control algorithm in a real world flight experiment with a 700g quadrotor shown in Fig. 1. The UAV contains a Jetson TX2 on-board computer and a flight controller (FC) running PX4 autopilot with an inertial measurement unit (IMU). The on-board computer runs ROS2 and receives external position measurements from OptiTrack motion capture cameras. The external position measurements are sent to the FC and processed by the PX4 extended Kalman filter estimation module. The motion planning and Q-learning control algorithm is

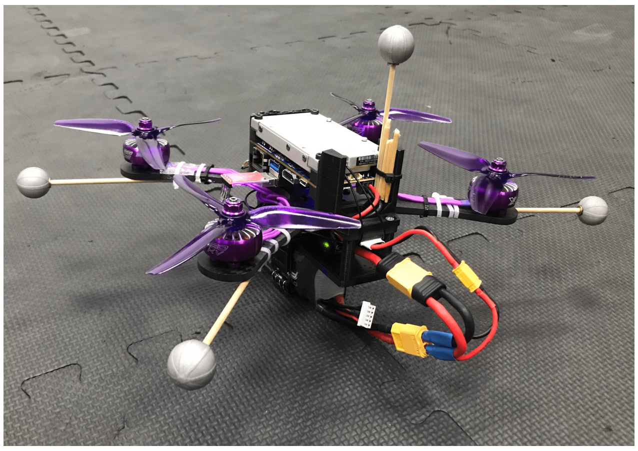


Fig. 1. Photograph of a 700g quadrotor used in the flight experiment. A Jetson TX2 single-board computer running ROS2 is connected to a flight controller running PX4 autopilot for state estimation and motor control.

run on ROS2 to send attitude and thrust commands to the FC. The FC then uses a low-level body-rate controller to track the attitude commands. We also incorporate a thrust model so that we can map normalized thrust commands to electronic speed controller commands.

4.1 Algorithm Overview

In this section, we discuss the algorithmic implementation of the Q-learning framework for UAVs. The code that accompanies this work is designed as a complete flight stack running on the Jetson TX2 on-board computer. [1]

In Fig. 2 we illustrate the code architecture as a block diagram. The navigator, planner, and controller are designed as servers with individual implementations running as plugins. For example, static predefined obstacles and motion capture tracked obstacles are custom plugins in the mapping server, RRT$^{\text{X}}$ is a plugin in the planner server, Q-learning is a plugin in the control server, and navigate-to-pose is a plugin in the navigator server amongst others such as takeoff, land, and loiter. We interface with the vehicle through the vehicle interface node. The vehicle interface node is also designed to be the bridge between the flight stack and other autopilot software such as PX4 or directly simulated data. This isolates the flight stack plugins and servers from direct dependency on specific autopilot software. One can design a new plugin within the vehicle interface to interface with other autopilots, e.g., Betaflight, ArduPilot, or a custom-built autopilot.

The procedure for testing a UAV with our flight stack is as follows. First, place the UAV on the desired takeoff location. Next, turn on the flight controller and autopilot software. Then, we startup our custom flight stack through a ROS2 launch file in the vehicle interface package. Through the RViz2 application there will be a custom graphical user interface (GUI) panel plugin for interfacing with the vehicle. Once the vehicle is safely positioned at the takeoff location, then the kill-switch can be turned off,

[1] The code and video that accompanies this work is publicly available through the GaTech FACTS Lab Github: `github.com/gtfactslab/Llanes_MECC2023`.

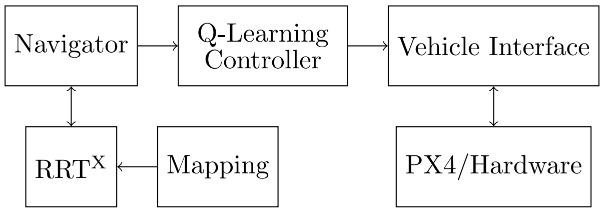


Fig. 2. Block diagram illustrating experiment code architecture.

and the vehicle armed. Select the takeoff command from the GUI panel. The UAV should now request to navigator to takeoff and switch to loiter at the user-defined takeoff altitude in the YAML parameters file. If the navigator is in loiter mode, then a goal command can be requested from the RViz2 *2D Goal Pose* tool. This will create an action request from the navigator action server to plan with RRT$^{\text{X}}$ around obstacles defined in the mapping node. The navigator or planner can deny a request if the goal intersects obstacles, the plan is infeasible, or the navigator is in the incorrect transition mode.

Algorithm 1 Navigate-to-Pose with RRT-Q$^{\text{X}}$

Input: T - finite horizon; t_i - timestamp at iteration i; M - state cost penalty matrix; R - control cost penalty matrix; $P(T)$ - terminal cost matrix; x_{start} - start state; x_{goal} - goal state; $\mathcal{X}_{\text{obs}}$ - obstacle map; $\mathcal{X}$ - state space; q - orientation quaternion

Output: q_{des} - desired orientation quaternion; c_{des} - desired mass-normalized thrust

```
1:  $\pi \leftarrow \text{RRT}^{\text{X}}(\mathcal{X}, \mathcal{X}_{\text{obs}}, x_{\text{start}}, x_{\text{goal}})$
2:  **if** $\pi$ is valid **then**
3:      NavigatorMode ← NavigateToPose
4:      $x_{\text{r}} \leftarrow \pi[0]$
5:      $i \leftarrow 0$
6:      $k \leftarrow 0$
7:  **while** NavigatorMode is NavigateToPose **do**
8:      $i \leftarrow i + 1$
9:      $\Delta t \leftarrow t_i - t_{i-1}$
10:     $\bar{x} \leftarrow x - x_{\text{r}}$
11:     $\hat{W}_{\text{c}} \leftarrow \text{Critic}(M, R, \Delta t, t_k, \alpha_{\text{c}}, \bar{x}, \hat{u})$ (20)
12:     $\bar{Q} \leftarrow \text{EstimateQ}(\hat{W}_{\text{c}}, t_k)$
13:     $\hat{W}_{\text{a}} \leftarrow \text{Actor}(\bar{Q}, \Delta t, t_k, \alpha_{\text{a}}, \bar{x})$ (21)
14:     $\hat{u} \leftarrow \text{Control}(\hat{W}_{\text{a}}, t_k, \bar{x})$ (14)
15:     $f_{\text{des}} \leftarrow \hat{u}$
16:     $(q_{\text{des}}, c_{\text{des}}) \leftarrow \text{Mellinger}(f_{\text{des}}, \psi_{\text{des}}, q)$ (22)-(24)
17:     **return** $(q_{\text{des}}, c_{\text{des}})$
18:     **if** $x_{\text{r}}$ reached and $x_{\text{r}} \neq x_{\text{goal}}$ **then**
19:         $k \leftarrow k + 1$
20:         $x_{\text{r}} \leftarrow \pi[k]$
21:     **else if** $x_{\text{r}}$ reached and $x_{\text{r}} = x_{\text{goal}}$ **then**
22:         NavigatorMode ← Loiter
```

The implementation of the navigate-to-pose navigator mode is presented as pseudocode in Algorithm 1. The navigator node has three default navigator modes: takeoff, land, and loiter. Loitering is a position hold mode where the UAV hovers at the coordinates where the loiter request is made. The navigate-to-pose navigator mode can be activated in a loiter mode or from cancelling another active navigation mission such as an active navigate-to-pose. When the navigate-to-pose mode is requested then the code follows a similar routine as in Algorithm 1. In Line 1 of Algorithm 1, we request a path from RRT$^{\text{X}}$. In

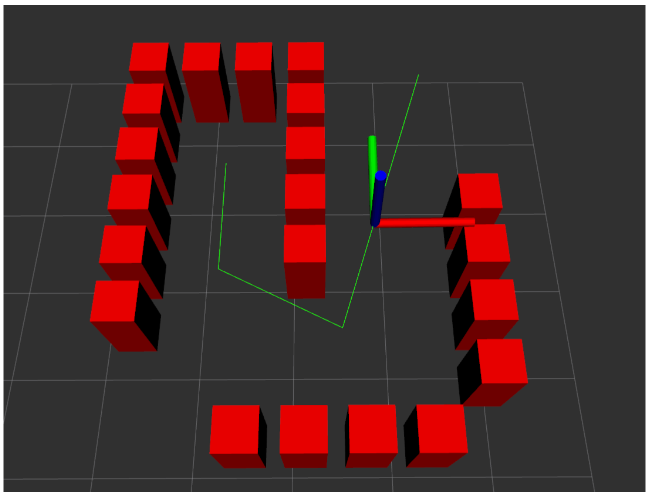


Fig. 3. Screenshot from a simulation of a quadrotor UAV traversing obstacles to reach a goal state from collision-free waypoints generated from RRT^X. The waypoints are inputs to the model-free Q-learning controller. The visualization software in this image is RViz2 from ROS2.

Line 2, we check whether a path was returned from the planner and if true then we switch the navigator mode to navigate-to-pose. Then, in Line 7 while the navigator mode is in navigate-to-pose we publish the current navigator setpoint as intermediate waypoints in the path computed by RRT^X to the Q-learning control node. The Q-learning controller is updated when a new current position message is received. For an iteration we compute the time since last update in Line 9 and the Q-learning update procedure in Lines 11-14. The computed control represents mass-normalized force commands to the UAV which are turned into desired orientation and collective mass-normalized thrust from the Mellinger controller in Line 16. We then check whether we have reached the reference setpoint x_r. If the setpoint is the goal, then the navigator switches to loiter mode. Otherwise, the next waypoint in the path is used as the new setpoint.

4.2 Experiment Results

In this section, we discuss the results of the simulation and real-hardware flight experiments. We first tested the implementation in a Gazebo simulation environment using the PX4 software-in-the-loop (SITL) tools with ROS2. We generate an obstacle environment for the RRT^X algorithm to compute a collision-free path. The visualization of this environment is shown in Fig. 4.

The simulation obstacle environment is reconstructed in our indoor flight lab as shown in Fig. 4. Updates for the Q-learning estimated parameters are done upon receiving new odometry data from the UAV which is approximately updated at $125Hz$. The parameters used in the simulation and hardware demonstration are critic convergence rate $\alpha_c = 10$, actor convergence rate $\alpha_c = 0.5$, finite horizon $T = 3$ seconds, $\Delta t \approx 0.008$ seconds, $P(T) = \text{diag}(100, 100, 100, 10, 10, 10)$, $R = \text{diag}(20, 20, 20)$, and $M = \text{diag}(10, 10, 10, 30, 30, 30)$.

In Fig. 5, we provide plots for the tracking performance of the learning controller on the hardware flight experiment

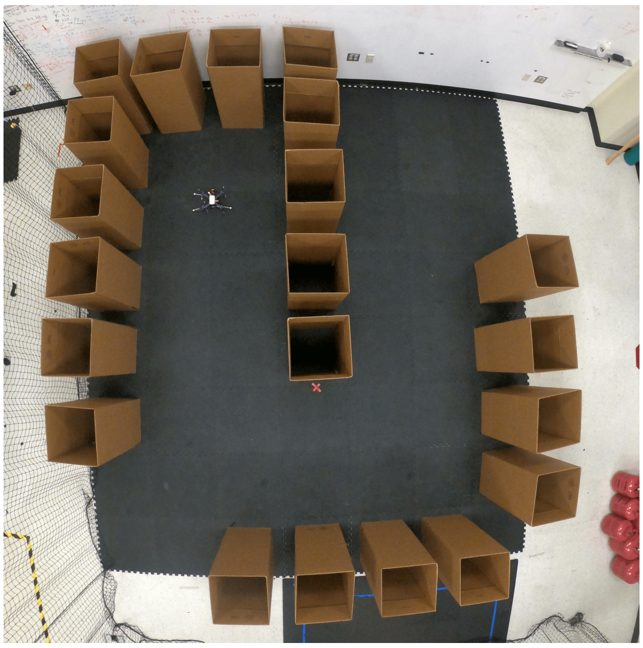


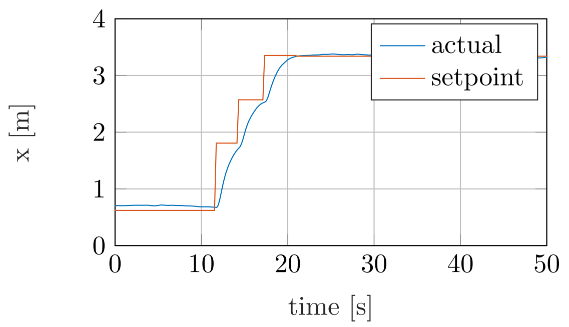
Fig. 4. Screenshot from the real hardware flight experiment in the indoor flight lab. A quadrotor UAV is commanded to reach a goal state from a collision-free path generated from RRT^X. The waypoints are inputs to the model-free Q-learning controller. This demonstration is a recreation of the simulated environment with cardboard boxes as obstacles.

as position time histories of the quadcopter tracking a collision-free path. The learning controller shows satisfactory steady-state error for the setpoints in x and y coordinates. There is a small error in the altitude towards the end of the hardware flight experiment when the battery voltage decrement causes a mismatch in the thrust model. We plan to address this in the future by incorporating a more sophisticated hybrid learning with first-principles thrust model as shown in Bauersfeld et al. (2021).

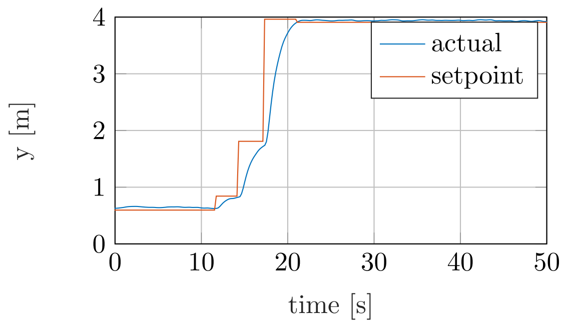
We also provide plots of the time history of $\hat{Q}$ in Fig. 6. The value of $\hat{Q}$ spikes when a new waypoint reference is commanded from the collision-free path and converges to zero as the state tracking error converges to zero.

5. CONCLUSION

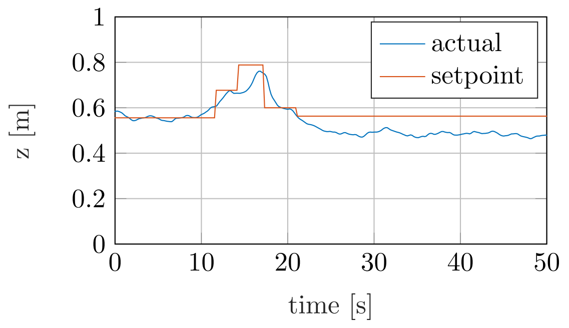
This work presents an algorithm for real-time kinodynamic motion planning in dynamic environments for systems with unknown dynamics. We use a random-sampling and space-filling tree for both planning and rapid replanning in the dynamic environment. We demonstrate our results in a custom-built quadrotor UAV simulation and hardware flight experiment to verify the real-time capabilities of the algorithm. We demonstrated collision-free tracking performance on a UAV hardware flight experiment with a learning-based controller.



(a) Quadrotor actual and setpoint x position vs. time



(b) Quadrotor actual and setpoint y position vs. time



(c) Quadrotor actual and setpoint z position vs. time

Fig. 5. Position flight data from real flight experiment for local indoor flight lab reference frame. Each coordinate axis actual position is plotted separate along with setpoints requested from navigator and stored from RRT^X path.

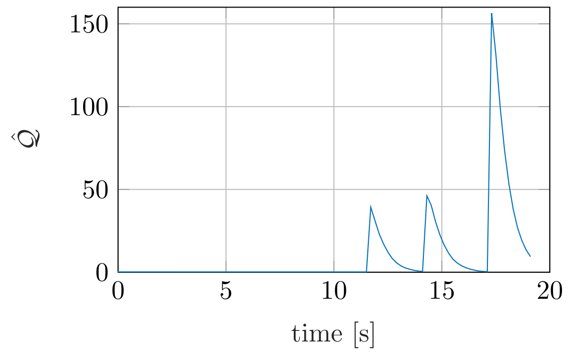


Fig. 6. Plot of $\hat{\mathcal{Q}}$ versus time from real flight experiment flight data. The data is cut off at the end when the navigator switches to loiter mode after successfully reaching the goal.

REFERENCES

Bauersfeld, L., Kaufmann, E., Foehn, P., Sun, S., and Scaramuzza, D. (2021). NeuroBEM: Hybrid aerodynamic quadrotor model. Robotics: Science and Systems 2021. doi:10.15607/RSS.2021.XVII.042.

Donald, B., Xavier, P., Canny, J., and Reif, J. (1993). Kinodynamic motion planning. *Journal of the ACM*, 40(5), 1048–1066.

Ioannou, P.A. and Sun, J. (2012). *Robust Adaptive Control*. Courier Corporation.

Kontoudis, G.P. and Vamvoudakis, K.G. (2019). Kinodynamic motion planning with continuous-time Q-learning: An online, model-free, and safe navigation framework. *IEEE Transactions on Neural Networks and Learning Systems*, 30(12), 3803–3817.

Kontoudis, G.P., Vamvoudakis, K.G., and Xu, Z. (2022). RRT-Q^X real-time kinodynamic motion planning in dynamic environments with continuous-time reinforcement learning. In *Brain and Cognitive Intelligence Control in Robotics*, 1–19. CRC Press.

Kontoudis, G.P., Xu, Z., and Vamvoudakis, K.G. (2020). Online, model-free motion planning in dynamic environments: An intermittent, finite horizon approach with continuous-time Q-learning. In *2020 American Control Conference (ACC)*.

Lewis, F.L., Vrabie, D., and Syrmos, V.L. (2012). *Optimal Control*. John Wiley & Sons,, 3 edition.

Mellinger, D. and Kumar, V. (2011). Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, 2520–2525. doi:10.1109/ICRA.2011.5980409.

Netter, J., Kontoudis, G.P., and Vamvoudakis, K.G. (2021). Bounded rational RRT-Q^X: Multi-agent motion planning in dynamic human-like environments using cognitive hierarchy and Q-learning. In *2021 60th IEEE Conference on Decision and Control (CDC)*, 3597–3602. doi:10.1109/CDC45484.2021.9683761.

Otte, M. and Frazzoli, E. (2014). RRT^X: Real-time motion planning/replanning for environments with unpredictable obstacles. In *WAFR*.

Otte, M. and Frazzoli, E. (2016). RRT^X: Asymptotically optimal single-query sampling-based motion planning with quick replanning. *The Int. J. of Rob. Res.*, 35(7), 797–822.

Schäcke, K. (2013). On the kronecker product.

Sutton, R.S. and Barto, A.G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Vamvoudakis, K.G. and Lewis, F.L. (2009). Online actor critic algorithm to solve the continuous-time infinite horizon optimal control problem. In *2009 International Joint Conference on Neural Networks*, 3180–3187. doi:10.1109/IJCNN.2009.5178586.

Vrabie, D., Vamvoudakis, K.G., and Lewis, F. (2012). *Optimal Adaptive Control and Differential Games by Reinforcement Learning Principles*.