Sample Efficient Learning of Path Following and Obstacle Avoidance Behavior for Quadrotors

Stefan Stevšić[1], Tobias Nägeli[1], Javier Alonso-Mora[2], Otmar Hilliges[1]

***Abstract*— In this paper we propose an algorithm for the training of neural network control policies for quadrotors. The learned control policy computes control commands directly from sensor inputs and is hence computationally efficient. An imitation learning algorithm produces a policy that reproduces the behavior of a path following control algorithm with collision avoidance. Due to the generalization ability of neural networks, the resulting policy performs local collision avoidance of unseen obstacles while following a global reference path. The algorithm uses a time-free model predictive path-following controller as a supervisor. The controller generates demonstrations by following few example paths. This enables an easy to implement learning algorithm that is robust to errors of the model used in the model predictive controller. The policy is trained on the real quadrotor, which requires collision-free exploration around the example path. An adapted version of the supervisor is used to enable exploration. Thus, the policy can be trained from a relatively small number of examples on the real quadrotor, making the training sample efficient.**

I. Introduction and Related Work

Many applications of micro aerial vehicles (MAVs) require safe navigation in environments with obstacles and therefore methods for trajectory planning and real-time collision avoidance. Several strategies exist to make this problem computationally tractable. The use of model free controllers with path planning [1] is computationally attractive but requires conservative flight. Model based methods, including local receding horizon methods such as Model Predictive Control (MPC) [2], combining slow global planning with fast local avoidance [3], or avoidance via search of a motion primitive library [4] are computationally demanding, but can achieve more aggressive maneuvers. A theoretical analysis of the dynamical system can provide insights in a limited number of cases [5], [6] leading to faster computation times. These methods have limited scope, taking into account a specific dynamics model. Furthermore, these methods require estimation of obstacle positions from the sensor data. In this paper, we address such issues with a novel imitation learning algorithm, schematically summarized in Fig. 1, that produces control commands directly from sensor inputs.

Producing control signals directly form sensor inputs has two main benefits. First, the algorithm does not require estimation of obstacle positions. Second, function approximators, such as neural networks, can be much more computationally efficient compared to traditional planning methods [2], [3], [4] while still achieving safe flight. Learning can be combined with motion planning. Faust et al. [7] combine learning of a low level controller with a path planning algorithm. This hybrid approach shows that control for quadrotor navigation can be learned, but still requires expensive offline collision avoidance. Our method learns how to avoid collisions and runs in real time.

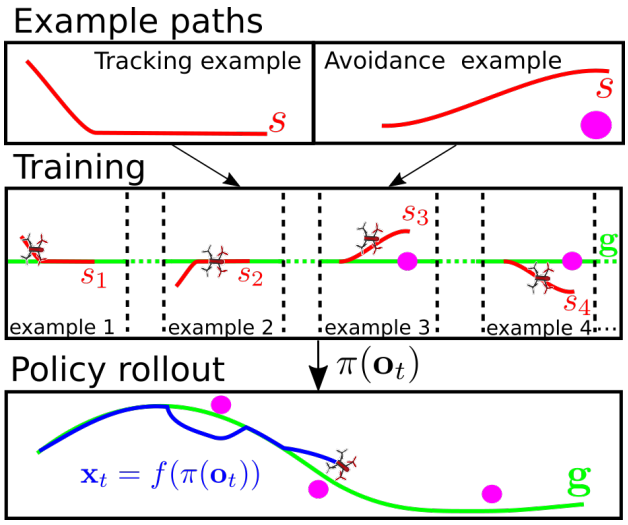

Fig. 1. A policy is learned from few, short local collision avoidance and path following maneuvers (red). The learned policy generalizes to unseen scenes and can track long guidance paths (green) through complex environments while successfully avoiding obstacles (blue).

The most general approach, to learn a controller, here called a control policy, is model-free reinforcement learning (RL) [8], a class of methods that learns the control policy through interaction with the environment. However, these methods are sample inefficient, requiring a large number of trials, and therefore can only be applied in simulation [9]. A more sample efficient option is model-based RL [10], [11], where the model parameters are learned while the control policy is optimized. In this setting, learning the model requires dangerous maneuvers, which can lead to damage of the quadrotor or the environment [12]. A final option is to learn the policy by imitating an oracle, either a human pilot or an optimization algorithm [13]. If the oracle can provide examples of safe maneuvers, this is the most immediate choice to learn policies for real quadrotors.

Imitating the oracle is not a trivial task. Primarily because

This work was supported by in parts by the Swiss National Science Foundation (UFO 200021L_153644) and NWO domain Applied Sciences. We are grateful for their support.

[1]AIT Lab, Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland {stefan.stevsic | naegelit | otmar.hilliges}@inf.ethz.ch

[2]Cognitive Robotics, Delft University of Technology, 2628 CD Delft, Netherlands j.alonsomora@tudelft.nl

data from the ideal trajectory is not enough to learn a policy, since it does not provide examples of correcting drift from the ideal trajectory. In [14], the policy learns to steer the quadrotor from a human pilot demonstration. The biggest challenge was to collect sufficient data, since it is challenging to control the quadrotor manually. As a result, the controls were limited to steering commands while speed and attitude were controlled externally. We resolve this issue by learning the policy from a trajectory optimization oracle.

Imitating a trajectory optimization oracle requires to generate data that can efficiently train the control policy. Two main issues arise when training the policy with this approach. First, efficiently generating training data that can produce the control policy is not straightforward because the trajectory optimization is computationally demanding. Thus in prior work a single trajectory was used to compute control signals for states on the trajectory and states close to the trajectory [15]. However, this approach only works in simulation with a perfectly known model. Second, the algorithm needs to work with an approximate model to be applied on a real world system. In [16], a long horizon trajectory was followed with a short horizon Model Predictive Controller (MPC) to generate training data efficiently. Since the model is not correct, this learning algorithm requires a complex adaptation strategy that guides the control policy to the desired behavior. Alternatively, one could use only a short horizon MPC to provide samples for training of the control policy [12]. However, MPC can produce suboptimal solutions that can lead to deadlocks or collisions during training.

We present an algorithm which produces a control policy by learning from a Model Predictive Contouring Control (MPCC) [17] oracle. Contrary to prior work [16], which relies on tracking of a *timed* trajectory via MPC, MPCC is *time-free*. More specifically: (i) the policy shows faster execution time compared to non-learning approaches. (ii) MPCC allows for an easier to implement training algorithm that is robust to modeling errors. In comparison to the policy obtained by MPC, the MPCC based policy performs better and shows faster convergence behavior. (iii) A collision-free exploration strategy, bounding divergence from the collision-free region during on-policy learning.

The policy can be trained using paths of arbitrary length i.e. the planning horizon length is not a limit as in [12]. As a result of (ii) and (iii), the algorithm is sample efficient, requiring a relatively low number of trials. This results in a training algorithm that can be executed on a real quadrotor.

II. Preliminaries

A. Robot Model

The full state of a quadrotor is 12-dimensional, consisting of quadortor position, velocity, rotation and angular velocity. However, in our experiments we use a 8-dimensional state [18], based on the Parrot Bebop2 SDK inputs. The state is defined by the position $\boldsymbol{p} \in \mathbb{R}^3$, velocity $\boldsymbol{v} \in \mathbb{R}^2$ in x, y directions, and roll ϕ, pitch θ and yaw ψ:

$$\mathbf{x} = [\boldsymbol{p}, \boldsymbol{v}, \phi, \theta, \psi] \in \mathbb{R}^8. \tag{1}$$

The set of feasible states is denoted by $\mathcal{X}$. The control inputs to the system are given by $\mathbf{u} = [v_z, \phi_d, \theta_d, \dot{\psi}_d] \in \mathbb{R}^4$, where v_z is the velocity of the quadrotor in z direction, ϕ_d and θ_d are the desired roll and pitch angles of the quadrotor. The rotational velocity around the z-body axis is set by $\dot{\psi}_d$. The set of feasible inputs is denoted by $\mathcal{U}$. Sets $\mathcal{U}$ and $\mathcal{X}$ are described in Sec. III-B.1. We use a first order low-pass Euler approximation of the quadrotor dynamics. Notice that velocities v_z and $\dot{\psi}_d$ are directly controlled. The velocity v_z directly controls the position dynamics in z direction $\dot{\boldsymbol{p}} = [\boldsymbol{v}, v_z]$. The dynamics of the state velocity vector are:

$$\dot{\boldsymbol{v}} = \boldsymbol{R}(\psi) \begin{bmatrix} -tan(\phi) \\ tan(\theta) \end{bmatrix} a_g - c_d \boldsymbol{v}, \tag{2}$$

where $a_g = 9.81 \frac{m}{s^2}$ is the earth's gravity, $\boldsymbol{R}(\psi) \in SO(2)$ is the rotation matrix only containing the yaw rotation of the quadrotor and c_d is the drag coefficient at low speeds. The rotational dynamics of the quadrotor are given by

$$\dot{\phi} = \frac{1}{\tau_a}(\phi_d - \phi), \;\; \dot{\theta} = \frac{1}{\tau_a}(\theta_d - \theta) \;\text{ and }\; \dot{\psi} = \dot{\psi}_d, \tag{3}$$

where τ_a is the time constant of a low-pass filter. As a result, the position $\boldsymbol{p}$ and velocity $\boldsymbol{v}$ cannot change instantaneously.

B. Control policy

Our work is concerned with dynamical systems, such as quadrotors, described by a state vector $\mathbf{x}$ and controlled via an input vector $\mathbf{u}$ (Fig. 2). We assume that the system has sensors, such as a laser range finder, odometry etc. We denote sensor readings at time t with an observation vector $\mathbf{o}_t$.

Dynamical systems are typically controlled via a manually tuned control law e.g. PID or LQR control. In contrast, we approximate the control law by learning a control *policy* from examples. The policy $\boldsymbol{\pi}(\mathbf{o}_t)$ is a function which, at every time step t, takes the vector $\mathbf{o}_t$ as input and outputs the system control inputs $\mathbf{u}_\pi = \boldsymbol{\pi}(\mathbf{o}_t)$. The control inputs $\mathbf{u}_\pi$ are independent of the time step, producing the same control vector for the same observation, i.e. the control policy is stationary and deterministic.

We learn a policy for local collision avoidance, while following a global guidance $\mathbf{g}$, coarsely describing the desired robot trajectory (cf. Fig. 1, bottom). In our case, the guidance $\mathbf{g}$ is a natural cubic spline. The guidance may be computed off-line based on mission goals or given by human and does not need to be collision-free. For example, maps provide information about walls or buildings, while obstacles like trees or humans are not represented. Thus, a control law that locally and at run time alternates the prescribed path, while continuing to follow the global mission goal is necessary.

C. Policy inputs and outputs

The input to the policy $\boldsymbol{\pi}(\mathbf{o}_t)$ is an observation vector $\mathbf{o}_t = [\boldsymbol{d}_t, \boldsymbol{v}_t, \boldsymbol{l}_t]$ (Fig. 3), consisting of the distance to the guidance $\boldsymbol{d}_t$ in the quadrotor yz-plane, the quadrotor velocity $\boldsymbol{v}_t$ and laser range finder readings $\boldsymbol{l}_t \in \mathbb{R}^{40}$. Distance measurements $\boldsymbol{d}_t$ are obtained by subtracting quadrotor positions $\boldsymbol{p}$ from the current setpoint on the guidance $\mathbf{g}$ (Fig. 2):

$$\boldsymbol{d}_t = (\boldsymbol{p} - \boldsymbol{p}_d) R(\phi_{\mathbf{g}}), \tag{4}$$

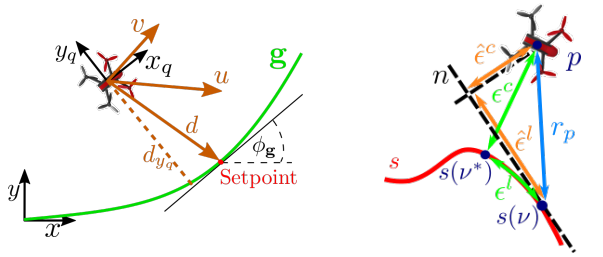

Fig. 2. Left: **Coordinate systems:** Global and quadrotor coordinate systems. The quadrotor coordinate system is denoted with a subscript q. Policy inputs and outputs are always calculated in the quadrotor frame. Right: **Contouring error approximation:** Illustration of the real contouring and lag errors (green) as well as the approximations (orange) used in our MPCC implementation.

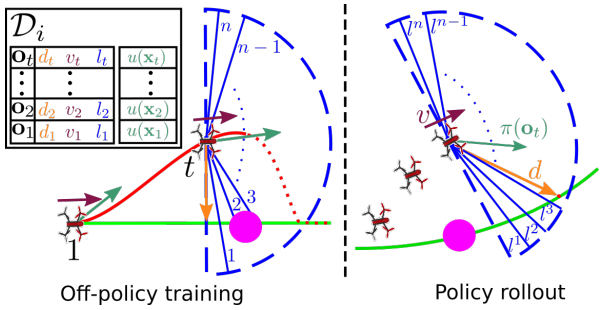

Fig. 3. Left: **Off-policy training example.** Observation data is collected during training while following the example path $\boldsymbol{s}_i$ with the MPCC controller. Right: **Global path following.** The policy produces control inputs based on the current observation vector.

where $R(\phi_\mathbf{g})$ is a rotation matrix around z. The angle $\phi_\mathbf{g}$ is calculated from the global path tangent. The setpoint $\boldsymbol{p}_d$ moves along the path with constant velocity. The distance in x can be omitted since the quadrotor is trained to progress along the guidance $\mathbf{g}$. Quadrotor positions $\boldsymbol{p}$ and velocity measurements $\boldsymbol{v}_t$ are obtained directly from the on-board odometry. The policy $\boldsymbol{\pi}(\mathbf{o}_t)$ outputs continuous control signals: vertical velocity, and roll and pitch angles of the quadrotor $\mathbf{u}_\pi = [v_z, \phi_d, \theta_d]$. The heading is controlled separately (Sec. V-A.1). This enables a simpler learning algorithm compared to learning of the full quadrotor input $\mathbf{u}$. Generally speaking, $\mathbf{o}_t$ could consist of arbitrary sensor data, such as depth images or ultrasound sensor readings.

1) Sensor models: At training time we have access to the full state $\mathbf{x}_t$. To obtained simulated observations $\mathbf{o}_t^s = [\boldsymbol{d}_t^s, \boldsymbol{v}_t^s, \boldsymbol{l}_t^s]$ we calculate $\boldsymbol{d}_t^s$ via Eq. (4), where $\boldsymbol{p}$ is taken directly from $\mathbf{x}_t$. Analogously, $\boldsymbol{v}_t^s$ is taken from $\mathbf{x}_t$. The laser range finder readings are obtained by casting rays from the quadrotor position in the directions of the scanning laser $\boldsymbol{l}_t^s = \boldsymbol{f}_l(\mathbf{x}_t, \boldsymbol{p}_{ob})$, where $\boldsymbol{p}_{ob}$ are obstacles position known at training time. We do not add any noise.

D. Trajectory-tracking (MPC) vs Path-following (MPCC)

Receding horizon tracking is the most common method (e.g., MPC used in [16]) to steer a quadrotor along a trajectory. The trajectory is a sequence of state vectors with associated timings $([\mathbf{x}_i, t_i])_{i=1}^N$. The aim is to position the robot on a *time-parametrized* reference trajectory i.e. to be at a particular position at each time step. The MPC optimization depends on the current time step and the quadrotor states.

We use a *time-free* path-following objective in an MPCC formulation. In path-following control, the robot inputs are optimized to stay close to the desired path $\boldsymbol{s}$ and to make progress along the path [17]. The desired path $\boldsymbol{s}$ is a geometric representation of desired robot positions $\boldsymbol{p}$ during movement. Through the paper, we use $\boldsymbol{s}$ for paths followed by MPCC, while paths followed by the policy $\boldsymbol{\pi}(\mathbf{o}_t)$ are refereed as global guidance $\mathbf{g}$. Control inputs $\mathbf{u}$ are obtained from an optimization problem, which minimizes the distance to the desired path $\boldsymbol{s}$, and maximizes the progress along $\boldsymbol{s}$. The distance to the closest point on $\boldsymbol{s}$ is denoted by the contouring error ϵ^c:

$$\epsilon^c = \|\boldsymbol{s}(\nu^*) - \boldsymbol{p}\|, \tag{5}$$

where $\boldsymbol{s}$ is a cubic spline parametrized with ν (cf. Fig. 2, right). Finding the closest point on the path $\boldsymbol{s}(\nu^*)$ is an optimization problem itself and cannot be solved analytically. We discuss a computationally tractable solution in Sec. III-B.

III. Method

A. Policy Learning Algorithm

We propose an imitation learning algorithm to iteratively refine a control policy $\boldsymbol{\pi}(\mathbf{o}_t)$, learning a general behavior from a set $\mathcal{S} = \{\boldsymbol{s}_i\}_{i=1}^N$ of *short* example paths $\boldsymbol{s}_i$.

The goal of the control policy $\boldsymbol{\pi}(\mathbf{o}_t)$ is to imitate the trajectory produced by the MPCC supervisor, tracking the path $\boldsymbol{s}_i$. We employ supervised learning on the dataset of observation-control mappings $(\mathbf{o}_t, \boldsymbol{u}(\mathbf{x}_t))$. The training data is obtained in a two-step procedure. First, an *off-policy* step generates training samples via tracking the example path $\boldsymbol{s}_i$ with the MPCC oracle (cf. Fig. 3). However, this only produces "ground truth" data, containing samples from the ideal trajectory, which are not enough to train the control policy as observed in DAgger [13]. We gather the necessary additional data by using the partially trained policy in an *on-policy* step. Inevitably the policy outputs $\mathbf{u}_\pi = \boldsymbol{\pi}(\mathbf{o}_t)$ will lead to drift from the ideal trajectory. Correct control inputs $\mathbf{u}^* = \boldsymbol{u}(\mathbf{x}_t)$, corresponding to recorded observations $\mathbf{o}_t$, are computed by the MPCC supervisor after the data collection.

1) Example paths: We provide examples via simple heuristics (Fig. 1), demonstrating returning to the spline at 45^o and showing obstacle avoidance maneuvers starting 3 m from the obstacle and passing it at a distance of 1.5 m. Each skill requires several examples of the same type. Importantly, these paths do not need to take the model of the robot into account. We use 12 example paths in total. The obstacles are cylinders of radius $r = 0.2$ m.

2) Policy representation: The policy is parametrized by a universal function approximator in the form of a neural network. The network parameters define a matrix $\mathbf{W}$. The full notation is $\boldsymbol{\pi}(\mathbf{o}_t; \mathbf{W})$, but we often use $\boldsymbol{\pi}(\mathbf{o}_t)$ for brevity. We use a fully connected network with two hidden layers,

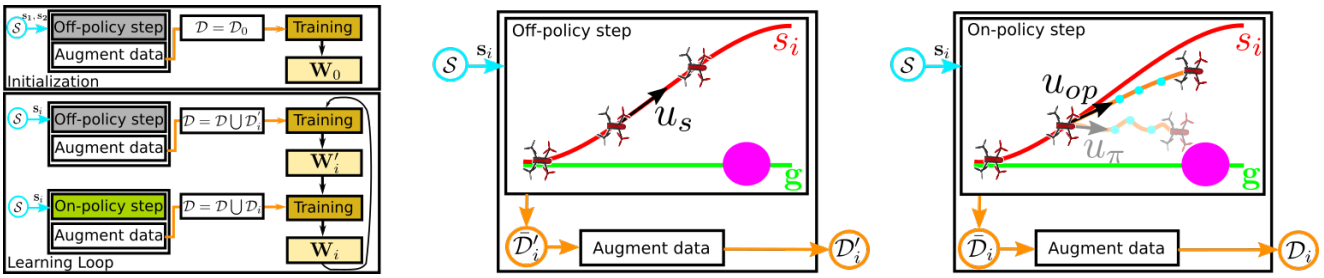

Fig. 4. **Overview:** The algorithm for training the policy $\boldsymbol{\pi}(\mathbf{o}_t)$ (left). *Off-policy* and *on-policy* steps for data collection (middle and right).

each consisting of 30 neurons with softplus activation and linear neurons in the output layer. Initial weights $\mathbf{W}$ are initialized randomly using zero mean normal distribution with standard deviation 0.01.

3) Data collection: To collect data for training, we have two different steps for which we use two different controllers: *off-policy step* (MPCC) and *on-policy step* (on-policy MPCC). Each of the steps is used to collect training data. The quadrotor tracks the given path $\boldsymbol{s}_i$ using the respective controller and we collect the observation samples $\mathbf{o}_t$ and system state $\mathbf{x}_t$ at each time step.

Off-policy step (MPCC path tracking): In this learning step, "ground truth" training samples are collected while the quadrotor tracks the given path $\boldsymbol{s}_i$ using the MPCC supervisor. After the off-policy step the dataset $\mathcal{D}$ contains only ideal trajectory data.

On-policy step (on-policy MPCC path tracking): We propose an exploration approach to visit the states $\mathbf{x}_t$ that the non-fully trained policy $\boldsymbol{\pi}(\mathbf{o}_t)$ would visit. For exploration, we use an *on-policy MPCC* that generates control inputs $\mathbf{u}_{op}$ (cf. Sec. III-C). The on-policy MPCC optimization cost balances between following the current policy output $\mathbf{u}_\pi = \boldsymbol{\pi}(\mathbf{o}_t)$ and minimizing the contouring error, pulling the quadrotor back to the path (cf. Fig. 4, right). This enables the collision-free exploration. Here, we assume that the region around the example path is safe and obstacle-free.

4) Data augmentation and training: The training dataset is constructed from collected observations $\mathbf{o}_t$ and states $\mathbf{x}_t$. For each state $\mathbf{x}_t$, the MPCC supervisor computes the optimal trajectory and control inputs in the horizon, with respect to the path $\boldsymbol{s}_i$. However, only the first control input $\mathbf{u}^* = \boldsymbol{u}(\mathbf{x}_t)$ is used as a training sample:

$$\bar{\mathcal{D}}_i = \{(\mathbf{o}_t, \boldsymbol{u}(\mathbf{x}_t)), t = 1..n\}. \tag{6}$$

We add noisy samples to the dataset $\bar{\mathcal{D}}_i$ to prevent overfitting during training [15]. First, Gaussian noise is added to every state $\mathbf{x}_t$ collected during path tracking. The resulting noisy states $\mathbf{x}_t + \boldsymbol{w}_t$ are used to calculate corresponding input samples $\boldsymbol{u}(\mathbf{x}_t + \boldsymbol{w}_t)$ via the MPCC supervisor. Observation samples $\mathbf{o}^s(\mathbf{x}_t + \boldsymbol{w}_t)$ are obtained by calculating the exact observations from the noisy states using sensor models (cf. Sec. II-C.1). For each real sample, we add three noisy samples to augment the dataset $\bar{\mathcal{D}}_i$:

$$\mathcal{D}_i = \bar{\mathcal{D}}_i \bigcup \{(\mathbf{o}^s(\mathbf{x}_t + \boldsymbol{w}_{tk}), \boldsymbol{u}(\mathbf{x}_t + \boldsymbol{w}_{tk})), \\ t = 1..n, k = 1..3\}. \tag{7}$$

We add the augmented dataset $\mathcal{D}_i$ to the global dataset $\mathcal{D} = \mathcal{D} \bigcup \mathcal{D}_i$. Using the new dataset $\mathcal{D}$, the policy $\boldsymbol{\pi}(\mathbf{o}_t)$ is trained via optimizing the mean squared error (MSE) on $\mathcal{D}$:

$$\min_{\mathbf{W}} \sum_{\mathbf{o}_j, \mathbf{u}_j^* \in \mathcal{D}} \|\boldsymbol{\pi}(\mathbf{o}_j; \mathbf{W}) - \mathbf{u}_j^*\|_2^2. \tag{8}$$

The neural network is trained incrementally by initializing the network weights $\mathbf{W}$ from the previous solution. We use the ADAM optimization algorithm for training.

5) Algorithm: The algorithm (Fig. 4) requires only a set of example paths $\mathcal{S}$ as input. Data collection is done on the real quadrotor because the on-policy data depends on the error of the approximate model. These are the most important steps:

- **Initialization**. We execute two *off-policy* data collection steps on two return-to-guidance paths selected at random. The data is augmented (see Sec. III-A.4) and the initial policy is trained. The initial policy needs enough data to ensure stable performance in the *on-policy* step.
- **Learning loop**. During training the algorithm alternates between *off-policy* and *on-policy* data collection steps, augmenting data, and re-training the policy after every step using the remaining samples $\mathcal{S} \setminus \{\boldsymbol{s}_1, \boldsymbol{s}_2\}$. The *off-policy* step collects ground truth data. The *on-policy* step helps to correct the behavior of the intermediate policy.
- **Output.** The final policy is trained from different examples. This enables the policy to generalize to different obstacle positions beyond the ones in the training set.

B. Policy supervisor (MPCC)

To follow the path we seek to minimize the contouring error ϵ^c defined in Eq. (5) and maximize the progress along the same path $\boldsymbol{s}$. To solve this problem we follow the formulation of [18]. We introduce an initial guess $\boldsymbol{s}(\nu)$ of the closest point $\boldsymbol{s}(\nu^*)$, which is found by solving the MPCC problem Eq. (11). The integral over the path segment between the closest point $\boldsymbol{s}(\nu^*)$ and location of $\boldsymbol{s}(\nu)$ denotes the lag error ϵ^l. To attain a tractable formulation, the errors ϵ^l and ϵ^c are approximated by projecting the current quadrotor position $\boldsymbol{p}$ onto the tangent vector $\boldsymbol{n}$, with origin at the current path position $\boldsymbol{s}(\nu)$ (Fig. 2, right). The relative vector between $\boldsymbol{p}$ and the tangent point $\boldsymbol{s}(\nu)$ can be written as

$\boldsymbol{r_p} := \boldsymbol{s}(\nu) - \boldsymbol{p}$. Using the path derivative $\boldsymbol{s'} := \frac{\partial \boldsymbol{s}(\nu)}{\partial \nu}$, the normalized tangent vector $\boldsymbol{n} = \frac{\boldsymbol{s'}}{\|\boldsymbol{s'}\|}$ is found. The approximated error measures are then given by:

$$\hat{\epsilon}^l = \|\boldsymbol{r_p}^T \boldsymbol{n}\|, \tag{9a}$$

$$\hat{\epsilon}^c = \|\boldsymbol{r_p} - \left(\boldsymbol{r_p}^T \boldsymbol{n}\right) \boldsymbol{n}\|, \tag{9b}$$

With these error measures, we define a stage cost function:

$$J_k = K_c(\hat{\epsilon}^c_k)^2 + K_l(\hat{\epsilon}^l_k)^2 - \beta \dot{\nu}_k, \tag{10}$$

where the subscript k indicates the horizon stage in Eq. 11. J_k represents the trade-off between path following accuracy and progress along the path, where $\dot{\nu}$ is a velocity of the parameter ν, describing path progress and $\beta \geq 0$ is a scalar weight. The scalar weight K_l determines the importance of the lag error and is set to a high value which gives better approximations of the closest point $s(\nu_k)$. The admissible contour error is controlled by the weight K_c.

1) MPCC Formulation: The trajectory and control inputs of the drone at each time step are computed via solving the following N-step finite horizon constrained optimization problem at time instant t:

$$\underset{\mathbf{u},\mathbf{x},\nu,\dot{\nu}}{\text{minimize}} \quad \sum_{k=0}^{N} J_k + \mathbf{u}_k^T \boldsymbol{R} \mathbf{u}_k \tag{11}$$

$$\begin{aligned}
\text{subject to} \quad & \mathbf{x}_{k=0} = \mathbf{x}_t && \text{(Initial state)} \\
& \nu_{k=0} = \nu_t && \text{(Initial path parameter)} \\
& \mathbf{x}_{k+1} = \boldsymbol{f}_m(\mathbf{x}_k, \mathbf{u}_k) && \text{(Robot dynamics)} \\
& \nu_{k+1} = \nu_k + \dot{\nu}_k T_s && \text{(Progress path)} \\
& 0 \leq \nu_k \leq l_{path} && \text{(Path length)} \\
& \mathbf{x}_k \in \mathcal{X}, && \text{(State constraints)} \\
& \mathbf{u}_k \in \mathcal{U}, && \text{(Input constraints)}
\end{aligned} \tag{12}$$

where $\boldsymbol{R}$ is a positive definite penalty matrix avoiding excessive use of the control inputs. The vector $\mathbf{x}_t$ and the scalar ν_t denote the values of the current states $\mathbf{x}$ and ν, respectively. The scalar T_s is the sampling time. The state constraints $\mathcal{X}$ limit roll and pitch angles ϕ, θ to prevent the quadrotor from flipping. The input constraints $\mathcal{U}$ are set according to the quadrotor's allowed inputs. This non-linear problem under constraints (11) can be formulated in standard software, e.g. FORCES Pro [19], where efficient code can be generated for real-time solving.

C. Exploration algorithm (On-policy MPCC)

For *on-policy* learning we apply a variant of the above MPCC, attained by adding a following cost to Eq. (10):

$$c_k = \|\mathbf{x}_{\pi k} - \mathbf{x}_k\|_2^2. \tag{13}$$

This term trades-off visiting states $\mathbf{x}_{\pi k}$ obtained by rolling-out the policy $\boldsymbol{\pi}(\mathbf{o}_t)$, while keeping the quadrotor close to the input path $\boldsymbol{s}_i$. The main difference compared to the *off-policy* supervisor is a larger admissible contouring error $\hat{\epsilon}^c$. In simulation the policy $\boldsymbol{\pi}(\mathbf{o}_t)$ is rolled-out over the entire horizon length to obtain the predicted quadrotor state $\mathbf{x}_{\pi k}$. The observation vector $\mathbf{o}_k$ is computed from these states using the sensor models (cf. Sec. II-C.1).

The cost used in the on-policy MPCC is similar to the one presented in PLATO [12], where the quadrotor tries to greedily follow the policy output in the first state, while keeping the standard objective in the next states. We build on top of this cost to improve safety during the exploration. The *on-policy* MPCC observes all policy states in the horizon $\mathbf{x}_{\pi k}$ which provides more complete information about the states. Furthermore, the exploration area is more precisely defined because the contouring cost is directly proportional to the distance from the collision-free example path.

IV. Method Discussion

A. Generalization and Limitations

It is important to note that our learning algorithm never sees entire trajectories. Instead we provide multiple, short examples of a class of behavior. They provide guidance on how to react in different *instances* of the same problem. The final policy $\boldsymbol{\pi}(\mathbf{o}_t)$ generalizes to unseen scenarios (cf. Fig. 5), following paths much longer than those seen during training.

These generalization properties can be explained from a machine learning perspective. Neural networks are universal function approximators, able to learn a function from a set of in- and output pairs. In our case, we assume that samples come from a non-linear stochastic function

$$\mathbf{u}^* = \boldsymbol{f}_{nn}(\mathbf{o}_t) + \boldsymbol{\varepsilon},$$

where $\boldsymbol{\varepsilon}$ is zero mean Gaussian noise $\mathcal{N}(\mathbf{0},\ \boldsymbol{\sigma})$. The control inputs $\mathbf{u}^*$ directly depend on the system state $\mathbf{x}_t$, but we assume partial observability of the state $\mathbf{x}_t$ from the observation $\mathbf{o}_t$. The function output $\mathbf{u}^*$ can be described by the conditional probability distribution

$$p\left(\mathbf{u}^* \mid O = \mathbf{o}_t; \mathbf{W}\right) = \mathcal{N}(\boldsymbol{\mu},\ \boldsymbol{\sigma}),$$

where the distribution mean $\boldsymbol{\mu} = \boldsymbol{f}_{nn}(\mathbf{o}_t)$ is parametrized by the neural network. Given the sample pair $(\mathbf{o}_j, \mathbf{u}_j^*)$, for fixed network parameters $\mathbf{W}$, we can calculate the probability $P(U = \mathbf{u}_j^* \mid O = \mathbf{o}_j; \mathbf{W})$. Maximum log likelihood estimation (MLE) yields the neural network parameters $\mathbf{W}$:

$$\mathbf{W} = \arg\max_{\mathbf{W}} \sum_{\mathbf{o}_j, \mathbf{u}_j^* \in \mathcal{D}} ln\ P(U = \mathbf{u}_j^* | O = \mathbf{o}_j; \mathbf{W}).$$

Since a Gaussian distribution is assumed, the mean can be obtained directly via the MSE loss in Eq. (8).

The policy $\boldsymbol{\pi}(\mathbf{o}_t)$ is trained sequentially on example paths to achieve sample-efficient learning. However, the policy $\boldsymbol{\pi}(\mathbf{o}_t)$ directly depends on the statistics obtained from the training samples in the final dataset $\mathcal{D}$. The MLE principle provably provides the best fit to the given samples, which leads to good generalization properties in cases where test samples come from the same (or a very similar) distribution. Our training set only partially covers the full space of possible observations, which results in successful avoidance of similarly sized and shaped obstacles, but not in avoidance

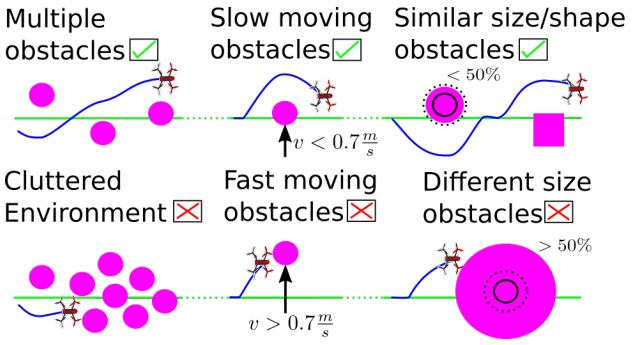

Fig. 5. **Generalization & limitations.** Schematic of settings the policy generalizes to and limits of generalization. We experimentally verified that obstacles moving up to $0.7\frac{m}{s}$ perpendicular to the quadrotor direction can be successfully avoided, while faster moving obstacles cannot. Changing the obstacle diameter up to 50% compared to training, results in satisfying behavior. Further, different shaped obstacles of similar size can be avoided.

of very different obstacles since they produce different observations. For moving obstacles, the observations are the same but the underlying true states of the world are different. The training set does not provide any examples of control inputs for moving obstacles.

Due to the nature of neural networks no formal guarantees regarding avoidance or stability can be given. We show experimentally that our approach works well in practice (see Sec. V). Finally, the results presented here are obtained by training the policy on a single static obstacle. The policy can be trained incrementally, e.g. adding larger obstacles.

B. Comparison to related work

While the proposed learning algorithm bears similarity to DAgger [13], it differs in important aspects. The proposed *on-policy* step maintains the sample efficiency of the original approach but makes exploration collision-free by using control inputs $\mathbf{u}_{op}$. It has been shown that directly applying outputs from intermediate policies can lead to crashes [12]. We analyze the exploration scheme in depth in Sec. V-D.

Applying general model based RL, where the true model is obtained during policy training, requires rollouts of the not fully trained policy, which in the case of quadrotors can lead to catastrophic failure [16]. Designing safe model based RL for quadrotors is not a trivial problem and hence adaptive learning techniques based on approximate dynamics have been used [12], [16]. We follow this approach.

When tracking the timed reference based on the approximate model using MPC [16], similar or identical states can be reached at different time steps. This results in ambiguous mappings of *different* control inputs for *similar* or identical states. In the case of MPCC supervision, the control vector $\mathbf{u}^*$ will be the same for a given state $\mathbf{x}_t$. This results in less ambiguous data and a more robust control policy $\boldsymbol{\pi}(\mathbf{o}_t)$ which we experimentally verify in Sec. V-C.

V. Experimental Results

To assess the proposed policy learning scheme we conducted experiments both in simulation (policy trained in simulation) and in real settings (policy trained on real robot).

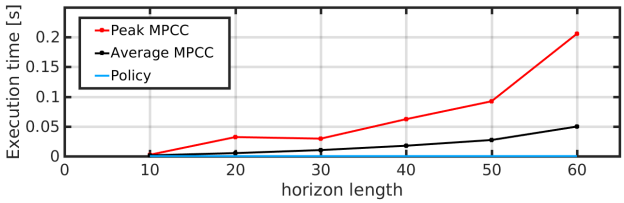

Fig. 6. **Execution time:** Horizon length wrt. execution time of controllers. The control policy imitates a long horizon behavior having the same computation time of $2 \cdot 10^{-4}$ s.

A. Implementation Details

1) Global path following: The global guidance $\mathbf{g}$ coarsely specifies quadrotor motion, but does not need to be aware of obstacles. The policy controls the ϕ_d, θ_d angles and the z-velocity of the quadrotor, while the yaw angle is controlled separately with a simple PD controller to ensure that the quadrotor always faces the direction of the global path (the distance sensor points in this direction). This parametrization allows for training on straight guidance splines, while at test time this can be applied to arbitrary splines (Fig. 3, B).

2) Hardware and simulation setup: We evaluate our method in a full physics simulation, using the Rotors quadrotor physics model [20] in Gazebo [21], and a Parrot Bebop 2 quadrotor for real world experiments. We use a Vicon system to simulate the sensor readings, using the method described in Sec II-C. In Experiment V-C.1 we use a simple MATLAB simulation that implements the model given in Sec. II-A.

B. Comparison with Non-learning Methods

1) Runtime MPCC vs. policy: First, we evaluate our method in terms of computational cost by comparing it to a trajectory optimization method. The baseline is a MPCC (cf. Sec. III-B) with an additional collision avoidance cost [18]. The sampling time of the MPCC is set to 0.1 s.

Fig. 6 shows that both average and peak time, measured over 3 iterations, of the MPCC increase depending on the horizon length. The policy can be trained to imitate a long horizon behavior while maintaining constant runtime.

2) Policy evaluation - simulation: We qualitatively evaluate the learned policy. A long, non-linear guidance is generated and we randomly place obstacles (cf. Fig. 10, left). To attain quantitative results we increase the density of the obstacles along the path of a length of $200m$. For comparison, we use an artificial potential field (APF) method, which has similar computational cost. The potential field pushes the quadrotor to track the global guidance, while repelling it from obstacles. The quadrotor follows a constant velocity reference in the direction of the potential field derivative.

Fig. 7 summarizes the average flight distance from three rollouts. The APF velocity reference is set to the average speed of the policy ($1.3\frac{m}{s}$). For non-trivial cases, the average flight of APF is shorter (cf. Fig. 7). Further, the APF method does not consider the robot dynamics which in

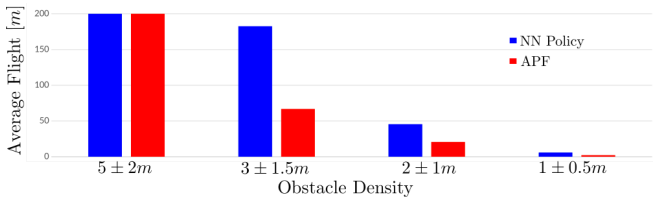

Fig. 7. **Average flight distance:** Distance to collision on different obstacle courses (higher is better). Blue (*ours*), red (APF).

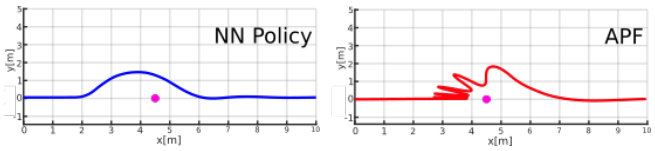

Fig. 8. **Comparison trajectories:** Trajectories while avoiding a single obstacle positioned on the guidance $\mathbf{g}$.

consequence produces non-smooth trajectories (cf. Fig. 8). Furthermore, APF is only suited for slow maneuvers. Our policy generalizes to much harder cases with obstacles closer to each other than seen at train time. However, once the density surpasses $2 \pm 1m$ the flight length drops drastically.

C. Supervision Algorithm - Comparison with the Baseline

One of the main contributions in this work is the MPCC-based path-following supervisor. To evaluate its impact, we compare to a MPC-based trajectory-tracking baseline. For the baseline we obtain an exploration algorithm by augmenting the original MPC objective with the cost in Eq. (13). Both supervisors are tuned for best learning performance, while producing similar task performance.

1) Single obstacle environment: To evaluate robustness with respect to model errors we perturb the value of the discretized time constant $\alpha = e^{-\frac{1}{\tau_a}T_s} = 0.85$ used in the supervisor's robot model. Only in this experiment, we use a MATLAB simulation and only use position and velocity measurements as policy inputs.

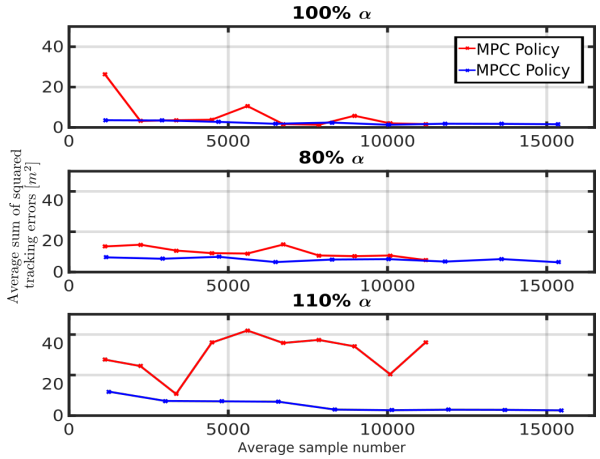

Fig. 9. **Policy robustness:** Policy performance as a function of the supervisor. The average error from three trained policies are shown. The error is bounded to 50. From experiments, we found that error below 10 gives satisfactory performance. Lower is better.

TABLE I

COMPARISON WITH THE BASELINE SUPERVISOR

Task	MPC policy	MPCC policy
Max. tracking deviation z axis	$0.847\ m$	$0.077\ m$
Average flight length	$41.67\ m$	$183.3\ m$

The task is to learn a single maneuver from four examples, each starting at different positions. At test time we roll-out the policy from six different positions. We compute the error as sum of squared distances of quadrotor positions from the ground truth. This error measures how accurate the policy imitates the supervisor.

Fig. 9 shows that our learning scheme leads to superior robustness and faster convergence compared to the MPC baseline under modeling errors. The baseline achieves desirable scores using the correct model parameters, but convergence behavior is unstable or slower in presence of modeling errors. Even with the true model parameter, MPCC yields faster convergence behavior. Besides modeling errors we have unmodeled effects on the real system, which may lead to unstable convergence of MPC-based schemes.

2) Multi-obstacle environment: In this experiment we compare the MPCC supervisor to the MPC baseline in the Gazebo simulator. The simulator implements complex quadrotor dynamics [20]. Contrary to the previous experiment, the policies are trained for the final task i.e. guidance tracking with collision avoidance. We train the policies on the same number of examples (12). The examples for the MPCC supervisor are generated by our algorithm, while the examples for MPC are generated by the off-line trajectory optimization algorithm. The trajectory optimization cost is adjusted so that the quadrotor follows the global guidance with constant speed while avoiding the obstacles.

Table I summarizes the results. We were able to train the policy with a MPC supervisor, but the performance of the policy was not satisfactory. The first issue is that the quadrotor cannot follow the global guidance, drifting from the prescribed path in the z direction. Although the policy performance is not satisfactory, we still evaluated the policy on the obstacle course. On the obstacle course with density of $3 \pm 1.5\ m$ along the path, the average flight length of the MPC policy is $41.67\ m$ which is significantly lower than the policy trained with the MPCC ($183.3\ m$) supervisor. In the light of the previous experiments these results are logical since the policy trained with MPC is not able to accurately follow trajectories in the presence of modeling errors.

D. Evaluation of Collision-free Exploration

The choice of the contouring penalty directly impacts which states are being visited in the on-policy step. Table II summarizes results for different values, measured as sum of squared distances from the example paths during training and from ground truth at test time under different parameters (averaged over trajectories).

We were not able to train the policy by using intermediate policies for exploration in the on-line step of the algorithm (*unsafe*). A too small contouring cost ($K_c = 0.1$) leads to

TABLE II

EXPLORATION ALGORITHM PERFORMANCE

	unsafe	$K_c = 0.1$	$K_c = 10$	$K_c = 25$
Collisions	Train time	Test time	No	Test time
Train error	/	2.75 m	1.38 m	1.02 m
Test error	/	1.99 m	0.84 m	2.55 m

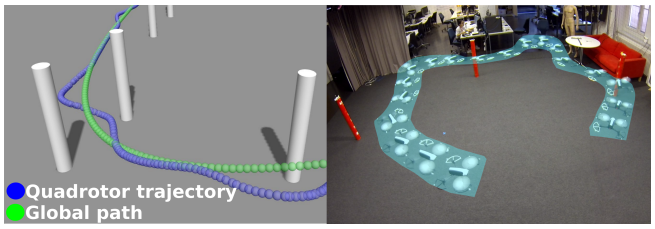

Fig. 10. Left: **Policy roll-out:** Unseen test scene including long guidance (green), obstacles and flown policy roll-out (blue). Right: **Static obstacle.** Policy roll-out in the real environment. Three obstacles are positioned along a circular reference.

large deviation from the example path (high train error) and results in poor generalization (high test error). Too large penalization of the contouring cost ($K_c = 25.0$) suppresses exploration and leads to overfitting (high test error).

E. Policy Evaluation

1) Policy generalization - simulation: We test the generalization ability with different obstacles. Various courses are obtained as in Sec. V-B.2 (density 3 ± 1.5), and we change the obstacle types. We increase the obstacle radius up to 50 % where the policy begins to predict invalid outputs (NaN). The policy successfully avoids cubic obstacles of similar size as the training obstacles. We conclude that the size of the obstacle is the critical factor for generalization.

Next, we evaluate the policy on obstacles that are moving perpendicular to the global guidance path. The obstacle velocity is gradually increased, until collision occurs at $0.7\frac{m}{s}$. Moving obstacles reduce the effective lateral robot speed and no such behavior was observed during training.

2) Policy evaluation - real: We conduct similar experiments on a physical quadrotor, positioning obstacles directly on the desired path (Fig. 10, right). Due to the small experimental space we reduce the avoidance onset to $d = 2m$. No collisions occur and the course is always completed.

A final experiment evaluates policy under moving obstacles such as humans. In our experiments the robot successfully avoids slow moving targets, keeping away from the human at distances similar to training time. Please refer to the video (https://youtu.be/eEqzhglPjNE) for more results.

VI. CONCLUSION

We have proposed a method for learning control policies using neural networks in imitation learning settings. The approach leverages a *time-free* MPCC path following controller as a supervisor in both off-policy and on-policy learning. We experimentally verified that the approach converges to stable policies which can be rolled out successfully to unseen environments both in simulation and in the real-world. Furthermore, we demonstrated that the policies generalize well to unseen environments and have initially explored the possibility to roll out policies in dynamic environments.

REFERENCES

[1] S. Grzonka, G. Grisetti, and W. Burgard, "A fully autonomous indoor quadrotor," *IEEE Transactions on Robotics*, vol. 28, pp. 90–100, 2012.

[2] M. W. Mueller and R. D'Andrea, "A model predictive controller for quadrocopter state interception," *Control Conference (ECC), 2013 European*, pp. 1383–1389, 2013.

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

[4] M. Pivtoraiko, D. Mellinger, and V. Kumar, "Incremental micro-uav motion replanning for exploring unknown environments," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 2452–2458.

[5] E. Frew and R. Sengupta, "Obstacle avoidance with sensor uncertainty for small unmanned aircraft," in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, vol. 1. IEEE, 2004, pp. 614–619.

[6] E. J. Rodríguez-Seda, C. Tang, M. W. Spong, and D. M. Stipanović, "Trajectory tracking with collision avoidance for nonholonomic vehicles with acceleration constraints and limited sensing," *The International Journal of Robotics Research*, vol. 33, no. 12, pp. 1569–1592, 2014.

[7] A. Faust, I. Palunko, P. Cruz, R. Fierro, and L. Tapia, "Automated aerial suspended cargo delivery through reinforcement learning," *Artificial Intelligence*, 2014.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[9] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A survey on policy search for robotics," *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[10] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *NIPS*, 2007.

[11] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[12] G. Kahn, T. Zhang, S. Levine, and P. Abbeel, "Plato: Policy learning using adaptive trajectory optimization," *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 3342–3349, 2017.

[13] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning." in *AISTATS*, vol. 1, no. 2, 2011, p. 6.

[14] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive uav control in cluttered natural environments," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1765–1772.

[15] I. Mordatch, K. Lowrey, G. Andrew, Z. Popovic, and E. V. Todorov, "Interactive control of diverse complex characters with neural networks," in *NIPS*, 2015.

[16] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 528–535.

[17] D. Lam, C. Manzie, and M. Good, "Model predictive contouring control," in *49th IEEE Conference on Decision and Control (CDC)*. IEEE, 2010, pp. 6137–6142.

[18] T. Nägeli, L. Meier, A. Domahidi, J. Alonso-Mora, and O. Hilliges, "Real-time planning for automated multi-view drone cinematography," in *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2017.

[19] A. Domahidi and J. Jerez, "FORCES Pro: code generation for embedded optimization," 2016, https://www.embotech.com/FORCES-Pro.

[20] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "Robot operating system (ros)," *Studies Comp.Intelligence Volume Number:625*, vol. The Complete Reference (Volume 1), p. Chapter 23, 2016.

[21] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems (IROS), 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2149–2154.