Proceedings of the 38th Chinese Control Conference
July 27-30, 2019, Guangzhou, China

Control based motion primitives for quadrotor trajectory generation

Shupeng Lai[1], Menglu Lan[2], Ben M. Chen[1,3]

1. Department of Electrical & Computer Engineering, National University of Singapore, Singapore 117583.
E-mail: elelais@nus.edu.sg

2. Graduate School for Integrative Science & Engineering, National University of Singapore, Singapore 119077.

3. Department of Mechanical and Automation Engineering, Chinese University of Hong Kong, Shatin, N.T., Hong Kong.

Abstract: The local motion planning is critical for the safe operation of an autonomous vehicle such as the quadrotor in which motion primitives are usually adopted to find valid and efficient reaction. We present in this work a method that is capable of generating a class of motion primitives for the quadrotor. Compared to the previous approaches, these primitives are capable of minimizing multiple types of cost functions while satisfying the motion constraints of the quadrotor. Moreover, neural networks are adopted to approximate the generated motions to further boost the online evaluation efficiency. Finally, the local motion planning is achieved through solving a non-convex optimization problem with particle swarm techniques in a receding horizon fashion. The optimization process is a natural extension of the previous tree searching based methods. The proposed approach is computationally efficient and has been tested in a real environment on an actual quadrotor platform.

Key Words: Motion planning, Trajectory generation, Motion primitives, Unmanned aerial systems

1 Introduction

Micro-sized aerial vehicles such as the quadrotors are usually required to navigate in a GPS-denied, obstacle-strewn environment. A motion planning module is necessary to generate a collision-free and dynamically feasible reference trajectory to guide the vehicle traveling safely towards its desired target area. Due to the limited onboard computational resources, motion planning is usually implemented in a hierarchical structure with a global planner and a local planner. The global planner operates on a simplified vehicle model to search in the entire workspace for an optimal plan with unlimited time horizon. Because of the simplified vehicle model and the unlimited time horizon, its working frequency is usually much lower than that of the local planner, and the generated reference cannot be tracked by the lower level controllers directly. On the other hand, the local planner considers a detailed vehicle model. It takes the reference from the global planner and uses it as heuristics to plan a dynamically feasible trajectory that is collision free within a limited time horizon. The local planner is required to be fast and robust against the change in the environment to always provide a valid trajectory.

The quadrotor is omnidirectional and capable of traveling in any arbitrary directions regardless of its heading angle. Therefore, it is usually simplified as a single integrator during the global planning stage, and classical methods such as Dijkstra, A*, D* and FMT* can be directly applied to plan its global path efficiently [1–3]. The resulting reference by these global planners is a series of interconnected line segments that links the vehicle's current position to the target point. On the other hand, the local planner of the quadrotor is more challenging to design. Because the vehicle, even after the simplification with differential flatness, processes a high ordered dynamics with various constraints on its states and inputs. Moreover, due to the limited computational resources, the sensed obstacles are usually represented as point clouds or grid maps instead of geometrical bounding polygons. Therefore, the local planner is required to search for valid trajectories directly over the point clouds or grid maps, which renders the resulting problem usually non-linear and non-convex.

On the other hand, motion primitives are frequently used in finding valid local trajectories for mobile robots by growing a local search tree from the current state of the vehicle to a set of states at or near the planning horizon [4]. Possible trajectories on the local search tree are evaluated against a predefined cost function, and then the best collision-free trajectory is selected as the reference. The process is repeated in a receding horizon fashion to deal with the uncertainties in the environment. The motion primitives can be generated by directly sampling in the vehicle's input space and performing a forward simulation. However, due to the non-linearity and constraints of the vehicle model, it could be computationally expensive to acquire a well-separated search tree in the state space. Alternatively, one can sample in the space of the vehicle's boundary state constraints and generate the motion by solving a boundary value problem. For the real-time performance, the motion primitives need to be evaluated efficiently. In the case of the quadrotor, some frequently used motion primitives include:

- Linearly changed acceleration [5]: The acceleration is linearly changed from its current value to the desired end value in a fixed duration.
- Time optimal motion [6, 7]: A time-optimal trajectory that brings the vehicle from its current state to the desired end velocity or end position while satisfying constraints on its velocity, acceleration, and jerk.
- Time-averaged jerk minimum polynomial [8]: A fifth order polynomial that connects the vehicle's current state to a fixed end state of arbitrary position, velocity, and acceleration while minimizing the time-averaged integration of the square of the jerk.
- Forward-arc motion primitives [9]: Originally constructed for ground vehicles, but also adapted for quadrotor usage by extending the smoothness up to the acceleration.

The primitives mentioned above have all been successfully implemented on real vehicles. However, in quest of the computational efficiency, [6, 7] use indirect methods that can only solve time optimal problems. The resulting trajectory could be unnecessarily aggressive due to the bang-zero-bang input strategy. The similar problem can also be found in [5]. In [8], the motion constraints are ignored to achieve a closed-form solution. An additional validation process is introduced to select the valid trajectories at online. However, as shown in [10], the chance of finding a valid trajectory drops significantly when the cost function is modified. In [9], the fixed motion primitive library that is sufficiently close to the initial condition is adopted to generate the motion, which causes the reference trajectory to be non-continuous.

In this paper, we present a novel method to construct motion primitives for quadrotors with dynamic programming generated controllers. The primitives are first generated offline and approximated with a neural network (NN) to boost the online evaluation efficiency. Furthermore, a dynamically evolving local search tree is constructed with the particle swarm optimization (PSO) to solve the motion selection problem in the continuous domain.

The rest of the paper is organized as follows: In Section 2, we introduce the related works for quadrotor motion planning, especially the local motion planning. An overview of our method is given in Section 3. In Section 4, we present how to generate the proposed motion primitives with dynamic programming designed controllers. These primitives are then used in a PSO algorithm to find the desired motion. In Section 6, the data and implementation details for the flight experiment is covered. Finally, in Section 7 concluding remarks are given.

2 Related works

The literature on trajectory generation for quadrotors are extensive. In [11], the author adopted a minimum snap trajectory that is proven to satisfy the underlying dynamic constraints of the quadrotor. The method is later simplified in [12] that also combines it with a rapid random tree global planner to guide the vehicle flying in indoor environments with pre-planned trajectory. For online sensor-based navigation in similar environments, [13] constructs a safe flying corridor using cubics. The non-convex optimization is made into a quadratic one by separating the constraints in time. The technique is further improved in [1] to allow the corridor to be constructed by arbitrarily shaped convex polygons. In [14], the constraints are made to be satisfied on a continuous time interval instead of individual points. However, the corridor is usually constructed based on the reference of the global planner that ignores the detailed dynamics. It is possible that the vehicle's initial condition is in an inevitable collision state with the corridor and no solution could be found. To address this issue, [15] choose to directly optimize on the cost map that is constructed through Euclidean distance transform. However, even with the reference from the global planner, the failure rate is still high due to the non-linear nature of the resulting optimization problem.

On the other hand, motion primitive based techniques are also widely used, especially in the local planning stage of the quadrotor. Due to the relatively short prediction horizon of the motion primitive based methods, they are usually executed in a receding horizon fashion, similar to the one used by model predictive control. In [8], a minimum jerk polynomial spline is used to guide the vehicle to strike a ball. However, it ignores the limits on the trajectory's derivatives. The work in [16] proposes to generate the trajectory by solving a time-optimal control problem with an indirect method. The vehicle can be regulated to the desired target point with hover condition while satisfying constraints on its velocity, acceleration, and jerk. The work in [3] uses the time-optimal motion primitives to navigate the quadrotor in an obstacle-strewn environment. In [7], the time-optimal primitives are relaxed to satisfy only the end velocity constraints and are used to navigate the quadrotor with a limited field of view. The time-optimal primitive can be further simplified to reach the desired acceleration with a constant jerk. The major drawback of the time-optimal primitives is that they usually lead to unnecessarily aggressive maneuvers. In [9], forward arch motion primitives are used for quadrotors but are limited to a fixed motion library sets. To guarantee the smoothness of the trajectory, the motion is generated with the motion libraries that is sufficiently close to the vehicle's current state. It requires to sample in the motion's initial condition densely and creates a large lookup table.

3 Method overview

In this section, we provide an overview of the proposed method that constructs motion primitives and uses them for in online motion planning. According to [6], the quadrotor can be simplified into a triple integrator, and all other state/input constraints can be satisfied by limiting the triple integrator's velocity, acceleration, and jerk along every single axis. Assume $\mathbf{p}, \mathbf{v}, \mathbf{a}, \mathbf{j}$ are all 3×1 vectors representing the position, velocity, acceleration and jerk of the quadrotor respectively, we consider the following quadrotor model:

$$\begin{aligned} \dot{\mathbf{p}} &= \mathbf{v} \\ \dot{\mathbf{v}} &= \mathbf{a} \\ \dot{\mathbf{a}} &= \mathbf{j} \end{aligned} \tag{1}$$

where $\mathbf{x} = [\mathbf{p}^\intercal, \mathbf{v}^\intercal, \mathbf{a}^\intercal]^\intercal$ is the state of the quadrotor and $\mathbf{u} = \mathbf{j}$ is input. According to [6], the state and input constraints, such as the maximum thrust and the maximum body rate can be satisfied by having

$$\begin{aligned} \mathbf{v}_{\min} &\le \mathbf{v} \le \mathbf{v}_{\max} \\ \mathbf{a}_{\min} &\le \mathbf{a} \le \mathbf{a}_{\max} \\ \mathbf{j}_{\min} &\le \mathbf{j} \le \mathbf{j}_{\max}. \end{aligned} \tag{2}$$

Let $\mathcal{O}$ denotes the obstacle set, $\mathbf{x}(t), \mathbf{u}(t)$ represents the state and input trajectory, then the local motion planning problem can be effectively written as a model predictive control problem that finds the state and input trajectories $\mathbf{x}(t), \mathbf{u}(t)$ to minimize

$$J = E(\mathbf{x}(t_0 + T)) + \int_{t_0}^{t_0+T} L(\mathbf{x}(t), \mathbf{u}(t))dt \tag{3}$$

subject to the constraints in Equation 1, 2, the collision free condition

$$\mathbf{p} \notin \mathcal{O}, \tag{4}$$

and the initial condition

$$\mathbf{x}(t_0) = \mathbf{x}_0. \tag{5}$$

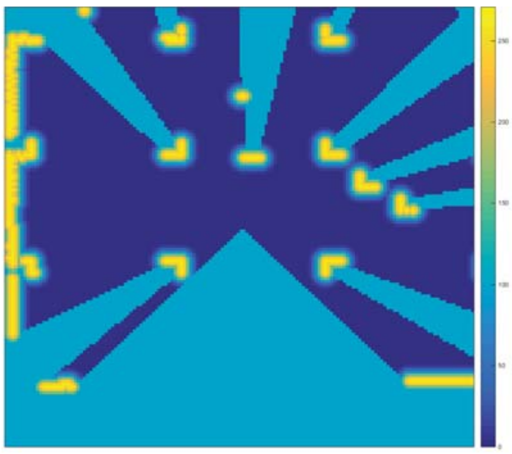

Fig. 1: Cost map constructed with a laser scanner. The lighter the color, the higher the cost.

If the $\mathbf{x}(t)$ and $\mathbf{u}(t)$ are directly discretized over the prediction horizon T, it usually gives a high dimensional optimization problem that needs to be solved with gradient-based method for real-time performance. In sensor-based navigation, the environment is usually represented as cost maps (see Figure 1). It is a non-trivial task to construct an explicit mathematical model for the obstacle set $\mathcal{O}$ from the cost map. Therefore, the collision-free condition $\mathbf{p} \notin \mathcal{O}$ is usually achieved as a soft-constraints $O(\mathbf{x})$ that is added into the cost function of Equation 3. The final problem is now to minimize

$$J = E(\mathbf{x}(t_0+T)) + \int_{t_0}^{t_0+T} L(\mathbf{x}(t), \mathbf{u}(t)) + O(\mathbf{x}(t))dt \quad (6)$$

which is usually non-linear and non-convex due to the $O(\mathbf{x}(t)$. The efficiency and reliability of solving non-linear programming problem in Equation 6 still remains an open problem.

In this paper we design non-linear controllers with dynamic programming that is capable of driven the system in Equation 1 to a desired point θ_p or velocity $theta_v$ subject to the constraints in Equation 2. Assume the initial state is $\mathbf{x}_0$. Let $\hat{\mathbf{x}}(t)$, $\hat{\mathbf{u}}(t)$ denotes the state and input trajectories generated by regulating the system to θ_p with controller π_p, there is a mapping relationship

$$\langle \hat{\mathbf{x}}(t), \hat{\mathbf{u}}(t) \rangle = M_p(\mathbf{x}_0, \theta_p). \quad (7)$$

Similarly, if it is regulated to θ_v, there is

$$\langle \hat{\mathbf{x}}(t), \hat{\mathbf{u}}(t) \rangle = M_v(\mathbf{x}_0, \theta_v). \quad (8)$$

By fixing the x_0 as the current state of the vehicle, $\hat{\mathbf{x}}(t)$, $\hat{\mathbf{u}}(t)$ are then dependent on θ_p or θ_v only. It is now possible to modify the cost function in 6 as

$$J_\theta = \bar{E}(\theta) + \int \bar{L}(\theta)dt, \quad (9)$$

where θ is used to express either θ_v or θ_p. Since θ_v and θ_p are both 3×1 vectors, the re-parameterization significantly reduces the problem's dimension. Though the cost in Equation 9 is still non-linear and non-convex, evolutionary based algorithms such as the PSO can be adopted to solve the problem due to its lower dimension.

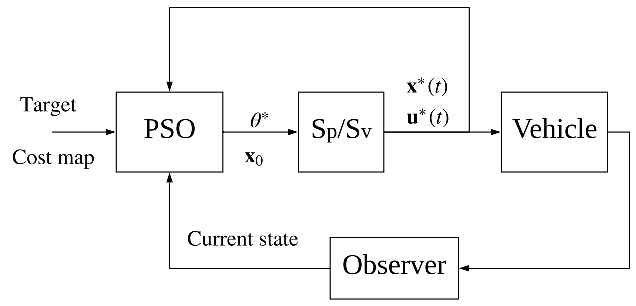

Fig. 2: The proposed frame work of using controller generated motion primitives by optimizing the desired target point/velocity.

To further improve the online computational efficiency, NNs $S_{p\text{NN}}$ and $S_{v\text{NN}}$ are used to approximate the mapping in Equation 7 and 8 during the PSO. The outputs from the optimization process are the optimal set point θ_p^* or velocity θ_v^*. In order to avoid the noise introduced by the NN when generating the actual reference and nominal input. The θ_p^* and θ_v^* are then plugged back into S_p and S_v to be evaluated by forward simulation with the corresponding controller. A block diagram of the proposed approach can be visualized in Figure 2.

4 Control based motion primitives

In this section, we present to general motion primitives with controllers that are automatically generated by solving a Markov decision process (MDP) with value iteration. The method is adopted because it could generate multiple types of discrete optimal controllers that is capable of handling the state and input constraints in Equation 2 while minimizing different types of cost functions. We first demonstrate the controller design and the NN approximation with the position setpoint controller. Then, we show the procedure can be easily extended to design a similar velocity controller. The controllers are designed independently for each axis of the vehicle. For convince, we use p, v, a, j to represent the corresponding single-axis counterpart of the $\mathbf{p}$, $\mathbf{v}$, $\mathbf{a}$ and $\mathbf{j}$. They forms a single axis triple integrator as $\dot{p} = v$, $\dot{v} = a$, $\dot{a} = j$ with state $s = [p, v, a]^\intercal$ and input $u = j$.

4.1 System Discretization

To obtain the MDP model, we first discretize the state and input space of the single-axis triple integrator. Let P, V, A, J denotes the finite set of uniformly discretized positions, velocities, accelerations and jerks in a predefined region of interest. Together, they form the discrete state set $\mathcal{D}$ with

$$\mathcal{D} = \{[p, v, a]^\intercal | p \in P, v \in V, a \in A\}.$$

Given an initial state $s_0 = [p_0, v_0, a_0]^\intercal \in \mathcal{D}$, if we apply a constant jerk input j_0 over a fixed time interval Δt, the resulting end state $s_t = [p_t, v_t, a_t]^\intercal$ can be expressed as:

$$s_t = As_0 + bj_0 \quad (10)$$

where

$$A = \begin{bmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix}, b = \begin{bmatrix} \frac{\Delta t^3}{6} \\ \frac{\Delta t^2}{2} \\ \Delta t \end{bmatrix}.$$

Due to the discretization, it is possible that $s_t \notin S$. In such cases, it is approximated linearly with its nearest discrete neighbours. The procedure effectively constructs a MDP over $\mathcal{D}$. For the state $s_t = [p_t, v_t, a_t]^\intercal$, let $p_a, p_b \in P$, $v_a, v_b \in V$ and $a_a, a_b \in A$ be the nearest neighbours of p_t, v_t, a_t respectively with

$$\begin{aligned} p_a \le p_t \le p_b \\ v_a \le v_t \le v_b. \\ a_a \le a_t \le a_b \end{aligned}$$

Let

$$\begin{aligned} \bar{s}_1 &= [p_a, v_a, a_a]^\intercal & \bar{s}_2 &= [p_b, v_a, a_a]^\intercal \\ \bar{s}_3 &= [p_a, v_b, a_a]^\intercal & \bar{s}_4 &= [p_b, v_b, a_a]^\intercal \\ \bar{s}_5 &= [p_a, v_a, a_b]^\intercal & \bar{s}_6 &= [p_b, v_a, a_b]^\intercal, \\ \bar{s}_7 &= [p_a, v_b, a_b]^\intercal & \bar{s}_8 &= [p_b, v_b, a_b]^\intercal \end{aligned}$$

then we can approximate s_t linearly as

$$s_t = \sum_{i=1}^{8} w_i \bar{s}_i \tag{11}$$

where

$$w_{9-i} = \frac{\eta(s_t - \bar{s}_i)}{\eta(\bar{s}_8 - \bar{s}_1)}, \quad \forall i \in \{1, 2, \dots, 8\}$$

and

$$\eta(s) = s_p \cdot s_v \cdot s_a.$$

with s_p, s_v, s_a being the position, velocity and acceleration component of state s.

4.2 Value Iteration

In dynamic programming, a value function $V(s)$ is used to approximate the remaining integrated cost from s to the target state. For states s, s_t have $s_t = As + bj$, their value functions shall satisfy $V(s) = C(s,j) + V(s_t)$. Here, $C(s,j)$ represent the cost of taking input j for Δt seconds starting from s. It is often called the instantaneous cost. Similar to Equation 11, a linear approximation is made for $V^*(s_t)$

$$V(s_t) = \sum_{i=1}^{8} w_i V(\bar{s}_i) \tag{12}$$

where w_i and $\bar{s}_i$ are defined in Equation 11. The goal of the dynamic programming is to find a policy map for j that minimize the value function for all $s \in \mathcal{D}$. According to [17], the minimum $V(s)$ can be achieved if

$$V(s) = \min_{j \in J} C(s,j) + V(s_t)$$

is satisfied for all $s \in \mathcal{D}$. Substitute $V(s_t)$ with Equation 12, there is

$$V(s) = \min_{j \in J} C(s,j) + \sum_{i=1}^{8} w_i V(\bar{s}_i) \tag{13}$$

Algorithm 1 Value iteration

```
1: $e = \inf, V(s) = 0 \; \forall s \in S$
2: while $e > \Delta$ do
3:     for $s \in S$ do
4:         $V'(s) = \min_{j \in J} C(s,j) + \sum_{i=1}^{8} w_i V(\bar{s}_i)$
5:         $P(s) = \operatorname{argmin}_{j \in J} C(s,j) + \sum_{i=1}^{8} w_i V(\bar{s}_i)$
6:     $e = max(abs(V(s) - V'(s))), \quad \forall s \in S$
7:     $V(s) = V'(s), \quad \forall s \in S$
```

Equation 13 is solved with an value iteration process. The details can be found in Algorithm 1. Once the value iteration has converged, the policy map $\pi_{\mathcal{D}}(s)$ for all $s \in \mathcal{D}$ is acquired. For an arbitrary state s_p, its action is approximated as

$$\pi(s) = \sum_{i=1}^{8} w_i \pi_{\mathcal{D}}(\bar{s}_i) \tag{14}$$

where w_i and $\bar{s}_i$ are defined in Equation 11.

4.3 Instantaneous cost

If the generated policy is to achieve certain desired behaviors, such as reaching a target state while satisfying the state constraints, the instantaneous cost $C(s,j)$ is to be designed accordingly. Here, we present four types of cost functions that are used for the quadrotor.

1) Target deviation: Without sacrificing the generality, we set up an objective function to regulate the triple integrator to the origin

$$Q(s) = s^\intercal Q s. \tag{15}$$

The resulting policy can still be used guide the vehicle to any set-point through coordinate shifting.

2) Input penalty: Similar to the linear quadratic regulator, we also penalize the system from taking aggressive actions using the cost:

$$R(j) = j^2 R. \tag{16}$$

3) Time penalty: If it is desirable to penalize the total time spent on bringing the system to the origin, the cost $\Phi(s)$ can be included:

$$\psi(s) = \begin{cases} 1, & if s \notin \mathbb{O} \\ 0, & if s \in \mathbb{O} \end{cases} \tag{17}$$

where $\mathbb{O}$ is a set at the vicinity of the origin and contains the origin. It works by penalizing the number of steps that takes the system to the origin. Since each step takes a fixed amount of time Δt, it also minimizes the total time.

4) State and input constraints: The limit on the velocity, acceleration and jerk are satisfied as soft constraints by including the following cost: It is possible to include soft constraints to to the desired range by including the following cost function

$$\begin{aligned} \theta(s,j) = &Q_1 \kappa(s_v, v_{\min}, v_{\max})^2 + Q_2 \kappa(s_a, a_{\min}, a_{\max})^2 \\ &+ R_1 \kappa(j, j_{\min}, j_{\max})^2 \end{aligned} \tag{18}$$

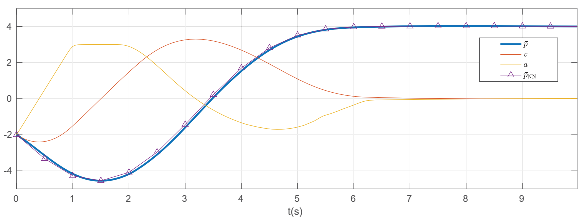

Fig. 3: Non-linear controller generated using dynamic programming

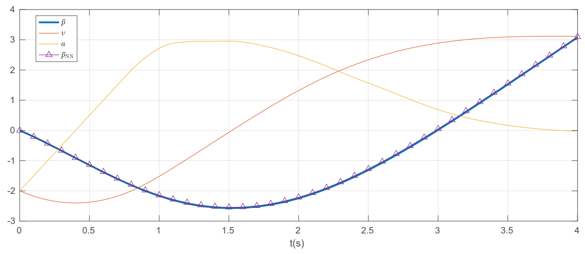

Fig. 4: Non-linear controller generated using dynamic programming

where

$$\kappa(x, x_{\min}, x_{\max}) = \begin{cases} x_{\min} - x, & if x < x_{\min} \\ x - x_{\max}, & if x > x_{\max} \\ 0, & else \end{cases}$$

Finally, the cost function $C(s, j)$ is a weighted combination of the costs mentioned above

$$C(s, j) = Q(s) + R(j) + \bar{w}\psi(s) + \theta(s, j). \tag{19}$$

With the given instantaneous cost, we can finally design a controller for each individuals axis of the vehicle.

4.4 NN approximation

Given a controller in Equation 14 for each axis of the quadrotor, we can now regulate it to the desired set point θ_p. The resulting trajectory is dependent only on the initial state $\mathbf{x}_0$ and θ_p. The relationship is expressed as a map in Equation 7. In order to fully utilize the modern parallel hardware, we propose to use a NN to approximate the S_p with the $S_{p\text{NN}}$. Similar to the controller, one approximation NN is used for each axis. The input of this NN includes the initial state s_0 and the set point s_e. Its output contains the future trajectory of the system sampled 2 Hz for 20 seconds. In order to evaluate the total input effort, the output also includes the integration of the square of the jerk. A comparison of the predicted and the original trajectory can be found in Figure 3.

4.5 Extension to the velocity controller

The extension to the velocity controller can be achieved by constructing a double integrator system for each axis $\dot{v} = a$, $\dot{a} = j$. The goal is to regulate the double integrator to a desired velocity θ_v. The control policy can be easily obtained by repeat the dynamic programming process with omitted position state p. A comparison of the predicted and the original trajectory of the velocity controller can be seen in Figure 4.

5 PSO

In this section, we present how to use the controller generated primitives for local motion planning. This is achieved by selecting the best set point θ_p or the best end velocity θ_v with the PSO algorithm. The PSO is a gradient-free technique and is capable of finding the best motion directly with a cost map. Compared to previous methods such as [15], it does not require to process the cost map with Euclidean distance transform. The detailed algorithm can be found in Algorithm 2. Each particle's location P_{position} in the optimization space represents either a set point θ_p or a desired velocity θ_v. They are first randomly initialized in line 6–7. Each particle is also assigned a particle velocity P_{velocity} determine its future movement in the optimization space. During each iteration, we first update the particle's location with their corresponding particle velocity (line 13 –14). Then we evaluate the cost of the particles using the function $\nu(P_{\text{position}})$. For each particle, we also record its best location P_{p_best} through all iterations. The best $P_{p\_best})$ among all particle is denoted as P_{g_best}. The P_{p_best} and P_{g_best} are used to update the particle velocity in line 13. If the PSO is limited to a single iteration, it performs the same procedure as previous methods in [5, 7–9], which is to construct a local search tree and searches for the best motion.

Algorithm 2 PSO

```
 1: Input: ProblemSize, PopulationSize
 2: Output: $P_{g\_best}$
 3: Population $\leftarrow \varnothing$
 4: $P_{g\_best} \leftarrow \varnothing$
 5: for $i = 1$ to PopulationSize do
 6:     $P_{\text{velocity}} \leftarrow \text{RandomVelocity}()$
 7:     $P_{\text{position}} \leftarrow \text{RandomPosition}()$
 8:     $P_{p\_best} \leftarrow P_{\text{position}}$
 9:     if $\nu(P_{p\_best}) \leq \nu(P_{g\_best})$ then
10:         $P_{g\_best} \leftarrow P_{p\_best}$
11: while ¬StopCondition() do
12:     for $P$ ∈Population do
13:         $P_{\text{velocity}} \leftarrow \text{updateVelocity}(P_{\text{velocity}}, P_{g\_best}, P_{p\_best})$
14:         $P_{\text{position}} \leftarrow \text{updatePosition}(P_{\text{position}}, P_{\text{velocity}})$
15:         if $\nu(P_{\text{position}}) \leq \nu(P_{p\_best})$ then
16:             $P_{p\_best} \leftarrow P_{\text{position}}$
17:             if $\nu(P_{p\_best}) \leq \nu(P_{g\_best})$ then
18:                 $P_{g\_best} \leftarrow P_{p\_best}$
19: Return($P_{g\_best}$)
```

When evaluating a particle P with the function ν, we first construct the trajectory with the approximation NN $S_{p\text{NN}}$ or $S_{v\text{NN}}$ and evaluate it against a cost function $J_s(\mathbf{x}(t))$ To achieve the desired behavior of safe navigation, we need to design the cost function $J_s(\mathbf{x}(t))$ with the following consideration:

- Trajectory clearness: Given a cost map as in Figure 1, we could evaluate the trajectory's clearness by adding up the costs of the grids that the robot's footprint has covered. We denote the clearness value as J_{ob}.
- Trajectory aggressiveness: For a smooth trajectory, we penalize the integration of the square of the jerk over the trajectory J_u.
- Progress: We evaluate the progress of the trajectory by measuring its remaining distance J_g to the goal. The

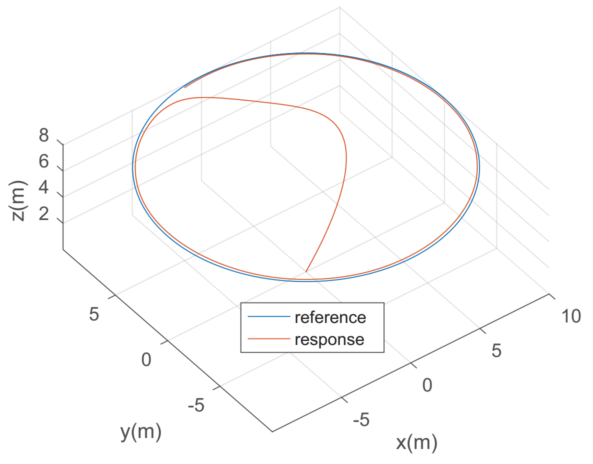

Fig. 5: Tracking a circling reference.

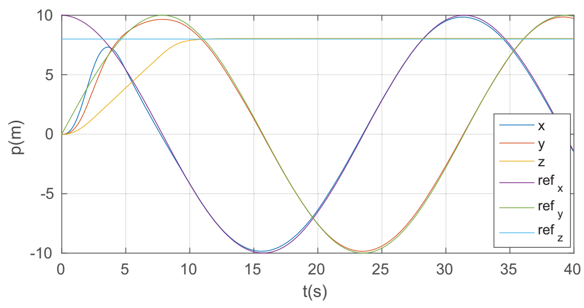

Fig. 6: Position response of tracking a circling reference

remaining distance can be obtained as heuristics from a higher level global planner.

The total cost function J_s is a weighted combination of J_{ob}, J_u and J_g. Finally, the PSO is executed repeatedly with a receding horizon. In each planning cycle, the optimization is solved based on the vehicle's current state and the updated environment map. Through the simulation, we demonstrate two possible applications with the proposed method. In Figure 5, we use the velocity controller generated primitives to track a circling reference. The vehicle is initially at the origin, but it is capable of navigating to the circle surface that is 8 meters above the ground while satisfying all limits on velocity, acceleration, and jerk. Its 3-axis position response is given in Figure 6. In Figure 7, the quadrotor is tasked to reach a pre-defined goal point in an indoor environment. The corresponding velocity reference is given in Figure 8. Due to the underlying control based motion primitives, the velocity is successfully capped between $\pm 2.5\,\mathrm{m/s}$.

6 Flight experiment

In this section, we present the real vehicle flight experiment. The system structure of our testing platform is given in The environment is sensed by a ZED stereo camera and an Rp-lidar. It is constructed as a grid map that is used by the PSO algorithm to evaluate the trajectories. A global planner such as the one in [5] is responsible for generating the progress function J_g. After the PSO algorithm successfully found the best set point θ_p or the best velocity target θ_v. They are sent to the corresponding controller designed by dynamic programming. Along with the current state of the vehicle, reference trajectory and nominal input can be constructed which are consumed by feedback and feed-forward controller.

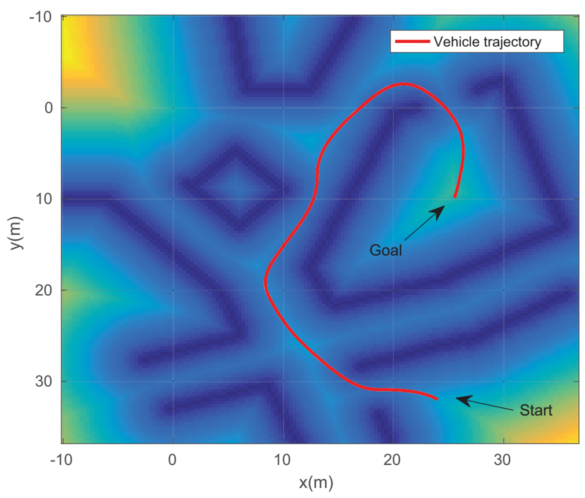

Fig. 7: Navigate in a indoor environment.

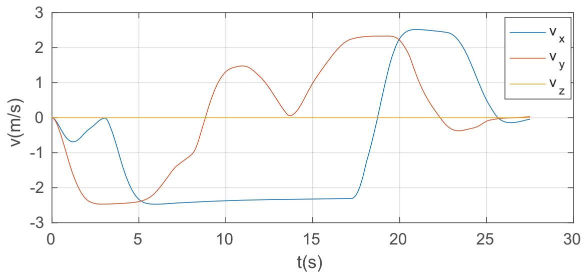

Fig. 8: Velocity response of navigating in a indoor environment.

In this experiment, we use motion primitives generated by the setpoint controller. The NN approximation is achieved with a $3 \times 64 \times 128 \times 128 \times 41$ rectifier (ReLU) network. It is capable of predicting the future trajectory for 20 seconds with a 2 Hz accuracy. ReLU layer is adopted since it can be easily implemented on a CPU only unit. In Table 6 we compare the time consumption between evaluating the NN and forward simulating for 20 seconds. Compared to directly generate the trajectory by forward simulation with the proposed controller, the NN is 4 times faster under the same CPU. With a small GPU that is currently equipped on our quadrotor, it is 20+ times faster. For the PSO, 15 particles are iterated for 10 times. The average time consumption for

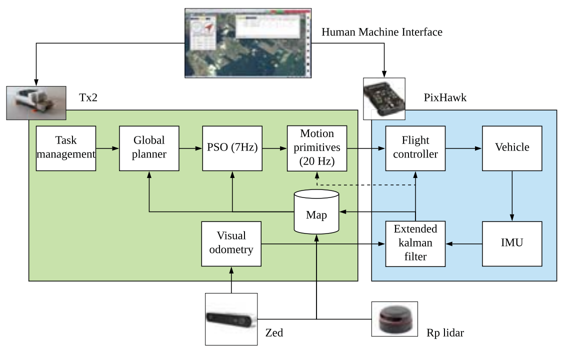

Fig. 9: System design of the experiment platform.

Table 1: Time consumption between NN and forward simulation

	I7	TX2 GPU
Forward simulation	70 μs	-
NN evaluation	15 μs	3 μs

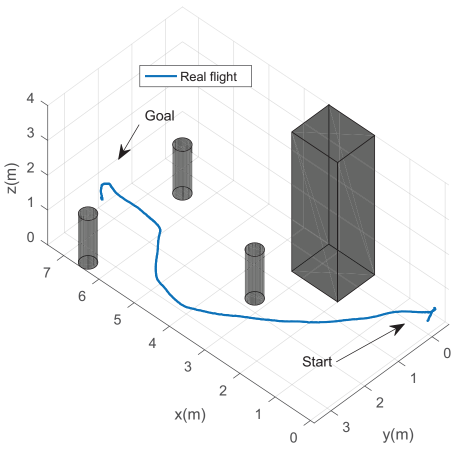

Fig. 10: Position response of tracking a circling reference

each planning cycle is only 14 ms. In Figure 10, the vehicle travels through the obstacles with only the onboard sensor and onboard computational power. Its velocity response is given in Figure 11.

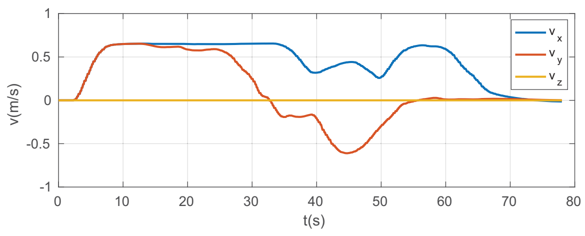

Fig. 11: Position response of tracking a circling reference

7 Conclusion

In this paper, we have presented a method to use controllers to generate motion primitives for the quadrotor. These motion primitives can be approximated by an NN to boost its online evaluation efficiency. In this manner, a wide range of motion primitives can be generated offline and used for online optimization. Due to the reduced dimension of the optimization problem, algorithms like the PSO is adopted. It is a gradient-free technique that can be applied directly on the underlying nonlinear and non-convex optimization problem. The planning is done in a receding horizon fashion to guide the vehicle to travel safely in an unknown and obstacle-strewn environment.

References

[1] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters*, vol. 2, pp. 1688–1695, July 2017.

[2] J. Chen and S. Shen, "Improving octree-based occupancy maps using environment sparsity with application to aerial robot navigation," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3656–3663, May 2017.

[3] S. Lai, K. Wang, H. Qin, J. Q. Cui, and B. M. Chen, "A robust online path planning approach in cluttered environments for micro rotorcraft drones," *Control Theory and Technology*, vol. 14, pp. 83–96, Feb 2016.

[4] T. Howard, M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Model-predictive motion planning: Several key developments for autonomous mobile robots," *IEEE Robotics Automation Magazine*, vol. 21, pp. 64–73, March 2014.

[5] P. Florence, J. Carter, and R. Tedrake, "Integrated perception and control at high speed : Evaluating collision avoidance maneuvers without maps," in *Int. Workshop on the Algorithmic Foundations of Robotics*, 2016.

[6] M. Hehn and R. DAndrea, "Real-time trajectory generation for quadrocopters," *IEEE Transactions on Robotics*, vol. 31, pp. 877–892, Aug 2015.

[7] B. T. Lopez and J. P. How, "Aggressive 3-d collision avoidance for high-speed navigation," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5759–5765, May 2017.

[8] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3480–3486, Nov 2013.

[9] X. Yang, K. Sreenath, and N. Michael, "A framework for efficient teleoperation via online adaptation," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5948–5953, May 2017.

[10] M. Lan, S. Lai, and B. M. Chen, "Towards the realtime sampling-based kinodynamic planning for quadcopters," in *2017 11th Asian Control Conference (ASCC)*, pp. 772–777, Dec 2017.

[11] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, pp. 2520–2525, May 2011.

[12] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for quadrotor flight," in *RSS Workshop on Resource-Efficient Integratiton of Perception, Control and Navigation for MAVs*, 2013.

[13] J. Chen, T. Liu, and S. Shen, "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1476–1483, May 2016.

[14] S. Lai, M. Lan, and B. M. Chen, "Optimal constrained trajectory generation for quadrotors through smoothing splines," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4743–4750, Oct 2018.

[15] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online uav replanning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5332–5339, Oct 2016.

[16] M. Hehn, R. Ritz, and R. D'Andrea, "Performance benchmarking of quadrotor systems using time-optimal control," *Autonomous Robots*, vol. 33, pp. 69–88, Aug 2012.

[17] R. Bellman, "The theory of dynamic programming," *Bull. Amer. Math. Soc.*, vol. 60, pp. 503–515, 11 1954.