This paper has been accepted for publication at the
IEEE International Conference on Robotics and Automation (ICRA), London, 2023. ©IEEE

User-Conditioned Neural Control Policies for Mobile Robotics

Leonard Bauersfeld, Elia Kaufmann, Davide Scaramuzza

***Abstract*— Recently, learning-based controllers have been shown to push mobile robotic systems to their limits and provide the robustness needed for many real-world applications. However, only classical optimization-based control frameworks offer the inherent flexibility to be dynamically adjusted during execution by, for example, setting target speeds or actuator limits. We present a framework to overcome this shortcoming of neural controllers by conditioning them on an auxiliary input. This advance is enabled by including a feature-wise linear modulation layer (FiLM). We use model-free reinforcement-learning to train quadrotor control policies for the task of navigating through a sequence of waypoints in minimum time. By conditioning the policy on the maximum available thrust or the viewing direction relative to the next waypoint, a user can regulate the aggressiveness of the quadrotor's flight during deployment. We demonstrate in simulation and in real-world experiments that a single control policy can achieve close to time-optimal flight performance across the entire performance envelope of the robot, reaching up to 60 km/h and 4.5 g in acceleration. The ability to guide a learned controller during task execution has implications beyond agile quadrotor flight, as conditioning the control policy on human intent helps safely bringing learning based systems out of the well-defined laboratory environment into the wild.**

Video: `https://youtu.be/rwT2QQZEH6U`

I. Introduction

Recently, learned controllers have become extremely popular in the mobile robotics community due to their success in a variety of complex real-world tasks, such as legged locomotion in challenging environments [1], underground exploration [2], autonomous drone racing [3]–[5], and virtual car racing [6]. In all the aforementioned works, neural-network controllers outperform their classical model-based counterparts both in terms of performance and success rate. However, this performance comes at the expense of adaptability, as the control approaches are trained to overfit on a narrowly defined task. A standard neural controller can only rigidly execute the specific task that it has been trained on and lacks the versatility of traditional model-based control. Consider, for example, a mobile robot tasked with time-optimal navigation: using model-predictive control (MPC) it would be straightforward to limit the maximum acceleration during deployment by adjusting the actuator constraints inside the model [7]. However, most neural controllers cannot be regulated and naively adding an additional input to the learned policy may not lead to the desired performance.

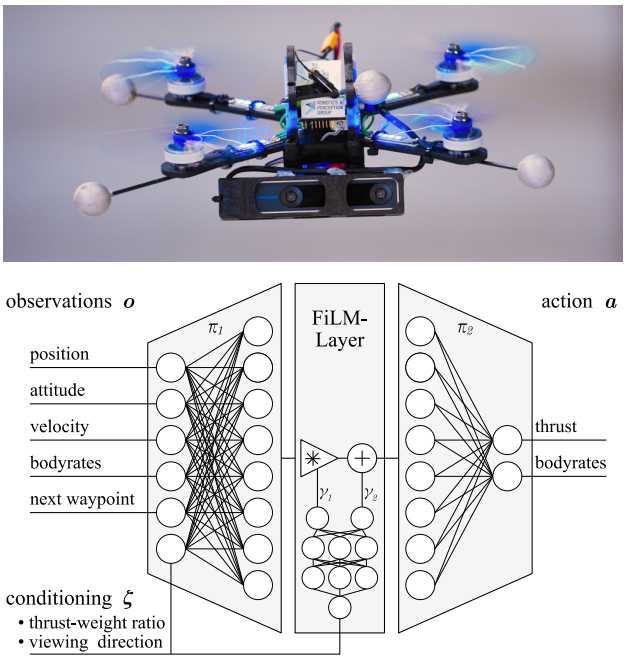


Fig. 1. Conditioning a control policy for agile quadrotor flight on an auxilliary input can be achieved through a FiLM architecture [8]. There, the intermediate activations of a policy that directly maps observations to control commands are linearly transformed based on the conditioning signal supplied by the user. In this work, we study conditioning on the maximum thrust-to-weight ratio (agility) and the viewing direction of the drone w.r.t the next waypoint.

This paper proposes an approach to alleviate the drawback of rigid task execution of learning-based controllers by conditioning the control policies on an auxiliary input which an operator (human, high-level planner) can then set to influence the neural controller as shown in Fig. 1. Aside from increasing the versatility, training a policy that can not only react to the environment but also condition its computed control actions on human intent allows safely bringing learning-based systems out of the controlled laboratory environment into the wild.

Training an embodied agent that can react to user inputs is a difficult endeavor as it requires to learn an entire *distribution* of policies, as opposed to learning a static policy that maps sensory observations to actions. Prior work primarily exists in the context of robotic manipulation conditioned on visual or natural language input [9], [10]. This so-called *multi skill learning* strongly focuses on handling complex visual or natural language queries [11] while the robotic system is mostly simulated and has minimal complexity from a control engineering perspective. Closely related are the visual question answering tasks encountered in the computer

The authors are with the Robotics and Perception Group, Department of Informatics, University of Zurich, and Department of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland (`http://rpg.ifi.uzh.ch`). This work was supported by the Swiss National Science Foundation (SNSF) through the National Centre of Competence in Research (NCCR) Robotics, the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 871479 (AERIAL-CORE), and the European Research Council (ERC) under grant agreement No. 864042 (AGILEFLIGHT).

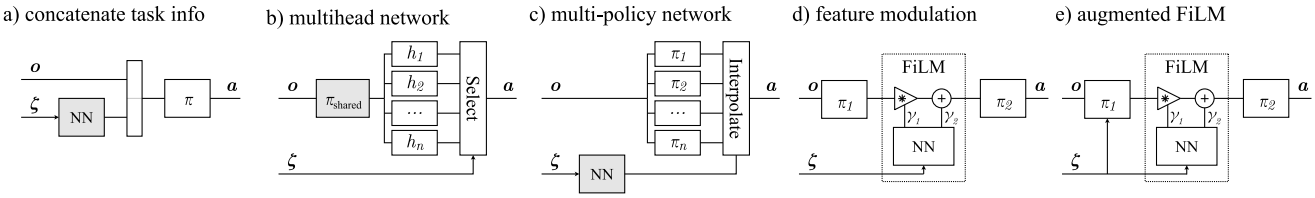


Fig. 2. Overview of the different architectures for conditioning of neural networks commonly found in literature. Boxes in gray represent optional components that can also be replaced with a direct connection. The variable $\boldsymbol{o}$ denotes some vector of observations (e.g. the system state) supplied to the policy. The conditioning signal is denoted by $\boldsymbol{\zeta}$ and the output of the network (e.g. control action) is denoted by $\boldsymbol{a}$.

vision community, where networks are again conditioned on complex natural language user queries [8]. In the context of mobile robotics, to the best of the authors' knowledge, only one prior work [12] exists where conditioning has been applied for three discrete user inputs: a remote-controlled car is trained to either turn left, go straight, or turn right at intersections by using a control network with a shared encoder and three disjoint network heads that are selected based on the operator's input.

A. Contribution

We present the first learning-based controller for an autonomous mobile robot—an agile quadrotor platform is used in this work—where the vehicle's agility and its viewing direction can be influenced through a continuous conditioning input supplied by a user. This advance is made possible by integrating a modified version of the feature-wise linear modulation layer (FiLM) [8] into the neural network controller, which is trained using model-free reinforcement learning. To support our choice of the FiLM architecture, we present a large ablation study which compares the commonly used methods in multi-skill learning tasks. The FiLM approach outperforms multi-head networks as well as a naive feature concatenation baseline in terms of performance and robustness. Finally, we demonstrate the applicability of our proposed method to real-world mobile robotics by conditioning a control policy for perception-aware, near time-optimal quadrotor flight. The continuous user input regulates the desired agility level or guides a perception objective. Furthermore we show that there is a less than a 2% performance difference between a single policy conditioned on a user-specified agility level and a set of overfit policies that can only operate at a fixed level.

II. Related Work

Outside the field of mobile robotics, conditioning neural networks on auxiliary user inputs has been studied in recent years for a variety of applications. In most of them, the conditioning signal is given by a natural language prompt specified by the user and the conditioned network is tasked with answering a question [8], controlling a robotic manipulator arm [9], [10] in a specified way, or planning a path such that a vehicle visits specific areas on a map [13]. Other tasks studied in literature range from optimally encoding information [14] to throwing simulated darts at different targets [15]. Yet, the only application to mobile robotics is driving a remote-controlled car autonomously [12] and conditioning on the direction it needs to turn at an intersection. However, in the context of this work, it is more informative to compare the works in terms of the architectures they leverage to condition their networks. Figure 2 presents a summary of the common approaches found in literature.

The conceptually simplest approach to conditioning neural networks is shown in Fig. 2 (a) where the conditioning signal $\boldsymbol{\zeta}$ is simply appended to the policy observation $\boldsymbol{o}$ [16]. As such approach is only possible with continuous scalar/vector inputs, many works include an encoding network [9], [13], [17]–[19], which generates a numeric representation of the conditioning signal. Such encoders can be represented by fully connected layers [17], transformers [9], recurrent neural networks [8] for natural language conditioning signals, or convolutional networks for image-based conditioning [13].

The multihead (b) and multipolicy (c) architectures shown in Fig. 2 are very similar. In a multihead network all heads operate on the same latent representation produced by a shared encoder π_{shared}. A subsequent multiplexer then selects one of the heads based on the current task signal. This architecture has been applied successfully to a real-world remote-controlled car, which can either turn left, go straight, or turn right at intersections based on the conditioning signal $\boldsymbol{\zeta}$ [12]. In this form the approach can only be applied when the task-space is discrete and a head for each discrete task-space class exists, e.g. three heads are required for a controller that enables the car to go left, straight, right. The multi-policy approach with a subsequent interpolation layer presented in [15], [20] is very similar. However, no shared encoder is used and the multiplexer is replaced by an interpolation module. The latter enables this approach to handle continuous task signals as the individual control actions by the respective policies are combined smoothly.

A novel approach to conditioning a network that does not require training separate heads nor performs naive input feature concatenation presented in [8]. Their proposed feature-wise linear modulation (FiLM) layer is illustrated in Fig. 2 d). The idea is that a FiLM layer is inserted between two layers of an existing network, effectively splitting the original network into two parts π_1 and π_2. The activations of the first part π_1 are passed through the FiLM layer which applies an affine transform with trainable parameters γ_1 and γ_2. The transformed activations are then used as input to π_2 which generates the final control action. This architecture was originally devised for transforming feature maps of convolutional neural networks [8] but has been applied to robotic manipulation tasks [21], optimal information encoding, and style transfer tasks [14]. As an extension, we also propose an augmented FiLM architecture Fig. 2 e) which also feeds the conditioning signal into the control policy directly.

III. Methodology

In this work we will compare and evaluate the different architectures shown in Fig. 2 for the task of conditioning a quadrotor control on user input. Focusing on the challenging task of agile quadrotor flight, policies are trained using model-free reinforcement learning and directly map a set of observations $\boldsymbol{o}_t$ to low-level control actions $\boldsymbol{a}_t$ [22]. This section first presents a brief overview of the quadrotor simulator used for training, then proceeds to explain the neural controller, and concludes by introducing the demonstrators evaluated in this work.

A. Notation & Quadrotor Dynamics

Throughout this paper, scalars are denoted in non-bold $[s, S]$, vectors in lowercase bold $\boldsymbol{v}$, and matrices in uppercase bold $\boldsymbol{M}$. World $\mathcal{W}$ and Body $\mathcal{B}$ frames are defined with an orthonormal basis i.e. $\{\boldsymbol{x}_\mathcal{W}, \boldsymbol{y}_\mathcal{W}, \boldsymbol{z}_\mathcal{W}\}$. The frame $\mathcal{B}$ is located at the center of mass of the quadrotor.

The quadrotor is assumed to be a 6 degree-of-freedom rigid body of mass m and diagonal moment of inertia matrix $\boldsymbol{J} = \mathrm{diag}(J_x, J_y, J_z)$. Furthermore, the rotational speeds of the four propellers Ω_i are modeled as a first-order system with time constant k_mot where the commanded motor speeds $\boldsymbol{\Omega}_\text{cmd}$ are the input. The state space is thus 17-dimensional and its dynamics can be written as:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{p}}_{\mathcal{WB}} \\ \dot{\boldsymbol{q}}_{\mathcal{WB}} \\ \dot{\boldsymbol{v}}_\mathcal{W} \\ \dot{\boldsymbol{\omega}}_\mathcal{B} \\ \dot{\boldsymbol{\Omega}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_\mathcal{W} \\ \boldsymbol{q}_{\mathcal{WB}} \cdot \begin{bmatrix} 0 & \boldsymbol{\omega}_\mathcal{B}/2 \end{bmatrix}^\top \\ \frac{1}{m}\left(\boldsymbol{q}_{\mathcal{WB}} \odot (\boldsymbol{f}_\text{prop} + \boldsymbol{f}_\text{res})\right) + \boldsymbol{g}_\mathcal{W} \\ \boldsymbol{J}^{-1}\left(\boldsymbol{\tau}_\text{prop} + \boldsymbol{\tau}_\text{res} - \boldsymbol{\omega}_\mathcal{B} \times \boldsymbol{J}\boldsymbol{\omega}_\mathcal{B}\right) \\ \frac{1}{k_\text{mot}}\left(\boldsymbol{\Omega}_\text{cmd} - \boldsymbol{\Omega}\right) \end{bmatrix}, \quad (1)$$

where $\boldsymbol{g}_\mathcal{W} = [0, 0, -9.81\,\mathrm{ms}^{-2}]^\top$ denotes earth's gravity, $\boldsymbol{f}_\text{prop}$, $\boldsymbol{\tau}_\text{prop}$ are the collective force and the torque produced by the propellers. To account for residual aerodynamic effects, we introduce a lumped residual term $\boldsymbol{f}_\text{res}$, $\boldsymbol{\tau}_\text{res}$ on the forces and torques respectively.

Model-free reinforcement learning suffers from a low sample-efficiency during training which necessitates an efficient simulator that can run fast. Hence, to model the thrust and torque produced by the i-th propeller, the commonly used and computationally lightweight quadratic model [23]–[25] is employed:

$$\boldsymbol{f}_i(\Omega) = \begin{bmatrix} 0 & 0 & c_\text{l}\,\Omega^2 \end{bmatrix}^\top \quad \boldsymbol{\tau}_i(\Omega) = \begin{bmatrix} 0 & 0 & c_\text{d}\,\Omega^2 \end{bmatrix}^\top \quad (2)$$

$$\boldsymbol{f}_\text{prop} = \sum_i \boldsymbol{f}_i \qquad \boldsymbol{\tau}_\text{prop} = \sum_i \boldsymbol{\tau}_i + \boldsymbol{r}_{\text{P},i} \times \boldsymbol{f}_i \quad (3)$$

where c_l, c_d denote the lift and drag coefficient of propeller respectively and $\boldsymbol{r}_\text{P}$. Compared to state-of-the-art methods that leverage blade-element-momentum theory [26], this quadratic model does not account for aerodynamic effects, such as rotor drag or blade flapping. This deficiency widens the sim-to-real gap when deploying the trained controller in the real-world. To increase the simulation fidelity while maintaining low computational complexity we use a polynomial graybox model [26], [27] for the residual force $\boldsymbol{f}_\text{res}$ and torque $\boldsymbol{\tau}_\text{res}$ term.

B. Neural Controller

In this work, the task of fast and agile quadrotor flight is defined as navigating through a sequence of drone racing gates as fast as possible. Or, to rephrase this using broader terms: Navigate through a sequence of predefined waypoints g_i in minimum time and pass each waypoint within an l_∞ distance less then the dimension of a racing gate. To accomplish this, the control policy directly maps an observation $\boldsymbol{o}_t$ and a conditioning input ζ_t to an action (control command) $\boldsymbol{a}_t$. The control policies are trained using model-free reinforcement learning (PPO [28]) purely in simulation.

1) Observation and Action Space: At each timestep t the policy has access to an observation $\boldsymbol{o}_t$ from the environment which contains (i) the current robot state, (ii) the relative position to the next waypoint to be passed, and (iii) the current conditioning signal. Specifically, the state consists of the vehicle position $\boldsymbol{p}_{\mathcal{WB}}$, its velocity in body-frame $\boldsymbol{v}_\mathcal{B}$ and its attitude. To avoid discontinuities the latter is represented by a rotation matrix instead of directly using the quaternion $\boldsymbol{q}_{\mathcal{WB}}$ [29]. The value network used during training time has access to the same input features as the policy network. In contrast to the policy network, the value network architecture does not contain any FiLM layers.

The control command $\boldsymbol{a}_t$ consists of a desired collective mass-normalized thrust c and a bodyrate setpoint $\boldsymbol{\omega}_{\mathcal{B},\text{ref}}$. Those commands are then tracked by a low-level controller, which finally controls the motors. In contrast to more abstract control modalities such as linear velocity references, operating on collective thrust and bodyrates has been shown to be well suited for agile learned quadrotor control [22].

2) Conditioning: We compare and evaluate different network architectures (illustrated in Fig. 2) to condition a neural controller for agile quadrotor flight. Specifically, the following architectures are considered:

- *Naive-c* a naive baseline (see Fig. 2 a)) where continuous scalar conditioning signal is concatenated with the observation,
- *Naive-d* the same architecture as *naive-c* but with a discretized one-hot vector encoding of the conditioning signal,
- *Multihead* an architecture (see Fig. 2 b)) with a discrete conditioning signal similar to [12],
- *FiLM-c* a standard FiLM architecture (see Fig. 2 d)) with a continuous scalar conditioning input,
- *FiLM*-c* our augmented FiLM architecture (see Fig. 2 e)) with a continuous scalar conditioning input,
- *FiLM*-d* the same architecture as *FiLM*-c* but with a discretized one-hot vector encoding of the conditioning signal.

3) Reward Function: We use a dense shaped reward to encode the task of high-speed flight through a set of predefined waypoints. The reward r_t at time step t is given by

$$r_t = r_t^\text{prog} + r_t^\text{perc}(\zeta) - r_t^\text{twr}(\zeta) - r_t^\text{crash}\,, \quad (4)$$

where r^prog rewards progress towards the next gate to be passed [5], $r^\text{perc}(\zeta)$ encodes perception awareness by adjusting the vehicle's attitude such that the optical axis of its

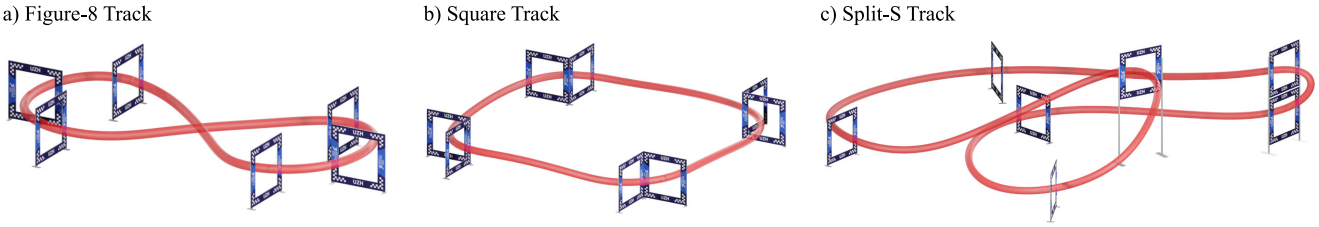


Fig. 3. We evaluate neural policy conditioning on the task of autonomous drone racing. Different approaches for policy conditioning are evaluated on a set of three different race tracks of varying complexity. All tracks are of similar size, spanning between 10 m and 16 m in width.

camera points towards the next gate's center with an optional user-specified offset, $r^{\text{twr}}(\zeta)$ is a penalty for violating the user-specified maximum thrust-to-weight ratio, and r^{crash} is a binary penalty that is only active when colliding with a gate or when the platform leaves a pre-defined bounding box, which also ends the episode.

Progress, perception, thrust-to-weight, and collision reward components are formulated as follows:

$$\begin{aligned} r_t^{\text{prog}} &= \lambda_1 \left(d_{\text{Gate}}(t-1) - d_{\text{Gate}}(t) \right) \\ r_t^{\text{perc}}(\zeta) &= \lambda_2 \exp\left(\lambda_3 \cdot \delta_{\text{cam}}(\zeta)^4 \right) \\ r_t^{\text{twr}}(\zeta) &= \max(\lambda_4 \exp\left(\lambda_5 (c_{\text{cmd}} - c_{\text{twr}}(\zeta)) \,/\, c_{\text{max}} \right) - 1, 0) \\ r_t^{\text{crash}} &= \begin{cases} -5.0, & \text{if } p_z < 0 \text{ or in collision with gate.} \\ 0, & \text{otherwise} \end{cases}, \end{aligned} \tag{5}$$

where $d_{\text{Gate}}(t)$ denotes the distance from the quadrotor's center of mass to the center of the next gate, $\delta_{\text{cam}}(\zeta)$ is the angle between the optical axis of the camera and the user-specified viewing direction (center of the next gate + offset angle). The parameters c_{cmd}, $c_{\text{twr}}(\zeta)$ and c_{max} are the commanded mass normalized thrust, the current user-specified maximum allowable mass normalized thrust and the maximum mass normalized thrust physically available for the quadrotor, respectively. The hyperparameters $\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5$ trade-off objectives regarding perception awareness and thrust-to-weight ratio constraints against progress objectives.

4) Policy Training: All control policies are trained using Proximal Policy Optimization (PPO) [28]. PPO has been shown to achieve state-of-the-art performance on a set of continuous control tasks and is well suited for learning problems where interaction with the environment is fast. Data collection is performed by simulating 100 agents in parallel using TensorFlow Agents [30]. At each environment reset, every agent is initialized in a random gate on the track layout with bounded perturbation around a state previously observed when passing the respective gate.

IV. Experiments

Using the training methodology described in the previous section, our experiments aim to answer the following research questions: (i) Which of the architectures (Naive, Multihead, FiLM, our augmented FiLM) presented in the previous section (III-B.2) is best suited for conditioning mobile robot control policies? (ii) Is it better to use a discrete or continuous conditioning signal? (iii) What role does the size of the network play? (iv) Do the results transfer to a real-world quadrotor platform?

TABLE I

Physical parameters of the quadrotor.

Parameter	Unit	Value
Mass	kg	0.807
Thrust	N	36
TWR	–	4.5
Inertia	g m^2	$I_{xx} = 2.5, \quad I_{yy} = 2.1, \quad I_{zz} = 4.3$

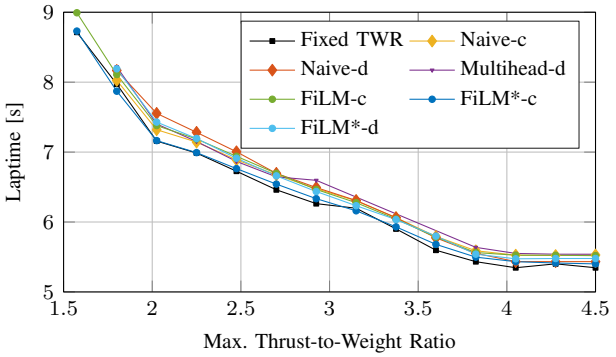
A. Experimental Setup

As an example for a mobile robot, this work uses the agile quadrotor platform shown in Fig. 1 with specifications listed in Table I. The evaluated control policies are all trained for the task of autonomous drone racing. In contrast to prior work tackling autonomous racing, our experiments focus on the racing performance when conditioned on a high-level user input. The user inputs evaluated in this work include (i) constraints on the maximum agility while racing and (ii) a user-defined perception objective. While the former conditioning signal aims at steering the aggressiveness of the racing strategy and allows to trade off between speed and safety, the latter enables to alter the robot's heading direction during flight, which could be used to focus perception on salient landmarks on the track or keep an opponent in the field of view during a race.

Our study is performed on the three track layouts shown in Fig. 3 - a square track, a figure-8 track and a complex three-dimensional track layout [4] called *Split-S* track, due to the maneuver required to pass the double gate on the far right. Throughout all experiments, performance is measured by comparing the achieved laptimes and perception awareness of the deployed policies. Perception awareness is quantified by evaluating the average angular error between the desired and observed camera viewing direction.

B. Choice of Network Architecture

In a first set of simulation experiments we aim at identifying the best network architecture for efficient neural policy conditioning in autonomous drone racing. To this end, we focus on the most complex track layout *Split-S* and measure the achieved laptime when conditioned on a maximum agility level. Specifically, the policies for all architectures are conditioned on the available thrust-to-weight ratio (TWR), ranging from 1.6 TWR to 4.5 TWR. To have an estimate of the lower bound of the laptime, we also train so-called *fixed-TWR* policies. These policies are trained for a single TWR setting, allowing them to overfit for a specific agility level, which typically results in faster training progress and superior performance.



Architecture	Avg. Rel. Laptime [%]	Max. Rel. Laptime [%]
Naive-c	2.63	3.52
Naive-d	3.25	5.98
Multihead-d	3.23	4.23
FiLM-c	2.80	3.64
FiLM*-c	**0.54**	**1.62**
FiLM*-d	3.82	4.69

Fig. 4. All architectures are able to condition a quadrotor control policy for agile flight on the maximum thrust-to-weight ratio. Our augmented *FiLM*-c* architecture even manages to be within 0.6 % of a fixed-TWR baseline, indicating that one does not have to trade-off control performance for the added flexibility to regulate the controller during deployment. Furthermore, the *FiLM*-based architectures cover the whole TWR-range and, unlike the *Naive* baseline, do not crash at the lowest TWR setting.

Figure 4 shows the results of this experiment. The fixed-speed reference is trained for and evaluated at 14 evenly spaced points throughout the TWR interval $[1.6,\ 4.5]$. Each of the conditioned policies are then evaluated at these thrust-to-weight ratio setpoints. To reduce the stochasticity of the results, each policy is the best of three identical policies trained with different initial random weights.

All the architectures we evaluated result in control policies that are able to race at a wide range of thrust-to-weight ratios. However, upon a closer look one can see that the *FiLM*-c* policy leveraging our augmented FiLM architecture outperforms the other approaches in terms of laptime. More importantly, the *FiLM*-c* is less than 0.6 % slower on average than a set of specifically trained fixed-TWR policies. We therefore gain the flexibility to regulate the neural controller during deployment while paying almost no penalty in terms of the optimality (i.e. laptime) of the control policy. Furthermore, at the lowest thrust-to-weight setting of 1.6 TWR only the policy trained with the *FiLM-c* and the *FiLM*-c* architectures are able to race collision-free through the track. All other policies do not complete a single lap as they crash into the ground at some point. This further highlights the superior versatility of the FiLM architecture, as it is able to cover a wider range of conditioning inputs compared to the other architectures. When comparing policies that operate on continuous inputs to policies trained using a one-hot encoding, we find that the continuous encoding outperforms its discrete counterpart both for the FiLM architecture as well as the naive architecture.

Based on the results presented above, we conclude that the *FiLM*-c* framework outperforms the other approaches and is extremely close to a fixed-TWR reference policy. We thus use this architecture in all subsequent experiments.

TABLE II
Average relative laptimes of a *FiLM*-c* policy achieved when trained with different sizes of the policy- and value-network.

Network Size	Avg. Rel. Laptime [%]	Max. Rel. Laptime [%]
64	5.14	13.26
128	**2.09**	4.97
256	2.22	**3.2**
512	3.87	4.76

C. Network Size

We ablate the impact of changes in the network size on the performance of the *FiLM*-c* policy. Both the policy-network and the value-network are implemented as two-layer MLPs and we vary their sizes together, such that the value network has four times wider layers. As in the previous experiments, all trained policies are conditioned on a maximum thrust-to-weight ratio while racing through the *Split-S* track layout.

Consistent with the previous experiments, all policies are trained on a thrust-to-weight ratio interval of 1.6 to 4.5 for a fixed number of environment interactions. For each setting, three policies are trained to reduce the variance and all numbers are averages across those three policies. Table II shows the average relative laptimes achieved by a *FiLM*-c* controller with the different network sizes. One can see that a too small network is not expressive enough while larger network sizes become increasingly difficult to train in an RL-setting, indicated by the increased laptime. Based on these results, we chose a *FiLM*-c* architecture where the policy/value network have 128/512 neurons per layer.

D. Different Track Layouts

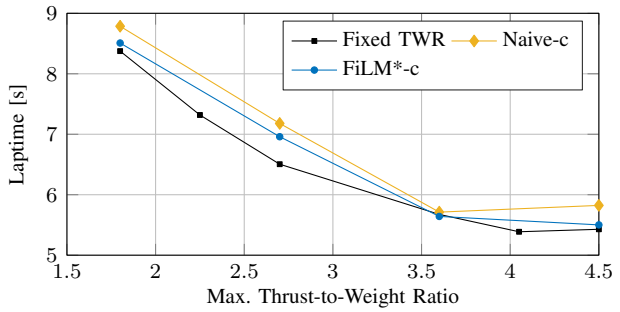
After discussing the choice of network architecture, we now study how the selected architecture performs on the different track layouts introduced above (see Fig. 3). We again consider the task of conditioning the policy on the maximally available thrust-to-weight ratio and summarize the results in Table III. The results obtained for the *Split-S* and the *Figure-8* track are very similar and verify that the *FiLM*-c* architecture works on a variety of track layouts. On the *Square* track, we obtain a surprising result: the *FiLM*-c* architecture consistently outperforms the fixed-TWR reference. This result indicates that the policy is able to combine experience gained at different TWR settings and finds a general policy that is strictly better in terms of laptime than the individually trained fixed-TWR policies.

E. Real-World Experiments

The ablation studies presented in the previous sections were all conducted in simulation. In this section we present the transferability of our approach to real-world experiments conducted on an agile quadrotor platform. The platform used for these experiments is shown in 1 and it matches our simulated quadrotor in specifications (see I). We encourage

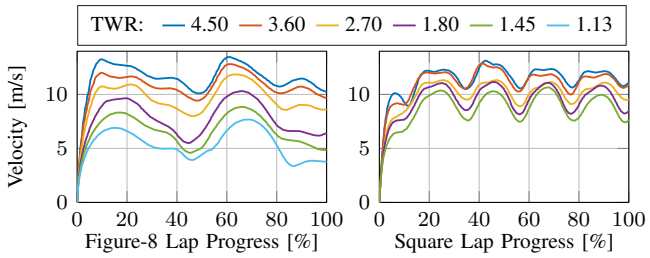
TABLE III
Comparison of the fixed of relative laptimes (w.r.t fixed-TWR reference)

	Avg. Rel. Laptime [%]	Max. Rel. Laptime [%]
Square Track	-4.60	-3.39
Figure-8 Track	0.50	3.14
Split-S Track	0.54	1.62



Architecture	Avg. Rel. Laptime [%]	Max. Rel. Laptime [%]
Naive-c	4.52	10.43
FiLM*-c	**1.91**	**6.98**

Fig. 5. The plot and table compare the laptimes achieved by the *Naive-c* and *FiLM*-c* approach in real-world experiments on the *Split-S* track. Similar to the simulation results, the *FiLM*-c* architecture outperforms the naive baseline.



TWR	Figure-8 Laptime [s]	Square Laptime [s]
4.50	2.93	3.22
3.60	3.10	3.26
2.70	3.50	3.27
1.80	4.59	4.31
1.45	5.44	–
1.13	6.52	–

Fig. 6. The plot and table compare for each track how a *FiLM*-c* policy performs in terms of achieved speeds and laptime. Especially on the *Figure-8* track the policy manages to fly the quadrotor with as little thrust margin (w.r.t hover) as 13% up to 350%.

the reader to watch the supplementary video to understand the dynamic nature of these experiments.

In a first set of experiments we repeat a subset of the experiments presented in IV-B and compare the *FiLM*-c* architecture with both fixed thrust-to-weight ratio policies and the *Naive-c* network (see Fig. 5). Similar to what we observed in simulation, the conditioning with the *FiLM*-c* works well and it outperforms the naive baseline in terms of laptime while being within 2 % of the fixed-TWR reference.

We also evaluate the chosen *FiLM*-c* conditioning approach on the two other tracks and show the flown trajectories for various user-defined thrust-to-weight ratios in Fig. 6. On the simple *Figure-8* track, the *FiLM*-c* policy can handle thrust-to-weight ratios as low as 1.13. The two plots in Fig. 6 illustrate the observed speeds. From the speed-plots one can also intuitively understand why the conditioning approach is very successful: when the speed is plotted over the lap-progress the curves for all speed levels exhibit very similar features. Thus the implicit assumption behind the FiLM framework—that the tasks are similar and related via some continuous transformation—holds.

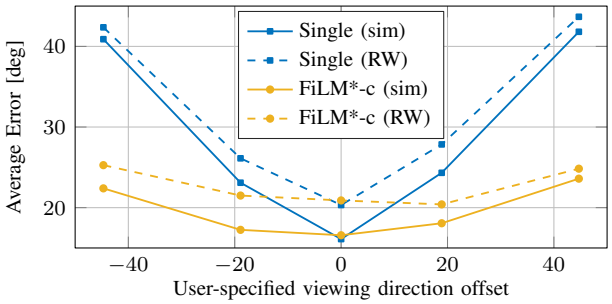


Fig. 7. The *FiLM*-c* architecture also generalizes to the task of conditioning a policy on the viewing direction (*Split-S* track). A policy that is only trained for the single task to look at the next waypoint (Single) performs much worse than a policy that can be conditioned on the desired viewing offset (*FiLM*-c*). This results holds both in simulation (sim) and real-world experiments (RW).

F. Viewing Direction

To demonstrate the generalizability of our proposed method, we extend the experimental evaluation with an additional demonstrator for policy conditioning: perception. Specifically, we condition the *viewing direction* of the quadrotor while racing through the same track layouts as in the previous experiments. The ability to actively control perception is extremely useful for a vision-based robot, as it allows to maintain visibility with visual landmarks and as a result can substantially improve performance of state estimation. We analyze conditioning on the viewing direction both in simulation and on a real-world robot. To this end, we task the quadrotor to race through the *Split-S* track, while maintaining a user-specified heading direction relative to the next gate to be passed.

Fig. 7 shows the results of this experiment. Both in simulation and the real world, the proposed FiLM*-c approach maintains low heading errors over the entire spectrum of desired viewing directions. In contrast, policies that are trained for a single heading direction can not react to such user input and exhibit large errors for desired viewing directions different from zero.

V. Conclusion

This work presented a method to condition learning-based control policies for agile quadrotor flight on an auxiliary input. We evaluated different network architectures that process such user input through simple concatenation, multiple action heads, or by leveraging FiLM layers on the intermediate activations. In an extensive ablation study, in simulation we compared the individual approaches by conditioning control policies on the maximally available thrust-to-weight ratio. Our augmented FiLM architecture achieved the best performance and is less than 0.6 % (in simulation) or 2 % (in the real world) slower than a set of policies trained specifically for one thrust-to-weight ratio. When conditioning on the viewing direction offset w.r.t to the next landmark, there was no visible difference in laptime. These findings implicate that we gain the additional flexibility to regulate a neural network controller and do not have to trade-off control performance. Therefore, we believe that this work is an important step in making neural controllers more accessible and safe to deploy for mobile robots.

REFERENCES

[1] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science robotics*, vol. 5, no. 47, 2020.

[2] M. Tranzatto, T. Miki, M. Dharmadhikari, L. Bernreiter, M. Kulkarni, F. Mascarich, O. Andersson, S. Khattak, M. Hutter, R. Siegwart, and K. Alexis, "Cerberus in the darpa subterranean challenge," *Science Robotics*, vol. 7, no. 66, p. eabp9742, 2022.

[3] P. Foehn, D. Brescianini, E. Kaufmann, T. Cieslewski, M. Gehrig, M. Muglikar, and D. Scaramuzza, "Alphapilot: Autonomous drone racing," *Auton. Robots*, vol. 46, no. 1, p. 307–320, 2022.

[4] E. Ackerman, "Autonomous Drones Challenge Human Champions in First "Fair" Race," *IEEE Spectrum*.

[5] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2021.

[6] P. R. Wurman, S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T. J. Walsh, R. Capobianco, A. Devlic, F. Eckert, F. Fuchs, L. Gilpin, P. Khandelwal, V. Kompella, H. Lin, P. MacAlpine, D. Oller, T. Seno, C. Sherstan, M. D. Thomure, H. Aghabozorgi, L. Barrett, R. Douglas, D. Whitehead, P. Dürr, P. Stone, M. Spranger, and H. Kitano, "Outracing champion gran turismo drivers with deep reinforcement learning," *Nature*, vol. 602, no. 7896, pp. 223–228, 2022.

[7] A. Romero, S. Sun, P. Foehn, and D. Scaramuzza, "Model predictive contouring control for time-optimal quadrotor flight," *IEEE Transactions on Robotics*, pp. 1–17, 2022.

[8] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. Courville, "Film: Visual reasoning with a general conditioning layer," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[9] O. Mees, L. Hermann, and W. Burgard, "What matters in language conditioned robotic imitation learning over unstructured data," *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 4, pp. 11205–11212, 2022.

[10] C. Lynch and P. Sermanet, "Language conditioned imitation learning over unstructured data," *RSS: Robotics, Science, and Systems*, 2021.

[11] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-maron, M. Giménez, Y. Sulsky, J. Kay, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar, and N. de Freitas, "A generalist agent," *Transactions on Machine Learning Research*, 2022. Featured Certification.

[12] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 4693–4700, IEEE, 2018.

[13] V. Blukis, D. Misra, R. A. Knepper, and Y. Artzi, "Mapping navigation instructions to continuous control actions with position-visitation prediction," in *Proceedings of The 2nd Conference on Robot Learning* (A. Billard, A. Dragan, J. Peters, and J. Morimoto, eds.), vol. 87 of *Proceedings of Machine Learning Research*, pp. 505–518, PMLR, 2018.

[14] A. Dosovitskiy and J. Djolonga, "You only train once: Loss-conditional training of deep networks," in *International Conference on Learning Representations*, 2020.

[15] B. C. Da Silva, G. Konidaris, and A. G. Barto, "Learning parameterized skills," in *Proceedings of the 29th International Conference on International Conference on Machine Learning*, ICML'12, (Madison, WI, USA), p. 1443–1450, Omnipress, 2012.

[16] M. P. Deisenroth, P. Englert, J. Peters, and D. Fox, "Multi-task policy search for robotics," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3876–3881, 2014.

[17] A. Dosovitskiy and V. Koltun, "Learning to act by predicting the future," in *International Conference on Learning Representations*, 2017.

[18] DeepMind Interactive Agents Team, J. Abramson, A. Ahuja, A. Brussee, F. Carnevale, M. Cassin, F. Fischer, P. Georgiev, A. Goldin, M. Gupta, T. Harley, F. Hill, P. C. Humphreys, A. Hung, J. Landon, T. Lillicrap, H. Merzic, A. Muldal, A. Santoro, G. Scully, T. von Glehn, G. Wayne, N. Wong, C. Yan, and R. Zhu, "Creating multimodal nteractive agents with imitation and self-supervised learning," 2021.

[19] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, and S. Levine, "Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, p. 3758–3765, IEEE Press, 2018.

[20] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 1312–1320, PMLR, 2015.

[21] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn, "BC-z: Zero-shot task generalization with robotic imitation learning," in *5th Annual Conference on Robot Learning*, 2021.

[22] E. Kaufmann, L. Bauersfeld, and D. Scaramuzza, "A benchmark comparison of learned control policies for agile quadrotor flight," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022.

[23] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robot.*, Springer, 2018.

[24] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "Rotors—a modular gazebo mav simulator framework," in *Robot Operating System (ROS)*, Springer, 2016.

[25] P. Foehn, E. Kaufmann, A. Romero, R. Penicka, S. Sun, L. Bauersfeld, T. Laengle, G. Cioffi, Y. Song, A. Loquercio, and D. Scaramuzza, "Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight," *Science Robotics*, vol. 7, no. 67, 2022.

[26] L. Bauersfeld, E. Kaufmann, P. Foehn, S. Sun, and D. Scaramuzza, "Neurobem: Hybrid aerodynamic quadrotor model," in *Proceedings of Robotics: Science and Systems*, 2021.

[27] S. Sun, C. C. de Visser, and Q. Chu, "Quadrotor gray-box model identification from high-speed flight data," *Journal of Aircraft*, vol. 56, no. 2, pp. 645–661, 2019.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv e-prints*, 2017.

[29] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2019.

[30] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke, and E. Brevdo, "TF-Agents: A library for reinforcement learning in tensorflow." `https://github.com/tensorflow/agents`, 2018. [Online; accessed 25-June-2019].