Model Predictive Local Motion Planning With Boundary State Constrained Primitives

Shupeng Lai, Menglu Lan, and Ben M. Chen

***Abstract*—Motion primitives are frequently used to find valid local trajectories for mobile robots, especially in cases where fast replanning is required, but the onboard computational power is limited. In this letter, we present a practical framework for constructing motion primitives from boundary state constraints, and then using them for online planning. The primitives are offline constructed with either a boundary value problem solver or a controller. They are then approximated with a neural network for fast evaluation during online optimization. The references and nominal inputs are generated in a receding horizon fashion by solving a model predictive control problem in the continuous domain with either gradient-based or gradient-free techniques. The proposed approach is computationally efficient and has been tested on quadrotors in real flight experiments, including sensor-based navigation, flying through a complex three-dimensional environment, dynamic obstacle avoidance, and tracking moving references.**

***Index Terms*—Motion and path planning, collision avoidance, motion primitives.**

I. Introduction

MOTION planning for mobile robots can be typically decoupled into global planning and local planning phases. The global planning is responsible for finding the connectivity information of the environment with the simplified dynamics model and providing guidance, such as heuristics, to the lower-level local planner. Local planning is responsible for the real-time reaction to environmental changes with a detailed dynamics model. Due to environmental uncertainties, the planning process is typically repeated in a receding horizon fashion, similar to the one used in the model predictive control (MPC). In this letter, we focus on the local planning problem which is solved by searching in the parameterized space of the robot's inputs and producing a dynamically feasible and collision-free local trajectory according to a predefined cost function.

Manuscript received February 24, 2019; accepted June 26, 2019. Date of publication July 12, 2019; date of current version July 24, 2019. This letter was recommended for publication by Associate Editor L. Tapia and Editor N. Amato upon evaluation of the reviewers' comments. *(Corresponding author: Shupeng Lai.)*

S. Lai is with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 119077 (e-mail: elelais@nus.edu.sg).

M. Lan is with the Graduate School for Integrative Science & Engineering, National University of Singapore, Singapore 119077 (e-mail: lanmenglu@gmail.com).

B. M. Chen is with the Department of Mechanical and Automation Engineering, Chinese University of Hong Kong, Hong Kong, and also with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 119077 (e-mail: bmchen@cuhk.edu.hk).

This letter has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. This video shows a practical framework to construct motion primitives from boundary state constraints and further use them for online planning.

Digital Object Identifier 10.1109/LRA.2019.2928255

Motion primitives (MPs) are frequently used for local motion planning by spanning a local search tree to abstract a continuous state space [1]. The best local trajectory can then be found efficiently via tree searching techniques. According to [2], the process resembles a localized version of the sampling-based planning process [3]. To achieve a fast reaction to environmental changes, the local search tree can often be limited to a single layer [2], [4]–[7]. Compared to the traditional gradient-based optimization approaches, it neither requires the environmental gradient information [8] nor restricts the trajectory in convex subsets of the free space [6]. However, the sparsity of the search tree limits the resolution of the solution, restricting its quality and availability, especially in applications that require precise maneuvers such as walking over cinder blocks [9] or tracking a moving reference. The two most common approaches of generating MPs are 1) sampling in the vehicle's input space and then performing a forward simulation, and 2) sampling on the vehicles' boundary state constraints, and then generating the actual motion by solving a boundary value problem (BVP) [1]. With input sampling, though dynamic constraints are inherently satisfied by forward simulation, it is difficult to ensure a well-separated coverage in the search space because the state space response of the sampled input can vary as a non-linear function. Therefore, we adopt the latter approach and refer to the generated MPs as the boundary state constrained primitives (BSCPs) which have been successfully applied to various platforms, including unicycles [10], autonomous cars [11], [12], rotorcrafts [5], [13], fixed wings [2], [14] and field robots [15]. With the BSCPs, a long and feasible trajectory can usually be encoded in a few parameters, thus effectively reducing the dimension of the optimization problem and boosting computational efficiency. However, solving the BVP remains a non-trivial problem and could potentially be time-consuming.

In this letter, we adopt a neural network (NN) to approximate the BVP solutions (i.e. the BSCPs) and present an innovative framework combining BSCPs and MPC for real-time applications. The main contributions of the letter can be summarized as follows:

- We formulate the problem of finding the best BSCP as an optimization problem over the continuous domain. Compared with the traditional tree searching based method, our method generates a solution of better quality for tasks like tracking a moving reference.
- We propose two methods to solve the optimization problem. The first one is gradient-based, where the gradient is obtained through the backpropagation of the NN. The second one is a gradient-free method using the NN and the particle swarm optimization (PSO) to construct a dynamically evolving single layer tree that gradually converges to the optimal local motion. The PSO inherits the advantages

of the tree searching techniques while addressing the issue of the limited resolution of the solution.
- We implement the proposed approach on a real quadrotor which is capable of navigating in 3D complex environments, tracking moving references and avoiding dynamic obstacles.

In our framework, the NN is used to learn the BSCPs offline and approximate them during online optimization. Therefore, the BSCPs can be evaluated efficiently by avoiding the time consuming numerical optimization [16], while not compromising on motion constraints [13], [17] and optimization targets [5], [18]. Although a similar effect can be achieved with a look-up table, a densely constructed look-up table can be prohibitively large for high-dimensional systems, while a sparse one restricts the BSCPs' initial condition to certain selected states [19].

The rest of the letter is organized as follows. In Section II, the related works are reviewed. The overview of the proposed approach is given in Section III. Section IV discusses the generation of the BSCP and its corresponding NN with both quadrotor and unicycle models. In Section V, we discuss how to solve the resulting optimization problem with two different methods. The real flight experiments with quadrotors are given in Section VI. Finally, concluding remarks are provided in Section VII. All values presented are in SI units unless otherwise stated.

II. Related Works

The successful application of MPs in local motion planning can be traced back to earlier works such as [10], in which the author uses the current state of the vehicle and the desired heading to encode the MPs. A similar approach can be found more recently in [5], where each MP is generated from the current state to the desired end velocity by solving a time-optimal BVP in closed-form. The best motion is then selected from a collection of uniformly sampled MPs. However, the trajectory could be unnecessarily aggressive due to the time-optimal formulation. A similar method that accepts the desired end position can be found in [18]. In order to achieve a smooth trajectory, works in [13], [17] use BVP solvers that minimize the integration of the square of the trajectory's high order derivatives. For efficiency, the state constraints are ignored in the BVP, and an extra checking process is needed to select the valid motions. In applications such as on-road driving, biased sampling can be adopted to reduce the search space. In [12], the MPs are regulated onto a carefully chosen set of terminal states aligned with the reference path and inside the lane. For applications where such biased sampling is not possible, a fixed or randomly generated set of MPs is usually used to construct the search tree [1], [2], [4], [5]. In [7], a densely sampled motion primitive library is adopted. During the online execution, the library that is sufficiently close to the initial condition is chosen to provide the motions. To improve the resolution of the solution, [11] selects the best matching motions from a look-up table and uses them to construct the initial guess for online optimization. However, the size of the look-up table grows exponentially with its dimension. Similar approaches can also be found in [6].

On the other hand, the encoding of complex motions requires an extra set of latent parameters, such as the dynamic movement primitive (DMP) [20] which is generated with a predefined linear system and inputs that are parameterized with Gaussian basis functions. With the increased number of parameters, it becomes exponentially expensive to find the best motion by constructing a search tree. Gradient-based methods, like the sequential quadratic programming, are then adopted [21]. In [6], [22], a quadrotor and a model car have been successfully guided through obstacles by performing such optimization at each control cycle. However, they require the obstacles to be considered as well-defined geometric shapes. In sensor-based navigation, constructing these shapes directly from sensor information remains a non-trivial task for a general environment. In [8], obstacles are modeled with a cost map instead of geometric shapes. A gradient-based method is then applied to the resulting non-convex problem. Due to the nature of such problem formulations, failure cases are often observed and require random restarts of the optimization process. Recently, the use of NNs for local motion planning has been gaining popularity. It can either be used to model the complex system dynamics and make predictions on future states given an input sequence [23], [24], or substitute the local planner entirely through reinforcement learning [25]. Complex and long-range motions can then be generated by combining the local motion with a sampling-based global planner [25], [26].

III. Preliminary

In this section, the proposed framework is presented with a general mobile robot model.

A. Problem Formulation

Let $\mathbf{x} \in \mathcal{X}$ denote the robot's state and $\mathbf{u} \in \mathcal{U}$ represent its input. Assume it has the dynamics

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}), \tag{1}$$

with invariant constraints

$$h(\mathbf{x}, \mathbf{u}) = 0,\ \tilde{h}(\mathbf{x}, \mathbf{u}) \leq 0 \tag{2}$$

which are the collection of state and input constraints that are invariant to the environment. In contrast, the collision free constraints

$$\mathbf{x} \notin \mathcal{O} \tag{3}$$

are environment-dependent with $\mathcal{O}$ being the obstacle set. Assume the initial condition at time instance t_0 is

$$\mathbf{x}(t_0) = \mathbf{x}_0. \tag{4}$$

Let $\mathbf{u}(t), \mathbf{x}(t), t \in [t_0, t_0 + T]$ represent the input and the state trajectories respectively with T being the planning horizon. The local motion planning is then formulated as an MPC problem which minimizes

$$J = \zeta(\mathbf{x}(t_0 + T)) + \int_{t_0}^{t_0+T} L(\mathbf{x}(t), \mathbf{u}(t))dt \tag{5}$$

subject to constraints in Equations (1)–(4). The functions ζ, L are user-defined terminal and running costs. The optimization is done every τ seconds with the updated robot and environmental information. Instead of solving the optimization problem directly, we use BSCPs to reduce its dimension. The BSCP is the solution of the BVP:

$$\min_{\mathbf{u}(t), \mathbf{x}(t), t_f} \quad G(\mathbf{x}(t), \mathbf{u}(t), t_f) \tag{6}$$

subject to constraints in Equations (1), (2), (4) and an additional end state constraint

$$g(\mathbf{x}(t_f), \boldsymbol{\theta}) = 0 \tag{7}$$

where G is a user-defined cost function, t_f is the final time and $\boldsymbol{\theta}$ denotes parameters of g. If $\mathbf{x}(t_f)$ happens to be on the trim manifold [19], it is possible to find a controller that regulates the system to

$$\|g(\mathbf{x}(t_f), \boldsymbol{\theta})\| < \epsilon, \forall t \geq t_f, \tag{8}$$

with ϵ being a small positive number. We formulate the BVP such that its solutions $\hat{\mathbf{u}}(t), \hat{\mathbf{x}}(t), \hat{t}_f$ are uniquely dependent on $\boldsymbol{\theta}$ and $\mathbf{x}_0$, which can be represented by the mapping:

$$\mathcal{S}: \langle \mathbf{x}_0, \boldsymbol{\theta} \rangle \rightarrow \left\langle \hat{\mathbf{u}}(t), \hat{\mathbf{x}}(t), \hat{t}_f \right\rangle. \tag{9}$$

If x_0 is fixed through the measurement process, $\hat{\mathbf{u}}(t), \hat{\mathbf{x}}(t)$ are then determined by $\boldsymbol{\theta}$ only. The problem of minimizing Equation (5) can then be rewritten as

$$\min_{\boldsymbol{\theta}} J = \int_{t_0}^{t_0+T} \bar{L}(\boldsymbol{\theta}) dt + \bar{C}(\boldsymbol{\theta}) \tag{10}$$

with $\bar{L}, \bar{C}$ being the running and the terminal costs respectively.

B. Method Overview

The optimization problem in Equation (10) is usually solved sparsely by sampling a set of discrete $\boldsymbol{\theta}$, generating a new trajectory for each $\boldsymbol{\theta}$ sample, evaluating all trajectories and finding the best trajectory among them [2], [4], [5], [13], [27]. In this manner, the optimization space is reduced to sparse discrete samples, and the solution could be highly suboptimal therefore unsuitable for tasks requiring precise maneuvers. Another challenge is to evaluate the mapping $\mathcal{S}$ efficiently for real-time applications. Although efficient closed-form solutions are available for some vehicles, they often ignore certain state constraints [13], [17] or require specific cost functions [5], [18]. On the other hand, numerical methods are usually much slower, therefore limited to a small number of discrete samples [16].

In the proposed method, we approximate $\mathcal{S}$ with a neural network $\mathcal{S}_{\mathrm{NN}}$. With modern hardware, the propagation of $\mathcal{S}_{\mathrm{NN}}$ can be evaluated very efficiently, which allows the PSO to be executed in real-time. The gradient $\frac{\mathrm{d}J}{\mathrm{d}\boldsymbol{\theta}}$ can also be found through backpropagation, so that J can be minimized with gradient-based methods. Although look-up tables can be used for a similar effect, it could become prohibitively large for high dimensional systems. In [26], an NN is trained to approximate the solution of the BVP of a nine degrees-of-freedom (DOF) system. The corresponding look-up table would have entries on the level of 10^{18}, which is challenging to construct and store.

Our approach consists of an offline training stage and an online MPC stage.

- Offline: Randomly sample the initial state $\mathbf{x}_0$ and the end state constraint $\boldsymbol{\theta}$. Generate trajectories for these sampled pairs using adequate BVP solvers or controllers. Finally, train an NN with $\mathbf{x}_0, \boldsymbol{\theta}$ as inputs and the desired trajectory information, such as the discrete time samples of $\hat{\mathbf{x}}(t)$, as outputs.
- Online: Each online MPC cycle further consists of three phases.
  1) Update the current state of the vehicle $\mathbf{x}_0$ and the environmental information.
  2) With $\mathbf{x}_0$ fixed, we search for $\boldsymbol{\theta}^*$ that minimizes J (see Equation (10)). Given an arbitrary $\boldsymbol{\theta}$, the NN can be used to approximate the trajectory defined by $\mathbf{x}_0$ and $\boldsymbol{\theta}$, or to find the derivative $\frac{\mathrm{d}J}{\mathrm{d}\boldsymbol{\theta}}$. Then, gradient-free (see Section V-B) or gradient-based (see Section V-A) methods can be used to find $\boldsymbol{\theta}^*$ in the continuous domain.
  3) With $\boldsymbol{\theta}^*$, the reference trajectory is reconstructed with the original BVP solver or the controller used in the offline stage. The reference is rechecked against the constraints. If the checking fails, we try the trajectory from the previous MPC cycle. In cases where the constraints are still violated, an emergency stop command is issued.

In this manner, the BVPs are solved at most 2 times (when constructing the final reference) in the MPC cycle. Furthermore, because the final reference is obtained through the original BVP solver or controller, it is kept free from the numerical noise of the NN. The checking on the final reference guarantees the soundness of the method.

IV. Construction of BSCP

In this section, we discuss the construction of the BSCP and its corresponding NN approximation through a quadrotor model (Section IV-A) and a second order unicycle model (Section IV-B).

A. Quadrotor

Following [18], the quadrotor is modeled as a 9 DOF system with a triple integrator on each of its x, y, z axis. In our previous work [26], we construct the BSCP by solving a 9 DOF BVP numerically. Here, we show how to construct the BSCP from a controller designed through dynamic programming (DP). On each axis, the 3 DOF discrete-time dynamics of a triple integrator can be expressed as

$$\mathbf{x}_q[n+1] = A_q \mathbf{x}_q[n] + b_q u_q[n] \tag{11}$$

where

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

with state $\mathbf{x}_q = [p, v, a]^\intercal$ being its position, velocity, acceleration and input $u_q = j$ being its jerk. The invariant constraints are:

$$v \in [v_{\min}, v_{\max}], a \in [a_{\min}, a_{\max}], j \in [j_{\min}, j_{\max}] \tag{12}$$

where the limits might be different for each individual axis. The controller regulates the triple integrator from an arbitrary initial state to a desired set point $\mathbf{x}_{q_{\mathrm{d}}} = [\theta_q, 0, 0]^\intercal$. The end state constraint is

$$g_q(\mathbf{x}_q, \theta_q) = \mathbf{x}_q - \mathbf{x}_{q_{\mathrm{d}}} = [p - \theta_q, v, a]^\intercal = 0 \tag{13}$$

We define the relative state $\bar{\mathbf{x}}_q$ and relative position $\bar{p}$ as

$$\bar{\mathbf{x}}_q = \mathbf{x}_q - \mathbf{x}_{q_{\mathrm{d}}}, \bar{p} = p - \theta_q. \tag{14}$$

Substituting Equation (14) into (11) gives

$$\bar{\mathbf{x}}_q[n+1] = A_q \bar{\mathbf{x}}_q[n] + b_q u_q[n]. \tag{15}$$

TABLE I
DETAILS ON THE NEURAL NETWORKS

	NN structure	Training set size	Testing set size	Batch size	Epoch	Average MSE	Maximum MSE
Quadrotor horizontal	64-128-128-41 MLP	800,000	200,000	20	12	4.72×10^{-4}	0.071
Quadrotor vertical	64-128-128-41 MLP	800,000	200,000	10	15	1.2×10^{-3}	0.088
2nd order unicycle	80-256-128-81 MLP	860,000	130,000	10	20	3.4×10^{-3}	0.094

* MSE: Mean Squared Error. MLP: Multi-Layer Perceptron.

The target now is to regulate $\bar{\mathbf{x}}_q$ to zero. We use the value function $V(\bar{\mathbf{x}}_q)$ to denote the remaining cost from $\bar{\mathbf{x}}_q$ to the origin, and it follows

$$V(\bar{\mathbf{x}}_q[n]) = C(\bar{\mathbf{x}}_q[n], u_q[n]) + V(\bar{\mathbf{x}}_q[n+1]). \tag{16}$$

where $C(\bar{\mathbf{x}}_q, u_q)$ is the instantaneous cost. In our implementation, it is responsible for regulating the system to the origin while satisfying Equation (12) as soft constraints. Therefore, it is given as

$$C(\bar{\mathbf{x}}_q, u_q) = \bar{\mathbf{x}}_q^\intercal H \bar{\mathbf{x}}_q + r u_q^2 + H_p(\bar{\mathbf{x}}_q, u_q) \tag{17}$$

where

$$H_p(\bar{\mathbf{x}}_q, u_q) = w_v \eta^2(v, v_{\min}, v_{\max}) + w_a \eta^2(a, a_{\min}, a_{\max}) + w_j \eta^2(j, j_{\min}, j_{\max})$$

with $\eta(k, k_1, k_2) = \max(k_1 - k, 0) + \max(k - k_2, 0)$, $k, k_1, k_2 \in \mathcal{R}$ and w_v, w_a, w_j being weighting factors. Here, $H_p(\bar{\mathbf{x}}_q)$ acts as a soft constraint to penalize the violation of Equation (12). As a necessary condition, the minimum value function for each state is achieved when

$$V^*(\bar{\mathbf{x}}_q[n]) = \min_{u_q[n]} C(\bar{\mathbf{x}}_q[n], u_q[n]) + V^*(\bar{\mathbf{x}}_q[n+1]) \tag{18}$$

is satisfied for all $\bar{\mathbf{x}}_q$ of interest. Equation (18) is called the Bellman equation and can be solved by discretizing the state/input space and performing value iteration. The resulting optimal policy $\pi(\bar{\mathbf{x}}_q)$ is obtained as a lookup table. With the policy, a unique state trajectory can be obtained by regulating the system (given in Equation (11)) from a given initial state to the origin. The process is represented through mapping

$$\bar{\mathrm{x}}_q(t) = \bar{S}(\bar{\mathbf{x}}_{q0}),\ \mu_q = \bar{R}(\bar{\mathbf{x}}_{q0}) \tag{19}$$

where $\bar{\mathrm{x}}_q(t)$ is the relative state trajectory, $\mu_q = \int u_q^2(t)dt$ is the integration of the square of the input and $\bar{\mathbf{x}}_{q0}$ is the relative initial state. From Equation (14), there is $\bar{\mathbf{x}}_{q0} = \mathbf{x}_{q0} - \mathbf{x}_{q_\mathrm{d}}$ with $\mathbf{x}_{q0}$ being the real initial state. The real state trajectory is recovered as $\mathbf{x}_q(t) = \bar{\mathbf{x}}_q(t) + \mathbf{x}_{q_\mathrm{d}}$.

In order to limit the size of the NN so that it can be deployed on devices without GPU, we learn the mapping from $\bar{\mathbf{x}}_{q0}$ to the relative position $\bar{p}(t)$ instead of the full relative state $\bar{\mathrm{x}}_q(t)$. By discretizing $\bar{p}(t)$ at 2 Hz for 20 seconds into the future and cascading the result with μ_q, the output of the NN is a 41×1 vector, whereas its input is a 3×1 vector (the size of $\bar{\mathbf{x}}_{q0}$). The training data is collected by randomly sampling $\bar{\mathbf{x}}_{q0}$ and generating trajectories through forward simulation with the optimal policy $\pi(\bar{\mathbf{x}}_q)$. The sampling volume shall be large enough such that the generated trajectories cover the state limits in Equation (12). The NN is trained using PyTorch with the Adam optimizer and more of its details can be found in Table I. Although a more complex NN can be used to further reduce the training error, the four-layer rectifier (ReLU) multi-layer perceptron (MLP) is adopted for its efficiency on CPU-only platforms. In this letter, no emergency stop command is issued through all simulations and experiments with the trained NN.

B. Second Order Unicycle

By reusing some of the symbols in Section IV-A, the model of the second order unicycle can be expressed as

$$\begin{aligned} \dot{x} &= v\cos(\phi), \quad \dot{\phi} = \omega \\ \dot{y} &= v\sin(\phi), \quad \dot{\omega} = \beta \\ \dot{v} &= a \end{aligned} \tag{20}$$

where $\mathbf{x}_c = [x, y, v, \phi, \omega]^\intercal$ is the state of the vehicle. Here, x, y are its positions on the 2D plane, v, ϕ, ω are its speed, heading angle and angular speed, the input $\mathbf{u}_c = [a, \beta]$ includes the translational and angular acceleration. The invariant constraints are:

$$\begin{aligned} v &\in [v_{\min}, v_{\max}], \quad a \in [a_{\min}, a_{\max}] \\ \omega &\in [\omega_{\min}, \omega_{\max}], \quad \beta \in [\beta_{\min}, \beta_{\max}] \end{aligned} \tag{21}$$

The end state constraints are enforced on the heading and the velocity as:

$$g_c(\mathbf{x}_c, \boldsymbol{\theta}_c) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{x}_c - \underbrace{\begin{bmatrix} \theta_v \\ \theta_\phi \end{bmatrix}}_{\boldsymbol{\theta}_c} = 0, \tag{22}$$

with θ_v being the end velocity and θ_ϕ being the end heading. To solve the BVP, the velocity v is varied linearly with $a = \{a_{\min}, a_{\max}, 0\}$ towards θ_v. On the other hand, the heading trajectory is solved as a non-linear optimization problem using the time-bisection method presented in [26] for a smoother response. With the velocity and the heading generated, the overall system trajectory is then constructed through forward simulation. Other trajectory generation methods can also be used as long as the solved trajectory $\mathbf{x}_c(t)$ is uniquely dependent on the initial condition $\mathbf{x}_{c0}$ and the end state constraint $\boldsymbol{\theta}_c$. The mapping from $\mathbf{x}_{c0}, \boldsymbol{\theta}_c$ to the trajectory is denoted as

$$\mathbf{x}_c(t) = S_c(\mathbf{x}_{c0}, \boldsymbol{\theta}_c) \tag{23}$$

Similar to Section IV-A, the mapping can be learned with a neural network $S_{c\mathrm{NN}}$. Since the trajectory can be easily translated and rotated, we assume it always starts from $x = 0$, $y = 0$, $\phi = 0$. The NN has an input size of 4×1 including the desired speed, the desired heading, and the remaining initial states ω_0, v_0. By sampling the resulting position trajectory at 10 Hz for the future 4 seconds and cascading the result with the total energy consumption, the output of the NN is an 81×1 vector. More details of the network can be seen in Table I. The NN can be evaluated in 50 μs, while solving the BVP numerically takes 107 ms, both computed in the Matlab with an I7 CPU.

V. Model Predictive Motion Planning With BSCP

In this Section, the BSCP is applied to solve the MPC problem in Equation (5) for local motion planning applications. Unlike previous methods, the optimization problem is solved in the continuous domain in this work. In Section V-A, the gradient-based method is adopted to find the optimal motion. The cost gradient is obtained through backpropagation of the NN. In Section V-B, the PSO is adopted for solving the optimization problem. The commonly used single layer local search tree is, in fact, a PSO with only one iteration.

A. Gradient Based Method

Gradient-based methods are widely used in solving optimization problems. Given a cost function $J(\mathbf{x}(t), \mathbf{u}(t))$ and a fixed initial state $\mathbf{x}_0$, the best BSCP can be found by descending $\boldsymbol{\theta}$ in the direction of $\frac{\mathrm{d}J(\mathbf{x}(t),\mathbf{u}(t))}{\mathrm{d}\boldsymbol{\theta}}$, where $\boldsymbol{\theta}$ is the parameter of the end state constraints defined in Equation (7). With the chain rule, there is

$$\frac{\mathrm{d}J(\mathbf{x}(t),\mathbf{u}(t))}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial J}{\partial \mathbf{x}(t)}\frac{\partial \mathbf{x}(t)}{\partial \boldsymbol{\theta}} + \frac{\partial J}{\partial \mathbf{u}(t)}\frac{\partial \mathbf{u}(t)}{\partial \boldsymbol{\theta}}. \tag{24}$$

Usually, the $\frac{\partial J}{\partial \mathbf{x}(t)}$ and $\frac{\partial J}{\partial \mathbf{u}(t)}$ can be evaluated analytically as long as the cost function is smooth and directly defined on $\mathbf{x}(t)$ and $\mathbf{u}(t)$. On the other hand, the analytical evaluation of the $\frac{\partial \mathbf{x}(t)}{\partial \boldsymbol{\theta}}$ and $\frac{\partial \mathbf{u}(t)}{\partial \boldsymbol{\theta}}$ requires a closed-form solution to the BVP problem. For BSCP generated with numerical solvers and controllers, the gradient needs to be found through numerical differentiation. It is time consuming as solving the BVP or performing forward simulation could both be expensive. With the NN approximated mapping, $\frac{\partial \mathbf{x}(t)}{\partial \boldsymbol{\theta}}$ and $\frac{\partial \mathbf{u}(t)}{\partial \boldsymbol{\theta}}$ can be efficiently evaluated through backpropagation following the work in [23].

We illustrate the process of finding the $\frac{\mathrm{d}J}{\mathrm{d}\boldsymbol{\theta}}$ using the quadrotor's model. We first consider a flying-through problem where the vehicle is tasked to pass a waypoint at a specific time t_s. It can be achieved by minimizing the following cost function

$$J = \frac{1}{2}(p(t_s) - p_{\mathrm{w}})^{\intercal}(p(t_s) - p_{\mathrm{w}}) \tag{25}$$

where $p(t_s)$ is the position of the quadrotor at time t_s and p_{w} is the position of the waypoint. Combining Equation (14), there is

$$J = \frac{1}{2}(\bar{p}(t_s) + \theta_q - p_w)^{\intercal}(\bar{p}(t_s) + \theta_q - p_w). \tag{26}$$

Taking the derivative to θ_q gives

$$\frac{\mathrm{d}J}{\mathrm{d}\theta_q} = (\bar{p}(t_s) + \theta_q - p_w)\left(\frac{\mathrm{d}\bar{p}(t_s)}{\mathrm{d}\theta_q} + 1\right). \tag{27}$$

With the mapping in Equation (19) and its NN approximation $\bar{S}_{\mathrm{NN}}$, there are

$$\bar{p}(t) = \bar{S}_{\mathrm{NN}}(\bar{\mathbf{x}}_{q0}) \tag{28}$$

and

$$\frac{\mathrm{d}\bar{p}(t)}{\mathrm{d}\theta_q} = \frac{\mathrm{d}\bar{S}_{\mathrm{NN}}}{\mathrm{d}\bar{\mathbf{x}}_{q0}}\frac{\mathrm{d}\bar{\mathbf{x}}_{q0}}{\mathrm{d}\theta_q}. \tag{29}$$

Here, the $\frac{\mathrm{d}\bar{S}_{\mathrm{NN}}}{\mathrm{d}\bar{\mathbf{x}}_{q0}}$ can be evaluated through the backpropagation of the $\bar{S}_{\mathrm{NN}}$ [23]. Moreover, there is $\bar{\mathbf{x}}_{q0} = \mathbf{x}_{q0} - [\theta_q, 0, 0]^{\intercal}$ with $\mathbf{x}_{q0}$ being a constant. Therefore, $\frac{\mathrm{d}\bar{\mathbf{x}}_{q0}}{\mathrm{d}\theta_q}$ can be calculated analytically. Substituting Equation (29) into (27), the gradient $\frac{\mathrm{d}J}{\mathrm{d}\theta_q}$ is acquired, and methods like [28] can be used to minimize the cost J. In Figure 1, we use the gradient descent method to find a trajectory that guides the quadrotor on a 2D plane to pass through a waypoint at $p_w = [-1, 5]$ at $t_s = 3$. The initial states are $p_0 = [0, 0]$, $v_0 = [2, 1]$ and $a_0 = [-1, -1]$. The same optimization procedure can also be used for the MPC problem in Equation (6). For example, given a reference signal $p_{\mathrm{ref}}(t)$, trajectory tracking can be achieved by minimizing the cost function:

$$J = \int_{t_0}^{t_0+T} (p(t) - p_{\mathrm{ref}}(t))^{\intercal}(p(t) - p_{\mathrm{ref}}(t))dt \tag{30}$$

in the MPC's receding horizon fashion. For the ease of visualization, we track a reference signal on a single axis in Figure 2. The reference signal is a combination of step, cosine and ramp signals. It is not always smooth and dynamically feasible to the quadrotor. However, thanks to the underlying MPs, the invariant constraints are always satisfied with an average tracking error of 0.64.

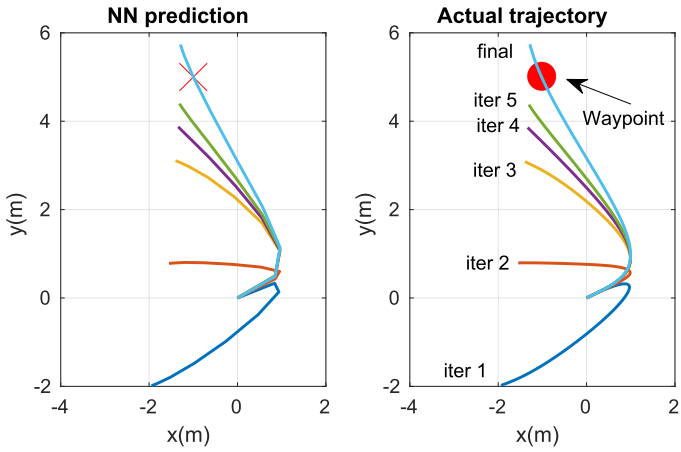


Fig. 1. Predicted vs actual trajectory of the quadrotor in gradient descent.

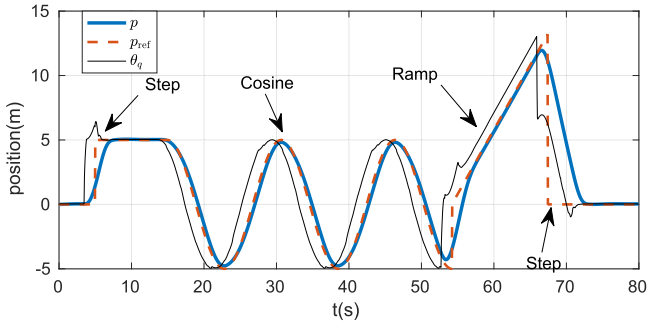


Fig. 2. Single axis trajectory tracking with the quadrotor.

B. Particle Swarm Optimization

In this section, we combine the BSCP with the PSO for local motion planning. Methods based on single layer search trees are a special case of the PSO algorithm, and they can be easily adapted into the PSO scheme for optimization in the continuous domain. PSO is a gradient-free technique, and it is capable of

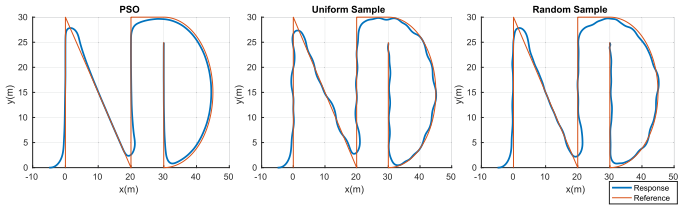


Fig. 3. Comparison of the tracking performance between using PSO and the uniform sampling applied on a second order unicycle.

Algorithm 1: Particle Swarm Optimization.

Input: $\mathbf{x}_{\text{ini}}$, $\mathcal{M}$(map)
Output: $\boldsymbol{\theta}^*$ (best end state constraint)

1: $\boldsymbol{\Theta} \leftarrow$ Particle_Initialization();
2: $c_i^* \leftarrow \infty$, $\theta_i^* \leftarrow \theta_i$, $\delta_i \leftarrow$ rand, $\forall i \in [1, \text{size}(\boldsymbol{\Theta})]$
3: **for** $m = 1$ to MAX_ITERS **do**
4: **for** each $\boldsymbol{\theta}_i \in \boldsymbol{\Theta}$ **do**
5: $[\mathrm{x}(t), \mathrm{u}(t)] = \mathcal{S}_{\text{NN}}(\mathbf{x}_{\text{ini}}, \boldsymbol{\theta}_i)$
6: $c_i = J(\mathrm{x}(t), \mathrm{u}(t), \mathcal{M})$
7: **if** $c_i < c_i^*$ **then**
8: $c_i^* = c_i$
9: $\boldsymbol{\theta}_i^* = \boldsymbol{\theta}_i$
10: $i^* = \underset{i}{\text{argmin}}(c_i^*)$
11: $\boldsymbol{\theta}^* = \boldsymbol{\theta}_{i^*}$
12: **for** each $\boldsymbol{\theta}_i \in \boldsymbol{\Theta}$ **do**
13: $\boldsymbol{\delta}_i = \boldsymbol{\delta}_i + k_1 \cdot \text{rand} \cdot (\boldsymbol{\theta}_i^* - \boldsymbol{\theta}_i) + k_2 \cdot \text{rand} \cdot (\boldsymbol{\theta}^* - \boldsymbol{\theta}_i)$
14: $\boldsymbol{\theta}_i = \boldsymbol{\theta}_i + \boldsymbol{\delta}_i$

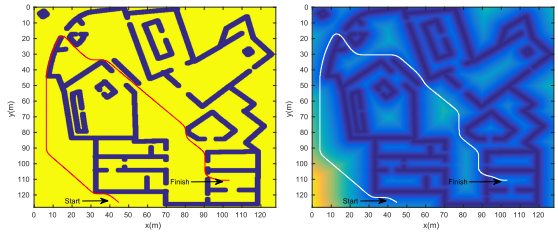
finding the solution even the cost function is non-continuous. The PSO process is given in Algorithm 1. First, a finite set of particles $\boldsymbol{\Theta}$ is initialized (line 1). Each particle represents an end state constraint. For the ith particle $\boldsymbol{\theta}_i$, $\boldsymbol{\theta}_i^*$ denotes its best value among all previous iterations with c_i^* being the corresponding cost value. The δ_i is its velocity in the parameter space. They are initialized in line 2 either randomly or with a predefined pattern. Within each iteration, we evaluate each particle with a cost function J (line 5–6), and update c_i^*, θ_i^* (line 7–9). During the evaluation process, the state trajectory is obtained with the NN mapping $\mathcal{S}_{\text{NN}}$, and evaluated against J with current environment map $\mathcal{M}$. Then the global best target $\boldsymbol{\theta}^*$ is found in line 10–11. Finally, the particle velocity and its location are updated in line 12–14. It is worth noting if the PSO is limited to one iteration, it becomes a local search tree based method. In line 1–9, a single layer search tree is constructed. In line 10–11, the best trajectory is selected from the tree. With multiple iterations, the search tree is gradually modified and converged to the optimal value.

In Figure 3, a second order unicycle presented in Section IV-B is tasked to track a moving reference. Its invariant constraints are $v \in [-0.5, 4], a \in [-2, 2], \omega \in [-1, 1]$ and $\beta \in [-2, 2]$. The cost function is

$$J = \int_{t=t_0}^{t=t_0+T} Q_c(\mathbf{x}_c(t)) + w_r R_c(\boldsymbol{\theta}_c) dt$$
$$Q_c(\mathbf{x}_c(t)) = (\mathbf{x}_c(t) - \mathbf{x}_{\text{cref}}(t))^\intercal (\mathbf{x}_c(t) - \mathbf{x}_{\text{cref}}(t))$$
$$R_c(\boldsymbol{\theta}_c) = \left\| \boldsymbol{\theta}_c - \boldsymbol{\theta}_{c,\text{previous}} \right\|_2 . \tag{31}$$

TABLE II
PERFORMANCE OF PSO AND UNIFORM SAMPLING

	η_ϕ	η_v	Average err
PSO	6.54	23.17	0.67
Uniform Sample	14.75	28.11	0.74
Random Sample	11.49	64.55	0.75



(a) Cost map generated through dilation of the occupied pixel. (b) Cost map generated through EDT.

Fig. 4. Motion planning in the indoor environment.

The tracking is done using MPC with prediction horizon $T = 4s$ and the re-planning frequency of 2 Hz. In each MPC iteration, we compare the PSO's performance with the uniform sampling strategy [5], [7], and the random sampling strategy [2]. In the uniform/random sampling cases, 100 uniformly/randomly sampled end state constraints are drawn from $\theta_v \in [-0.5, 4]$ and $\theta_\phi \in [-0.9\pi + \phi_0, 0.9\pi + \phi_0]$ with θ_v, θ_ϕ defined in Equation (22) and ϕ_0 being the initial heading. The PSO has 10 randomly initialized particles and is iterated for 10 times. Here, we use the unicycle BSCP presented in Section IV-B. All methods have the same computational burden of evaluating 100 trajectories. The result can be seen in Figure 3. Compared to the uniform and random sampling, PSO gives a much smoother trajectory by optimizing in the continuous domain. The smoothness is measured by calculating $\eta_\phi = \frac{\int \omega^2(t)dt}{T_{\text{total}}}$, $\eta_v = \frac{\int a^2(t)dt}{T_{\text{total}}}$ where T_{total} is the total time of the trajectory. The comparison of the smoothness and the average tracking error are given in Table II. The values are acquired as average of 10 consecutive runs. Besides tracking a moving reference, the PSO method is also suitable for guiding the vehicle in an obstacle-strewn environment. Here, we use it to guide a quadrotor in a simulated indoor environment. Following Equation (11), where $\mathbf{x}_q$ is the vehicle's state on a single axis, we use $\mathbf{X}_q = [\mathbf{x}_{q_x}^\intercal, \mathbf{x}_{q_y}^\intercal, \mathbf{x}_{q_z}^\intercal]^\intercal$ to denote the vehicle's state in the 3D space. The cost function for indoor navigation is then

$$J = \int_{t=t_0}^{t=t_0+T} H_q(\mathbf{X}_q(t)) + O_q(\mathbf{X}_q(t)) dt \tag{32}$$

The local motion is encoded with the BSCP and the MPC is executed at 4 Hz with a prediction horizon of 10 seconds. Here, the heuristic H_q is the remaining distance to the target and is estimated by a global planner. The ability of the vehicle to handle non-convex obstacles depends on the accuracy of H_q. Following [4], we use Dijkstra's algorithm to calculate H_q for each grid in the workspace. In practice, efficient methods like [3], [29] can be used to estimate the heuristics partially. The incomplete heuristics can then be updated with hierarchical planning techniques [30]. The details on the integration of the global and the local planner are out of the scope of this letter. The obstacle cost O_q is defined upon a cost map (see Figure 4(a) and

TABLE III
TIME CONSUMPTION BETWEEN NN AND FORWARD SIMULATION

	C++ with I7	PyTorch with TX2 GPU
Forward simulation	70 μs	-
NN evaluation	15 μs	3 μs

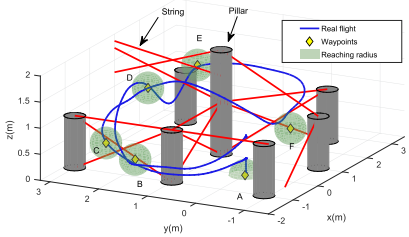
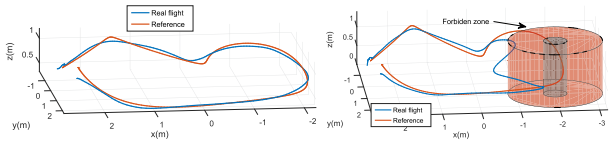
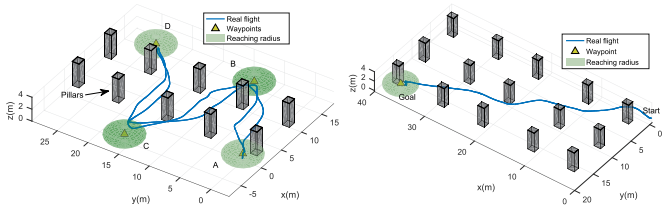


Fig. 5. The real flight experiment of flying through complex environments to reach a series of targets. The location of the next target is not known to the quadrotor until it reaches the current target. (The obstacle plots are for illustration purpose only, the planning is based on an EDT map.)



(a) Without obstacle (b) With obstacle

Fig. 6. Real flight experiments of trajectory following. In the case that an obstacle blocks the desired trajectory, the quadrotor could avoid it by deviating from the path and converge back to it afterward.



(a) Zig-zag (b) Straight flight

Fig. 7. Real flight experiments of sensor based navigation in an underground car park. (The obstacle pillars are for illustration purpose only, the environment is priorly unknown. The cost map is constructed with a ZED camera and an RP-lidar.)

4(b)). Each grid in the map is given a cost value depending on its distance γ to the nearest obstacle. The obstacle cost $O_q(\mathbf{X}_q(t))$ is then evaluated as the sum of the cost of the grids that are occupied by the robot while at $\mathbf{X}_q(t)$. One possible way to assign the grid cost is through the dilation. Specifically, a grid with $\gamma < r_s$ can be identified and assigned with a high-cost value, where r_s is a design factor. However, the resulting cost function jumps at the edge of the dilated obstacles. To get a smooth cost map, techniques such as Euclidean distance transform (EDT) are often used, but incurring a higher computational burden. Unlike the gradient-based method [8], a smooth cost map is not a necessity for the PSO algorithm. Figure 4(a) and 4(b) demonstrate the PSO planning results on both non-smooth and smooth cost maps. Furthermore, we compare our BSCP (from Section IV-A) to the state-of-the-art one [18] by examining the smoothness of the overall trajectory. A hundred consecutive simulations are conducted with randomly sampled initial and target positions in the indoor environment (Figure 4(b)) using both BSCPs, and the vehicle always safely reaches the target. The smoothness is then evaluated with the time-averaged integration of the square of the jerk (denoted as η_j) [13]. The one from [18] gives $\eta_j = 4.44 \times 10^5$, while the proposed BSCP results in a much smoother flight with $\eta_j = 7.87 \times 10^3$. The BSCP in [18] is set to be time optimal for an efficient closed-form solution but causes unnecessary aggressive maneuvers. In contrast, our BSCP optimizes a multi-objective target that includes the square of the jerk, therefore generates a smoother response. In terms of computational efficiency, the two methods are close to each other. The NN of our BSCP can be evaluated in 3 μs in PyTorch with the TX2 GPU while the BSCP in [18] takes 2.5 μs in C++ with an I7 CPU.

VI. EXPERIMENT WITH QUADROTORS

In this Section, the performance of the proposed framework is studied with real flight experiments on quadrotors (the experiment video is in the attached file). The BSCP used here is the one discussed in Section IV-A. The NNs used for the model prediction can be seen from Table I. To solve the optimization problem, we use the PSO algorithm presented in Section V-B. The calculation within each iteration is implemented parallelly to utilize the modern multicore hardware fully. In Table III, we compare the time consumption between evaluating the NN and forward simulation for 20 seconds. The NN helps to increase the efficiency on both the CPU and the GPU.

In Figure 5, the quadrotor is tasked to fly in a 4 m $\times$ 4 m $\times$ 2 m space containing 7 pillars and 13 strings. The online planning is done wirelessly on a laptop equipped with an Intel I7 CPU. The quadrotor has no prior knowledge about the locations of its targets. It is given the next target only if it enters the reaching radius (0.3 m) of the current one. The localization and obstacle sensing are achieved with the VICON system. The environment is represented as a 3D EDT map. For the PSO algorithm, 40 particles are iterated over 20 times, which averagely takes 14 ms on the I7 CPU. The model predictive planning is executed at 10 Hz. The experiment is repeated 4 times consecutively. The quadrotor can reach every target safely in all trails. The maximum speed reached is 1.39 m/s with an average speed of 0.74 m/s. With the 10 Hz replanning rate, the quadrotor is also capable of avoiding dynamic obstacles (see the experiment video).

Moreover, our method also allows for obstacle avoidance while following a reference trajectory. In Figure 6(a), the vehicle follows a given trajectory with an average tracking error of 0.31 m. In Figure 6(b), a pillar is added to block the desired trajectory with a forbidden radius of 1.2 m. The vehicle automatically avoids the obstacle and converges back to the reference trajectory afterward. The reference trajectory can also be given online by another moving object such as a marker in the user's hand (see the experiment video). Finally, the proposed method is tested with sensor-based navigation tasks in an underground car-park which has approximately 0.02 pillars per m^2. The vehicle has no prior information of the environment; the localization and mapping are achieved with a ZED stereo camera and an RP-lidar laser scanner. All algorithms are running onboard with a TX2 computer. The PSO uses 15 particles over 10 iterations which consume 14 ms on the TX2 CPU averagely. In Figure 7(a), the vehicle performs a zig-zag flight among 4 waypoints while avoiding obstacles, the maximum speed reached is 3.25 m/s,

and the average speed is 2.0 m/s. In Figure 7(b), the vehicle flys forward for about 40 m, the maximum speed reached is 2.5 m/s, and the average speed is 1.7 m/s. A total of 12 similar flight experiments are conducted. The total length of the flight is 933.2 m, and the quadrotor is always capable of reaching the targets safely.

VII. Conclusion

In this letter, we have presented a local motion planning framework based on BSCPs and model predictive control. By approximating the BSCP with a neural network, it allows the fast evaluation of local trajectory and its derivative against the cost function. Furthermore, we formally express the local motion planning with BSCPs as an optimization problem where the programming variables uniquely define the end state constraints. With two different optimization techniques, we demonstrate that the resulting optimization problem can be solved efficiently in the continuous domain. It gives an increased performance for tasks that require precise maneuver. It also allows the BSCPs without closed-form solution to be used in online optimization with algorithms such as the PSO. Finally, the proposed method is tested with real flight experiments on quadrotors for various tasks.

References

[1] T. Howard, M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Model-predictive motion planning: Several key developments for autonomous mobile robots," *IEEE Robot. Autom. Mag.*, vol. 21, no. 1, pp. 64–73, Mar. 2014.

[2] A. A. Paranjape, K. C. Meier, X. Shi, S.-J. Chung, and S. Hutchinson, "Motion primitives and 3d path planning for fast flight through a forest," *Int. J. Robot. Res.*, vol. 34, no. 3, pp. 357–377, 2015. [Online]. Available: https://doi.org/10.1177/0278364914558017

[3] Z. Kingston, M. Moll, and L. E. Kavraki, "Sampling-based methods for motion planning with constraints," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 1, no. 1, pp. 159–185, 2018. [Online]. Available: https://doi.org/10.1146/annurev-control-060117-105226

[4] P. Florence, J. Carter, and R. Tedrake, "Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps," in *Proc. Int. Workshop Algorithmic Found. Robot.*, 2016.

[5] B. T. Lopez and J. P. How, "Aggressive 3-D collision avoidance for high-speed navigation," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2017, pp. 5759–5765.

[6] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1:43 scale RC cars," *Optimal Control Appl. Methods*, vol. 36, pp. 628–647, 2015.

[7] X. Yang, K. Sreenath, and N. Michael, "A framework for efficient teleoperation via online adaptation," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2017, pp. 5948–5953.

[8] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online UAV replanning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2016, pp. 5332–5339.

[9] S. Kuindersma *et al.*, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Auton. Robots*, vol. 40, no. 3, pp. 429–455, Mar. 2016. [Online]. Available: https://doi.org/10.1007/s10514-015-9479-3

[10] I. Ulrich and J. Borenstein, "Vfh+: Reliable obstacle avoidance for fast mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 1998, vol. 2, pp. 1572–1577.

[11] D. Ferguson, T. M. Howard, and M. Likhachev, "Motion planning in urban environments: Part I," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2008, pp. 1063–1069.

[12] U. Schwesinger, M. Rufli, P. Furgale, and R. Siegwart, "A sampling-based partial motion planning framework for system-compliant navigation along a reference path," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2013, pp. 391–396.

[13] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Trans. Robot.*, vol. 31, no. 6, pp. 1294–1310, Dec. 2015.

[14] M. Hwangbo, J. Kuffner, and T. Kanade, "Efficient two-phase 3D motion planning for small fixed-wing UAVs," in *Proc. IEEE Int. Conf. Robot. Autom.*, Apr. 2007, pp. 1035–1041.

[15] M. Pivtoraiko, I. A. D. Nesnas, and A. Kelly, "Autonomous robot navigation using advanced motion primitives," in *Proc. IEEE Aerosp. Conf.*, Mar. 2009, pp. 1–7.

[16] J. Tordesillas, B. T. Lopez, J. Carter, J. Ware, and J. P. How, "Real-time planning with multi-fidelity models for agile flights in unknown environments," 2018, *arXiv preprint arXiv: 1810.01035*.

[17] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a frent frame," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2010, pp. 987–993.

[18] M. Hehn and R. D'Andrea, "Real-time trajectory generation for quadrocopters," *IEEE Trans. Robot.*, vol. 31, no. 4, pp. 877–892, Aug. 2015.

[19] E. Frazzoli, M. A. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Trans. Robot.*, vol. 21, no. 6, pp. 1077–1091, Dec. 2005.

[20] S. Schaal, *Dynamic Movement Primitives: A Framework for Motor Control in Humans and Humanoid Robotics*. Berlin, Germany: Springer, 2006, pp. 261–280. [Online]. Available: https://doi.org/10.1007/4-431-31381-8_23

[21] R. Krug and D. Dimitrov, "Model predictive motion control based on generalized dynamical movement primitives," *J. Intell. Robot. Syst.*, vol. 77, no. 1, pp. 17–35, Jan. 2015. [Online]. Available:10.1007/s10846-014-0100-3

[22] I. M. Caireta, "Planning and control of a multiple-quadcopter system cooperatively carrying a slung payload in dynamical environments," Bachelor's thesis, Barcelona School of Informatics, Polytechnic Univ. of Catalonia, Spain, 2019.

[23] I. Lenz, R. A. Knepper, and A. Saxena, "DeepMPC: Learning deep latent features for model predictive control," in *Proc. Robot., Sci. Syst.,* Rome, Italy, 2015.

[24] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2017, pp. 2786–2793.

[25] A. Faust *et al.*, "PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2018, pp. 5113–5120.

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

[27] T. M. Howard, C. J. Green, and A. Kelly, *State Space Sampling of Feasible Motions for High Performance Mobile Robot Navigation Highly Constrained Environments*. Berlin, Germany: Springer, 2008, pp. 585–593. [Online]. Available: https://doi.org/10.1007/978-3-540-75404-6_56

[28] J. C. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 07 2011.

[29] D. Harabor and A. Grastien, "Online graph pruning for pathfinding on grid maps," in *Proc. 25th AAAI Conf. Artif. Intell.*, 2011, pp. 1114–1119. [Online]. Available: http://dl.acm.org/citation.cfm?id=2900423.2900600

[30] H. Zhang, J. Butzke, and M. Likhachev, "Combining global and local planning with guarantees on completeness," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2012, pp. 4500–4506.