Aggressive Perception-Aware Navigation Using Deep Optical Flow Dynamics and PixelMPC

Keuntaek Lee, Jason Gibson, and Evangelos A. Theodorou

***Abstract*—Recently, vision-based control has gained traction by leveraging the power of machine learning. In this work, we couple a model predictive control (MPC) framework to a visual pipeline. We introduce deep optical flow (DOF) dynamics, which is a combination of optical flow and robot dynamics. Using the DOF dynamics, MPC explicitly incorporates the predicted movement of relevant pixels into the planned trajectory of a robot. Our implementation of DOF is memory-efficient, data-efficient, and computationally cheap so that it can be computed in real-time for use in an MPC framework. The suggested Pixel Model Predictive Control (PixelMPC) algorithm controls the robot to accomplish a high-speed racing task while maintaining visibility of the important features (gates). This improves the reliability of vision-based estimators for localization and can eventually lead to safe autonomous flight. The proposed algorithm is tested in a photorealistic simulation with a high-speed drone racing task.**

***Index Terms*—Model learning for control, optimization and optimal control, visual servoing, visual tracking, visual-based navigation.**

I. Introduction

WE INTRODUCE a novel mechanism which combines vision into a model predictive control (MPC) framework. Deep learning (DL)-based perceptual control using end-to-end imitation learning has shown great success in many robotics disciplines including autonomous driving [1], manipulation [2], and autonomous drone flying [3], [4].

In this letter, instead of taking a fully end-to-end approach ([3], [4]), we deploy the power of DL in novel system modeling. In a traditional (not end-to-end) navigation, DL-aided vision pipeline played a big role in detecting objects and obstacles as a perception module and sometimes as a part of state estimation (e.g. VSLAM [5]). A controller then performed its task of navigation, avoidance, or tracking using the information provided from the vision part [6].

The visual object tracking or visual servoing technologies have been developed over the past few decades and can be found in some commercial drone products. However, most of the work in literature [3], [4], [7] are all based on reactive controllers; the robot turns left if the object is on the right-side of a robot's view, and vice versa. This reactive visual servoing requires the drone to fly at a slow speed or hover until it finishes servoing. Here we propose a predictive visual tracking controller for high-speed racing with a data-driven optical flow dynamics model composed of optical flow and robot dynamics.

In a drone racing scenario, the optical flow mostly comes from a moving camera and a static environment. Since the controller moves the robot through space, the changes in scene, the optical flow, can be thought of as indirect dynamics.

Recently, there has been a lot of progress in DL-based optical flow techniques [8]–[10]. However, all prior work relies on large convolutional neural networks with a lot of parameters to estimate the optical flow of the entire image. In our work, the application of the optical flow is to predict the relative motion of a 'single' pixel, so we use a small fully-connected feedforward network.

The main problem we address in this letter is the visibility/field of view of a moving camera, especially when it comes to high-speed racing. The more the robot observes through a camera, the more information we use to perform accurate state estimation and navigation. Therefore, it is important to control the robot to see more information, for example, by pitching up or rolling/yawing. However, this conflicts with the high-speed flying task for a drone because a quadrotor needs to pitch down to fly at a high speed and this results in losing more visual information.

To solve the problem of limitation in the field of view by visual servoing, [11], [12] proposed a Sequential Quadratic Programming-based approach where the visibility is formulated in hard constraints. However, these methods do not fit into our problem formulation which requires real-time planning and control.

In the visual servoing literature, to the best of our knowledge, the real-time predictive controllers used for a visual tracking task are [13], [14]. Although [13] formulated an MPC problem for a viewpoint optimization, the goal of the paper was controlling the drone to stabilize a gimbal to get a good quality of a video. In [14], the most relevant work to us, the authors derived the target pixel velocity based on the information of the relative 3D position (x, y, z) of the target and the robot. With the pixel velocity information, the authors were able to form an MPC problem along with vision and perform visual object tracking control in a predictive way.

However, in our work, we implement a data-driven deep-learning approach, that does not require any prior information of camera intrinsics, extrinsics, or the 3D global position of the target. Instead, our algorithm requires an object detector that detects a target in image space. Thanks to the great success

Manuscript received September 10, 2019; accepted January 6, 2020. Date of publication January 13, 2020; date of current version January 30, 2020. This letter was recommended for publication by Associate Editor Dr. F. Tombari and Editor E. Marchand upon evaluation of the reviewers' comments. This work was supported by NASA. *(Corresponding author: Keuntaek Lee.)*

The authors are with the Autonomous Control and Decision Systems Laboratory, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: keuntaek.lee@gatech.edu; jgibson37@gatech.edu; evangelos.theodorou@ae.gatech.edu).

This article has supplementary downloadable material available at https://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/LRA.2020.2965911

in the field of computer vision, we can use real-time object detectors [15], [16] with GPUs. Although our method requires prior knowledge of the image of the targets (gates) and a trained detector, we believe this is less restrictive than full knowledge of the global 3D position of features like in [14]. Furthermore, we believe our case is less restrictive since our proposed approach can be used for any moving target objects located anywhere in the scene.

In summary, the contributions of this work are twofold:

- We introduce data-driven Deep Optical Flow (DOF) dynamics, learned from the optical flow of consecutive images and robot dynamics. DOF dynamics are efficient in memory and computation.
- We introduce the Pixel Model Predictive Control (PixelMPC) algorithm which predicts the relative motion of pixels by actuating the robot to visually track important features (targets) while accomplishing the high-level tasks (e.g. racing or chasing). The algorithm makes the vision-based state estimation more robust as it explicitly allows the control algorithm to prioritize visual information.

II. Preliminaries

A. Model Predictive Optimal Control

Model Predictive Control (MPC)-based optimal controllers (e.g. Model Predictive Path Integral (MPPI) [17]) provide planned control trajectories given an initial state and a cost function by solving the optimal control problem. An optimal control problem whose objective is to minimize a task-specific cost function $J(\mathbf{X}, \mathbf{U})$ can be formulated as follows:

$$J(\mathbf{X}(t), \mathbf{U}(t)) = \phi(\mathbf{X}(t_f)) + \int_{t=t_0}^{t_f} l(\mathbf{X}(t), \mathbf{U}(t))dt \quad (1)$$

$$V(\mathbf{X}(t_0), t_0) = \min_{\mathbf{U}(t)} \Big[J(\mathbf{X}(t), \mathbf{U}(t)) \Big] \quad (2)$$

subject to dynamics

$$\frac{d\mathbf{X}}{dt} = F(\mathbf{X}(t), \mathbf{U}(t), t), \quad (3)$$

where $\mathbf{X} \in \mathbb{R}^n$ represents the system states, $\mathbf{U} \in \mathbb{R}^m$ represents the control, ϕ is the state cost at the final time t_f, l is the running cost, and V is the value function. By solving this local optimization problem, we get the optimal control sequences. This can be solved in a receding horizon fashion in an MPC framework and it allows us to have a real-time optimal controller with feedback.

In our work, a sampling-based receding-horizon stochastic optimization algorithm, MPPI controller [17] is used as an MPC controller. We chose MPPI for several reasons, first off being the generality of cost functions and dynamics allowed. Most variants of MPC require us to have a convex cost function and first or second-order approximations of the dynamics. MPPI has neither of these requirements. Therefore we can directly encode our task into the cost function without any modifications to the high-level objective. Second, MPPI has been shown to be highly successful at aggressive autonomous racing on ground vehicles with general cost functions and neural network dynamics [17].

For a short summary of MPPI algorithmically, it samples N trajectories by applying noise into the control channels and forward propagating the dynamics. Each sample can be rolled out in parallel, and then each corresponding trajectory and cost are combined to generate a final control vector. The optimization can be run K times to further refine the solution before executing it. The previous control solution is used as the center value of the Gaussian sampling to warm start the optimization each round.

B. Quadrotor Dynamics

We use the quadrotor dynamics model provided in the FlightGoggles simulator [18] used in this letter. The defined 10 states are $\mathbf{X}_{\text{robot}} = [\mathbf{p}; \mathbf{q}; \mathbf{v}] = [x, y, z, q_w, q_x, q_y, q_z, \dot{x}, \dot{y}, \dot{z}]^T$, where $\mathbf{p} = [x, y, z]^T$ is the world-coordinate position vector, $\mathbf{q} = [q_w, q_x, q_y, q_z]^T$ is the vehicle attitude unit quaternion vector, and $\mathbf{v} = \dot{\mathbf{p}} = [\dot{x}, \dot{y}, \dot{z}]^T$ is the world-coordinate linear velocity vector. The vehicle dynamics are given by

$$\dot{\mathbf{p}} = \mathbf{v} \quad (4)$$

$$\dot{\mathbf{v}} = \mathbf{g} + m^{-1}(\mathbf{R}_b^\omega \mathbf{f}_T + \mathbf{f}_D + \mathbf{w}_\mathbf{f}), \quad (5)$$

where $\mathbf{g}$ is the gravitational acceleration, m is the quadrotor mass, $\mathbf{R}_b^\omega$ is the rotation matrix from body to world frame, $\mathbf{f}_T$ is the total thrust, $\mathbf{f}_D$ is the aerodynamic drag, and $\mathbf{w}_\mathbf{f}$ is the stochastic force vector to capture unmodeled dynamics (e.g. vibrations and turbulance). The rotation matrix from body to world frame is

$$\mathbf{R}_b^\omega = \begin{bmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y - q_zq_w) & 2(q_xq_z + q_yq_w) \\ 2(q_xq_y + q_zq_w) & 1-2(q_x^2+q_z^2) & 2(q_yq_z - q_xq_w) \\ 2(q_xq_z - q_yq_w) & 2(q_yq_z + q_xq_w) & 1-2(q_x^2+q_y^2) \end{bmatrix}, \quad (6)$$

and the relation between quaternions and the angular rates is

$$\dot{\mathbf{q}} = \frac{1}{2} \begin{bmatrix} -q_x & -q_y & -q_z \\ q_w & -q_z & q_y \\ q_z & q_w & -q_x \\ -q_y & q_x & q_w \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}, \quad (7)$$

where the angular rates ω_x, ω_y, and ω_z are part of the control inputs we used along with the total thrust $\mathbf{f}_T$. The control $\mathbf{U}$ is $[\omega_x, \omega_y, \omega_z, \mathbf{f}_T]$. We make a small assumption here that the model immediately follows the control inputs, especially the angular rates. Indeed, the quadrotor in the FlightGoggles takes $\mathbf{U}$ as an input and the low-level PID controller controls the robot to follow the commands. Since we directly input the angular rates, we do not use the dynamics of the angular rates, described in [18] when we propagate the model in MPC.

C. Optical Flow

Optical flow estimates the instantaneous motion of objects and features in a visual scene from a sequence of ordered images. The motion comes from the relative motion between an observer and a scene. In our case, the motion comes from a moving observer (a camera attached on a robot) and a static environment. To compute the optical flow, two strict assumptions are required: 1) The brightness of any observed object point on images is constant over time, 2) In the image plane, neighborhood points move similarly with similar velocity. The first constraint can be written as:

$$I(u, v, t) = I(u + \Delta u, v + \Delta v, t + \Delta t), \quad (8)$$

where I represents the intensity of a pixel (u, v) and $\Delta u, \Delta v$ represent the displacement of the pixel position between two

Fig. 1. MPC-predicted future pixel trajectory (Green) of a target pixel, the center of a gate. PixelMPC computes the optimal control which accomplishes a racing task and drives the target pixel to the center of the image.

(a) Legend of optical flow (b) Optical flow ground truth (c) Deep optical flow (DOF) prediction

Fig. 2. Ground truth optical flow and the DOF prediction in the case of a quadrotor flying forward, pitching down. Deep optical flow provides more smooth optical flow compared to the ground truth. DOF predicts a single pixel's optical flow instead of the whole image's flow.

consecutive images observed at time t and $t + \Delta t$. This equation can be written in a form of Taylor series by assuming that the movement is small:

$$I(u + \Delta u, v + \Delta v, t + \Delta t) = I(u, v, t) + \frac{\partial I}{\partial u}\Delta u + \frac{\partial I}{\partial v}\Delta v + \frac{\partial I}{\partial t}\Delta t + H.O.T, \tag{9}$$

which results in

$$\frac{\partial I}{\partial u}\frac{\Delta u}{\Delta t} + \frac{\partial I}{\partial v}\frac{\Delta v}{\Delta t} + \frac{\partial I}{\partial t} = 0. \tag{10}$$

However, it is impossible to estimate the two unknowns $\frac{\Delta u}{\Delta t}$ and $\frac{\Delta v}{\Delta t}$, only with one equation, so all the optical flow calculation methods make additional assumptions to estimate the actual flow.

We used one of the most popular algorithms [19] to calculate the dense optical flow. The algorithm approximates each neighborhood of both frames by quadratic polynomials. The details of the algorithm can be found in [19] and the implementation of the algorithm is available in OpenCV [20]. For a better calculation of the dense optical flow, we used a sequence of downsized gray-scaled images instead of original RGB images. The parameters used for calculating optical flow with [19] were: pyr scale $= 0.5$, levels $= 10$, winsize $= 51$, iterations $= 15$, poly$_n$ $=$ 5, poly$_\sigma$ $= 1.1$. The visualization of the dense optical flow as a vector or in color can be found in Fig. 2 and the supplementary video includes the optical flow of a full run of racing.

III. Deep Optical Flow Dynamics

By taking advantage of the algorithms [19] calculating the optical flow, deep optical flow learning becomes self-supervised learning, which does not require any manual labeling. Our proposed neural network-based Deep Optical Flow (DOF) dynamics have two major selling points:

1) Computationally Efficient: DOF dynamics predict an optical flow/vector of a single pixel while most of the DL-based optical flow [8]–[10] predicts the next timestep's image of optical flow, with the same size of the input images. This allows us to have a very small network, so we can use the model in a real-time optimal controller that performs optimization within 20–50 ms. If we build a U-Net-like convolutional neural network, which predicts an image from an input image, we have to propagate the deep CNN every timestep in MPC framework to generate optical flow, which is computationally very expensive and slow. For the parameters used in the paper, our MPC algorithm samples over a million times per second.

2) Data-Efficient: Given an image, size of W $\times$ H, DOF can use W $\times$ H data points for training, whereas typical DL-based optical flow [8]–[10] only uses a single data point (an image of the whole optical flow).

DOF dynamics predict, just like typical robot dynamics models, the derivative of the states. Here, in DOF, it predicts the velocity of a pixel. DOF takes 3 components as input: pixel state (position) $\mathbf{X}_{\text{pixel}}$, control actions $\mathbf{U}$, and robot orientation $\mathbf{q}$. The pixel position means the position in (u, v) coordinate system on the image plane, where the top left corner is the origin (0, 0). Control actions command angular velocities in x, y, z frame and total thrust, which affects both robot motion/acceleration and the image stream. The main point here in the DOF input is the robot orientation part. We incorporate the orientation of the robot into the DOF dynamics because even with the same control input, the optical flow changes depending on the roll, pitch, and yaw angles of the robot.

We train the DOF dynamics with a neural network (NN) model to predict the magnitude l and the angle θ of a single optical flow/vector. By defining the state of the pixel $\mathbf{X}_{\text{pixel}} = [u, v]^T$, we can write the optical flow as

$$\dot{u} = lcos(\theta), \quad \dot{v} = lsin(\theta), \tag{11}$$

where l and θ are the optical flow vector component, predicted from the DOF. Therefore, the final DOF dynamics F_{pixel} is

$$\dot{\mathbf{X}}_{\text{pixel}} = F_{\text{pixel}}(\mathbf{q}, \mathbf{X}_{\text{pixel}}, \mathbf{U}) \tag{12}$$

$$= \text{PolarToEuler}(DOF(\mathbf{q}, \mathbf{X}_{\text{pixel}}, \mathbf{U})), \tag{13}$$

where the PolarToEuler mapping is Eq. (11).

Algorithm 1 describes the training process of DOF dynamics. In the first for-loop of Algorithm 1, the robot state $\mathbf{X}_{\text{robot}}$ can be either ground truth or estimated states. The first for-loop describes collecting training dataset of robot states, optimal control actions, images, and optical flow between two consecutive images. Then the following for-loops update the weights and biases of DOF dynamics model with respect to the mean squared error (MSE) loss between the target magnitude and the angle of optical flow and the prediction.

We normalize the pixel state into [0.0, 1.0] $\times$ [0.0, 1.0] space and do regression. This allows the original discrete image space [0, W] $\times$ [0, H] to be a continuous 2D space [0.0, 1.0] $\times$ [0.0, 1.0] and same for the pixel state space, as well.

We designed a feed-forward NN with 5 layers having [10, 128, 128, 128, 2] neurons per each, where 10 is for an input layer and 2 is for the output. The Rectified Linear Unit ($ReLU$) function, $f(x) = max(0, x)$, is used for the activation function in layers 1–4, and the output layer has a linear activation. All

Algorithm 1: Training Deep Optical Flow (DOF) Dynamics.

Input: Img_t: Observed image from onboard camera at timestep t, W, H: Image width, height, $\mathbf{X}_{\text{robot},t}$: Robot states at timestep t, $\mathbf{q}$: Robot orientation, MPC: Model predictive optimal controller, $J_{\text{robot}}(\mathbf{X}_{\text{robot}})$: Task-dependent state cost function for MPC, $f_{\text{robot}}(\mathbf{X}_{\text{robot}}, \mathbf{U})$: Robot Dynamics, $\text{OptFlow}(\text{Img}_t, \text{Img}_{t+1})$: Function calculating optical flow, $\mathbf{N}_{data}$: Number of data points for training, $\mathbf{N}_{epoch}$: Number of training epochs, $\mathbf{N}_{batch}$: Number of batches in total data, ϕ: Initial weights and biases of DOF NN, *Adam:* Stochastic optimization algorithm [21]

for $t = 1 : \mathbf{N}_{data}$ **do**
  $U_t^* \leftarrow \text{MPC}(J_{\text{robot}}(\mathbf{X}_{\text{robot},t}), f_{\text{robot}}(\mathbf{X}_{\text{robot},t}, \mathbf{U}_t))$
  $l_{t-1}, \theta_{t-1} \leftarrow \text{OptFlow}(\text{Img}_{t-1}, \text{Img}_t)$
end for
for 1: $\mathbf{N}_{epoch}$ **do**
  for 1: $\mathbf{N}_{batch}$ **do**
    $\mathcal{L} = 0$
    for 1: # of images in a batch **do**
      for $u = 1$: W **do**
        for $v = 1$: H **do**
$\hat{l}, \hat{\theta} \leftarrow DOF(\mathbf{q}, u, v, \mathbf{U})$ % per pixel
$\mathcal{L} += MSE(l(u,v), \hat{l}) + MSE(\theta(u,v), \hat{\theta})$
        end for
      end for
    end for
    $\phi \leftarrow Adam.step(\mathcal{L}, \phi)$
  end for
end for

Algorithm 2: Testing Deep Optical Flow (DOF) Dynamics.

Input: Detector: detects targets on image, $\mathbf{U}$: Control candidate, *DOF*: Trained DOF dynamics, $\mathbf{q}$: Robot orientation, Img: Observed image from an onboard camera

$u, v \leftarrow$ Detector(Img) % center of the object
$l, \theta \leftarrow DOF(\mathbf{q}, u, v, \mathbf{U})$
$\dot{u} = l \cos(\theta)$, $\dot{v} = l \sin(\theta)$

the layers are fully connected with regularization via 10% of dropouts. The motivation behind using the stated number of neurons was to achieve real-time performance with MPPI. For training the neural network, the Adam [21] optimizer was used with Tensorflow.[1]

The usage of the trained model can be found in Algorithm 2. Given a center of the object (u, v) from a Detector (e.g. YOLOv3 [15]), a trained DOF dynamics model takes the center position (u, v), robot orientation, and control action as an input. The output of the trained DOF dynamics is the magnitude l and the angle θ of a predicted optical flow of that single point (u, v). From predicted l and θ, the velocity of the single point is calculated as in Eq. (11).

We have included a comparison table, Table I, that shows the differences in runtimes of our optical flow prediction with another state-of-the-art network. However, even though our DOF dynamics approach cares about accuracy, our primary constraint was speed. Therefore, we compared our network with the fastest and smallest of state-of-the-art networks, the SpyNet [10]. We refer to Table 9 in [22] for benchmark results for optical flow. The table shows the accuracy and the runtime of the state-of-the-art approaches ([8]–[10], etc). Note that the total number of parameters in DOF dynamics NN is 34,690, whereas the SpyNet has 1,200,250 parameters. We tested DOF dynamics both with a single pixel prediction case and the whole-image prediction case (31,212 pixels). We clearly see that for multi-step prediction ($N_{\text{timestep}} = 80$), running DOF dynamics for a whole image ($N_{\text{pixel}} = 204 \times 153$) to predict the optical flow is too slow (3 Hz) and does not fit into real-time MPC algorithms. Since the SpyNet requires the input image pairs to have width and height to be multiple of 32, we resized the image to have a similar size as our training data set: $192 \times 160 = 30{,}720$ pixels. From Table I, it is apparent that the single pixel approach with our DOF dynamics can only fit into a real-time "sampling-based" MPC framework.

TABLE I
THE RUNTIME COMPARISON OF OUR DOF DYNAMICS NN AND THE STATE-OF-THE-ART WHOLE-IMAGE BASED OPTICAL FLOW PREDICTION NN, THE SPYNET [10]. N_{SAMPLE} IS THE NUMBER OF SAMPLES (BATCH) USED IN A SAMPLING-BASED CONTROLLER, MPPI, TO PROPAGATE IN PARALLEL WITH GPU AND N_{TIMESTEP} IS FOR MULTI-STEP PREDICTION (MPC), WHICH REQUIRES SEQUENTIAL COMPUTATION. OOM (OUT OF MEMORY) SHOWS THAT THE FULL-SIZE OPTICAL FLOW PREDICTION CANNOT BE RUN IN SOME CASES. THE RUNTIME WAS MEASURED WITH INTEL XEON(R) CPU E5-1650 V4 @ 3.60 GHZ X 12 CPU AND NVIDIA GEFORCE GTX 1060 6 GB

	Runtime [ms]				
	N_{pixel}	N_{sample}	N_{timestep}	$\mu \pm 2\sigma$	max
YOLOv3	204×153	1	1	14.9 ± 5.6	21.1
DOF	1×1	1	1	1.5 ± 0.4	2.1
DOF	1×1	1	80	6.7 ± 2.9	10.1
DOF	1×1	512	1	1.6 ± 0.7	2.8
DOF	**1×1**	**512**	**80**	**8.6 ± 3.0**	**11.9**
DOF	204×153	1	1	6.9 ± 1.5	9.4
DOF	204×153	1	80	327.0 ± 10.6	340.3
DOF	204×153	512	1	OOM	OOM
SpyNet	192×160	1	1	3.3 ± 0.5	4.0
SpyNet	192×160	512	1	OOM	OOM

We believe comparing the accuracy of our method and the standard full-image optical flow method is unfair because both approaches use different information to predict the optical flow. While the full-image approach uses more perceptual information, our DOF approach uses more non-visual information; the robot orientation and controls.

We report the prediction error of our DOF dynamics on the test dataset in Average Endpoint Error (AEE) of **2.45**. The endpoint error calculates the Euclidean distance between the ground truth optical flow vectors and the predicted vectors. In the optical flow literature, depending on the training dataset, the state-of-the-art methods report AEE of 0.5-10.0.

IV. PIXEL MODEL PREDICTIVE CONTROL

In this chapter, we introduce Pixel Model Predictive Control (PixelMPC) algorithm for visual object tracking and autonomous racing. PixelMPC literally predicts the future state

[1]https://www.tensorflow.org/

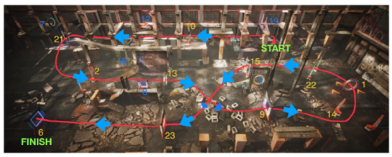

Fig. 3. The race course in the FlightGoggles [18].

trajectory of a "pixel model," the deep optical flow (DOF) dynamics, and calculates the optimal control sequence (Fig. 1).

Assuming we have a visual object detector, for example, detecting custom classes of objects using You Look Only Once (YOLO) [15] algorithm. Given some detected objects, we can predict the future trajectories of their center points/pixels $[u, v]$. For a visual tracking task, one cost function for the optimal control of the DOF dynamics can be the L1 distance between the object pixel position $[u, v]$ and the center of the image. This L1 cost function will force the pixel to be close to the center of the image O:

$$J_{\text{pixel}}(\mathbf{X}_{\text{pixel}}) = \int_{t=0}^{t_f} c_{\text{pixel}} L1([u_t, v_t], O)dt. \tag{14}$$

This cost function is reasonable for visual object tracking because the closer the target is to the center of the image, the longer we observe the target. In addition, the center of the image has the least distortion, which means the lowest information lost.

The autonomous racing task-related cost function for a finite-horizon optimal control problem can be designed in a form of Eq. (1). For example, to follow the desired position, orientation, and velocity $\mathbf{p}_d$, $\mathbf{q}_d$, and $\mathbf{v}_d$:

$$J_{\text{robot}}(\mathbf{X}_{\text{robot}}) = \int_{t=0}^{t_f} c_1 h(\mathbf{p}_d, \mathbf{p}_t)^2 + c_2(\mathbf{q}_d - \mathbf{q}_t)^2 + c_3(\mathbf{v}_d - \mathbf{v}_t)^2 dt, \tag{15}$$

which control cost is ignored and $h(\mathbf{p}_d, \mathbf{p})$ is an indicator function which returns 1,000 if a robot crashes into a gate or a value between $[-1, 1]$. A smaller return represents the robot being closer to the desired path. The ordered waypoints (gates) are assumed to be given with the map of the entire racing track (e.g. Fig. 3). Note that this information including the position of the targets is only for the racing task along with a real-time path planning, not for the visual-servoing task. If the task of PixelMPC is similar to [14], where the task is following given waypoints, then prior information of target locations is not required.

Now, the total cost function for the optimization Eq. (2) is formed as

$$J(\mathbf{X}) = J_{\text{robot}}(\mathbf{X}_{\text{robot}}) + J_{\text{pixel}}(\mathbf{X}_{\text{pixel}}), \tag{16}$$

where a new state $\mathbf{X}$ is defined as $\mathbf{X} = [\mathbf{X}_{\text{robot}}; \mathbf{X}_{\text{pixel}}] = [\mathbf{p}, \mathbf{q}, \mathbf{v}, u; v] = [x, y, z, q_w, q_x, q_y, q_z, \dot{x}, \dot{y}, \dot{z}, u, v]^T$.

The total dynamics $F(\mathbf{X}, \mathbf{U})$ used to optimize Eq. (16) can be written as a combination of two dynamics Eqs. (4)–(7) and Eq. (13). Our formulation allows us to emphasize one task over another by tuning the cost function. If we want to achieve a faster speed instead of more visibility, then we can weight it more heavily.

Algorithm 3 and Fig. 4 shows the PixelMPC algorithm. Either from a ground truth or a state estimator, we receive a new robot state and an image from a monocular camera. A detector

Algorithm 3: Pixel Model Predictive Control (PixelMPC).

Input: Detector [15]: detects targets on image,
DOF: Deep optical flow dynamics, Δt: timestep size,
Img: Observed image from an onboard camera,
Ctrl$^*(J(\mathbf{X}, \mathbf{U}))$: Optimal controller, computes $d\mathbf{U}^*$,
$J_{\text{robot}}(\mathbf{X}_{\text{robot}})$: Task-dependent robot state cost function,
$J_{\text{pixel}}(u, v)$: Task-dependent pixel state cost function,
f_{robot}: Robot dynamics, $\mathbf{U}_{0:T}$: Initial control sequence,
$\mathbf{T}$: MPC time horizon, $\mathbf{K}$: Number of optimization

while Task done **do**
  Receive a new state $\mathbf{X}_{\text{robot},0}$ and Img
  for $k = 0 : \mathbf{K}$ **do**
    $u_0, v_0 \leftarrow$ Detector(Img) % center of the object
    for $t = 0 : \mathbf{T}$ **do**
      $J_t = J_{\text{robot}}(\mathbf{X}_{\text{robot},t}) + J_{\text{pixel}}(u_t, v_t)$
      $\mathbf{X}_{\text{robot},t+1} = \mathbf{X}_{\text{robot},t} + f_{\text{robot}}(\mathbf{X}_{\text{robot},t}, \mathbf{U}_t)\Delta t$
    $l, \theta \leftarrow DOF(\mathbf{q}_t, u_t, v_t, \mathbf{U}_t)$
    $\dot{u} = l\cos(\theta)$, $\dot{v} = l\sin(\theta)$
    $u_{t+1} = u_t + \dot{u}\Delta t, v_{t+1} = v_t + \dot{v}\Delta t$
    end for
    $d\mathbf{U}^*_{0:T} \leftarrow$ Ctrl$^*(J_{0:T}(\mathbf{X}_{\text{robot},0:T}, u_{0:T}, v_{0:T}, \mathbf{U}_{0:T}))$
    $\mathbf{U}_{0:T} \leftarrow \mathbf{U}_{0:T} + d\mathbf{U}^*_{0:T}$
  end for
  Execute $\mathbf{U}_0$
  $\mathbf{U}_{0:T-1} \leftarrow \mathbf{U}_{1:T}$ % shift for a warm-start
end while

(e.g. YOLOv3 [15]) detects the center of a target (gate, in a racing scenario) (u, v) on the image space and an optimal model predictive controller solves the optimization problem with respect to the total cost J, Eq. (16), with a receding time horizon $\mathbf{T}$. After propagating the combined model dynamics and running the optimization step, we execute the first control action and use the remaining control trajectory solution for the next optimization loop as a warm start. Then again we receive a new robot state with an image and repeat the optimization at a rate of 40 Hz.

V. Experiments/Results

A. Experimental Setup

We tested our algorithm in the FlightGoggles simulation [18], which is developed for agile flight simulation with high fidelity. The racing scenario is from the AlphaPilot–Lockheed Martin AI Drone Racing Innovation Challenge[2] (Fig. 3).

To derive our DOF dynamics from optical flow data, we collected 10 rounds of autonomous flight using a nominal MPPI controller, which took around 30 seconds for each round. To fully explore the state space we varied the target speed between 6 m/s and 14 m/s across rounds. The timestep in MPPI was 0.025 seconds. In total 14,000 images from a monocular camera along with drone states and controls were collected. The images were each downsized to a size of [204, 153]. This provided $204 \times 153 = 31{,}212$ data points. As a result, around 437 million data points for training DOF were collected from 5 minutes of flying data. The states are collected from ground truth provided in the FlightGoggles simulator.

[2]https://www.herox.com/alphapilot/85-2019-virtual-qualifier-tests

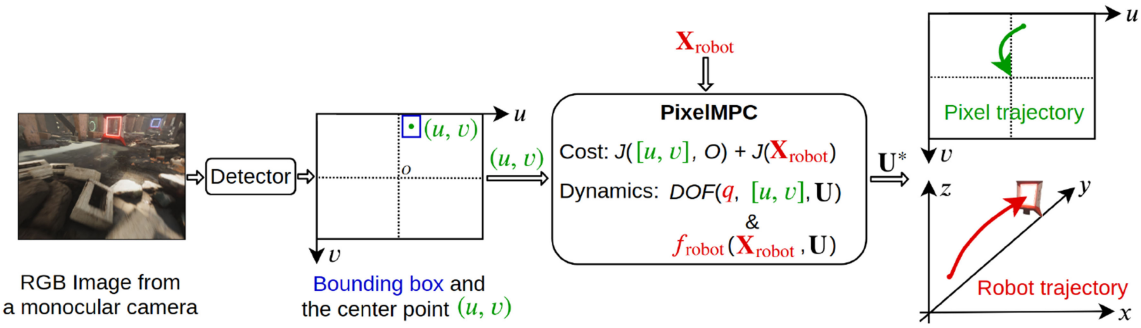

Fig. 4. A block diagram showing the algorithmic flow of PixelMPC.

TABLE II
THE TIME (sec) THE ROBOT VISUALLY LOSES THE TARGET. THE RESULTS ARE FROM 10 RACES PER EACH CASE IN $\mu \pm 2\sigma$ V.S. $[t_f\,(sec), c_{\text{PIXEL}}]$ WHEN THE TARGET SPEED WAS $14m/s$. THE TOP-LEFT $[t_f, c_{\text{PIXEL}}] = [0.0, 0.0]$ SHOWS THE NOMINAL MPPI RUNNING WITHOUT ANY DOF DYNAMICS CONTROL

Time (sec) of less than 50% visibility				
t_f \ c_{pixel}	0.0	3.0e+6	6.0e+6	9.0e+6
0.0	13.6±3.6	-	-	-
1.0	-	3.6±0.6	1.9±0.6	**1.5±0.2**
2.0	-	3.1±0.9	2.0±0.6	1.9±1.2
Time (sec) of 0% visibility				
t_f \ c_{pixel}	0.0	3.0e+6	6.0e+6	9.0e+6
0.0	3.6±1.1	-	-	-
1.0	-	1.0±0.3	1.1±0.5	**0.2±0.1**
2.0	-	0.7±0.4	0.6±0.2	**0.2±0.1**

TABLE III
LAP TIME FROM 10 RACES PER EACH CASE IN $\mu \pm 2\sigma$ (sec) V.S. $[t_f\,(sec), c_{\text{PIXEL}}]$ WHEN THE TARGET SPEED IS $14m/s$

Lap time (sec)				
t_f \ c_{pixel}	0.0	3.0e+6	6.0e+6	9.0e+6
0.0	**31.8±1.0**	-	-	-
1.0	-	32.7±0.4	33.2±0.2	33.5±0.2
2.0	-	33.2±0.1	34.2±0.3	34.0±0.5

For object detection, we used one of the state-of-the-art algorithms, YOLOv3 [15], which allows us to detect multiple objects in real-time. 3,000 downsized images were used to train the YOLOv3 model to predict 7 classes of gates in the FlightGoggles racing environment (Fig. 3).

B. Model Predictive Path Integral Control (MPPI)

For a drone racing task along with the visual object tracking task, the cost function parameters used for MPPI are $\Delta t = 0.025$ (sec), $c_1 = 400$, $c_2 = 250$, $c_3 = 8.0$, $T = 80$, and $K = 1$. The control variance had noise profiles: $\sigma_{\text{roll}} = 0.2$, $\sigma_{\text{pitch}} = 0.2$, $\sigma_{\text{yaw}} = 0.3$, and $\sigma_{\text{thrust}} = 2.2$. c_{pixel} was tuned between 3.0e + 6 and 9.0e + 6 and the resulting different behaviors are reported in Table II and Table III. The reason why the parameter c_{pixel} is chosen 4 orders of magnitude higher than all other cost function parameters is because we only normalized the pixel position term from [0, W] × [0, H] to [0.0, 1.0] × [0.0, 1.0]. A total of 512 samples were used to propagate the 12 states with a time horizon of 80 in 40 Hz, which results in a 2 second trajectory. The number of samples depends on the hardware (GPU, CPU, RAM, etc.) and the size of the DOF NN dynamics. The nominal MPPI case for the racing task used the cost function only composed of Eq. (15) and the same parameters described above were used to give a fair comparison.

Although the drone dynamics we introduced in this letter are the simplified linear dynamics from the simulator [18], any nonlinear dynamics model can also be used as robot dynamics model in PixelMPC.

C. Drone Racing With Object Tracking

We compare the visibility in percentage; how long the robot grabs the target in its view. In the PixelMPC framework, there are some additional DOF dynamics-related parameters we can tune: 1) the time horizon t_f considered for the pixel cost and 2) the cost coefficient c_{pixel}. $t_f = 1.0$ sec means the pixel cost Eq. (14) only penalizes the pixel trajectory within 1.0 second.

Table II shows the time the drone loses the target in sec ($\mu \pm 2\sigma$). We consider the 'loss' as visually losing the target after the robot first sees it. In this experiment, we used the ground truth provided by the FlightGoggles for robot states.

In the nominal case, without considering DOF dynamics in MPPI, the time the robot has less than 50% visibility of a target was. 6 sec, which is more than 42% of the total flying time (∼31.8 sec). With PixelMPC, we can decrease it to 1.5 sec, 4.5% of the flying time (∼33.5 sec). The time of robot having less than 0% visibility of a target also decreased from 3.6 sec (4% of flying time) to 0.2 sec (less than 1% of flying time). Notice that in both 0% and 50% cases, the 2σ of the lost time is very large in the nominal case, compared to the PixelMPC cases. This can be explained in Fig. 6, where the plots show how smooth the movement is when we use PixelMPC. Also, compared to the nominal MPPI, PixelMPC showed 29% decrease in linear and angular accelerations in mean, which resulted in a slower speed but it provided much smoother behavior; please see Fig. 6 (Best shown in the supplementary video). However, the smoothness behavior of PixelMPC is a byproduct of the visual target tracking, not the main goal. Also, the visual target tracking cannot be accomplished by simply applying a smoothing/filtering to a controller.

In Table III, we compare the race time for each case to see how much lap time delay we get to pay for more visual information. Table III shows the mean and the 2σ standard deviation from

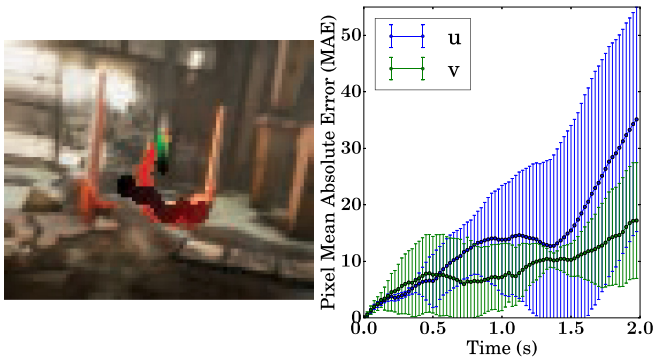

Fig. 5. *Left:* Cropped image showing PixelMPC-predicted pixel trajectory (Green) vs. Actual pixel trajectory (Red). *Right:* Mean absolute error and standard deviation of the multi-step prediction of pixel position on DOF dynamics.

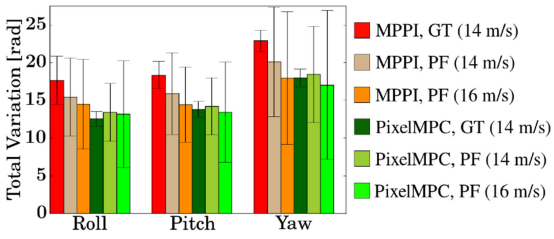

Fig. 6. The total variation of roll, pitch, yaw angles of 25 laps of robot trajectories running the nominal MPPI and the PixelMPC. Both controllers were tested with ground truth (GT) and a particle filter (PF) state estimator. The error bar represents $\mu \pm 2\sigma$. The smaller total variation of the robot orientation implies less shaky robot behavior.

10 laps of racing per each case. As expected, the PixelMPC loses lap time by achieving more visibility of the racecourse. However, we believe it is worth to pay 1.7 sec, sometimes less than 0.2 sec, to achieve 42% $\rightarrow$ 4.5% decrease in time that the robot loses important information in its view.

We also report DOF dynamics' multi-step prediction error in Fig. 5. We see the predicted pixel trajectory is shorter than the actual trajectory in general, but the predicted trajectory closely follows the actual trajectory direction-wise. Note that our MPC scheme solves this compounding error problem with feedback and real-time optimization.

D. Vision-Based State Estimation With Particle Filter

For state estimation with sensors (IMU, cameras, etc.), having more visual information and smooth flying behavior will benefit the state estimation and result in fewer failures overall. The most likely cause of a collision is an inaccurate state estimate. In a racing scenario, we can still assume that the racing map, i.e. the gates' location information is given. Then, one of the biggest challenges will be estimating the robot's state, to perform accurate path planning and control.

For estimating the robot's state, we use a particle filter with an observation model using gate information from observed images. The particle filter is run with 6400 particles and uses the GPU to parallelize the motion and sensor updates.

1) Motion Update: The motion update of the particle filter is done by integrating the IMU measurements directly. Additional Gaussian noise is injected into the filter with mean 0 and variance 0.2 directly on position $[m]$. In addition to that, Gaussian noise is added to the IMU measurements directly both with mean 0 and variance 0.2 for acceleration and variance 0.1 for angular rates. These tunings allow the particle filter to quickly jump to whatever sensor update occurs, but make the state estimate very unstable. The filter's covariance will quickly balloon without any feature detections.

2) Sensor Update: The only sensor model of the particle filter is to use the nominal locations of the gate corners in the 3D world and back project them into the image plane. Then we find the difference between the detected results and the expected ones. Any missing detection is penalized heavily by $4 \times$ W where W is the width of the camera image. Our custom YOLOv3 [15] gate detector is used to generate the detection of the 2D positions from an image along with a bounding box, which includes the third (depth) information.

TABLE IV
SUCCESS RATE (%) OF 25 LAPS (14 m/s AND 16 m/s) AND THE SUCCESS CASE LAP TIME (sec) OF RUNNING MPPI AND PIXELMPC ($t_f = 1.0$, $c_{\text{PIXEL}} =$ 9.0E + 6) WITH A PARTICLE FILTER. THE MAXIMUM COVARIANCE OF POSITION Σ_{max} WAS TESTED WITH A TARGET SPEED 20 m/s ON A STRAIGHT LANE

	[Success rate (%), Lap time (sec)]		Σ_{max}
Ctrl \ $\mathbf{v}_d$	$14m/s$	$16m/s$	$20m/s$
MPPI	[80%, 31.8±0.7]	[52%, 29.6±0.8]	9.2
PixelMPC	[80%, 33.0±0.8]	[**60**%, 30.6±0.7]	**5.7**

Table IV shows that, with the target speed of 14 m/s, the success rate of both cases are the same (80%) but if we increase the target speed to 16 m/s with the same cost parameters, the PixelMPC reports a higher success rate. The failure (crash) cases came from losing target visibility which resulted in the divergence of the state estimation. The 25 trajectories of running PixelMPC ($t_f = 1.0$, $c_{pixel} = 9.0\text{e} + 6$) and the nominal MPPI with a particle filter is shown in Fig. 7. Since the racetrack we used only allows few seconds of flying between two consecutive gates, it is not intuitive to see if the PixelMPC decreases the particle filter covariance because even nominal MPPI could see the target gates very often. Therefore, we did one more straight-line flying test to fully see the effect of PixelMPC on state estimation. In this case, we increased the target speed to 20 m/s, where MPPI has to pitch down a lot to hit the target speed. As soon as the detector detects the gates, the PixelMPC tries to grab the feature in its view and this results in a smaller covariance of the particle filter. The last column of Table IV shows the maximum covariance of position from 25 runs of nominal MPPI and PixelMPC.

VI. CONCLUSION

By fusing vision, path planning, and control into a single optimization framework, high-speed racing can be accomplished with more stable state estimation along with more visual information. Our algorithm can be generally used in any camera-based robot system for visual servoing. Testing our algorithm with real hardware will be our next step to move forward, but there is still room for improvement. The suggested deep optical flow (DOF) dynamics does not take the depth/distance of the target pixel and the robot's velocity information into account. The current DOF approach works well thanks to the

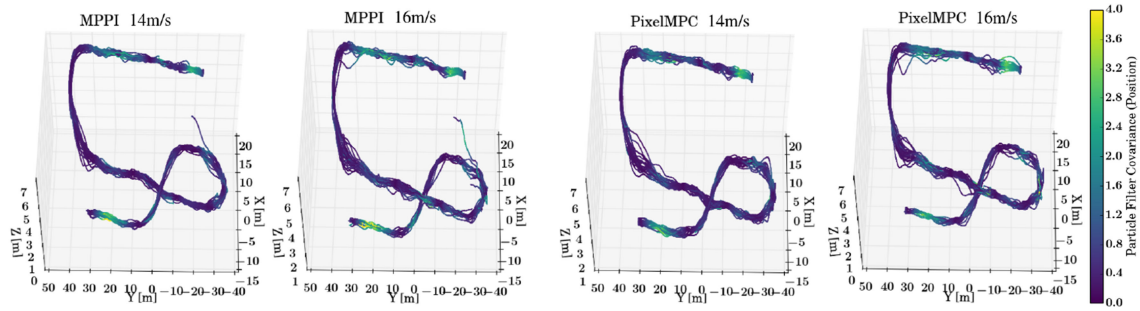

Fig. 7. 25 Laps of running the nominal MPPI (*Left*) and the PixelMPC ($c_{\text{pixel}} = 9.0\text{e}+6$ and $t_f = 1.0$) (*Right*) with a particle filter. The target speed was set to 14 m/s and 16 m/s. The color represents the total covariance in position.

generalization property of the deep neural network, but incorporating the target pixel's depth information will result in a more robust dynamics propagation. Another direction to robustify the suggested dynamics will be propagating the target bounding box, i.e. the 4 corners of it, like a particle filter approach. Lastly, although the constant target velocity settings for racing and other inputs indirectly include the velocity information, directly incorporating the velocity will be helpful also for other non-racing tasks dealing with variable speed and other specific maneuvers.

REFERENCES

[1] Y. Pan *et al.*, "Agile Autonomous Driving using End-to-End deep imitation learning," *Robot.: Sci. Syst.*, 2018. [Online]. Available: http://www.roboticsproceedings.org/rss14/p56.pdf

[2] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, no. 39, pp. 1–40, 2016. [Online]. Available: http://jmlr.org/papers/v17/15-522.html

[3] A. Giusti *et al.*, "A Machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robot. Autom. Lett.*, vol. 1, no. 2, pp. 661–667, Jul. 2016. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7358076

[4] N. Smolyanskiy, A. Kamenev, J. Smith, and S. T. Birchfield, "Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 4241–4247. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8206285

[5] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras," *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7946260

[6] E. Kaufmann *et al.*, "Beauty and the Beast: Optimal methods meet learning for drone racing," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 690–696. [Online]. Available: https://ieeexplore.ieee.org/document/8793631

[7] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "deep drone Racing: Learning agile flight in dynamic environments," in *Proc. 2nd Conf. Robot Learn., (Proceedings of Machine Learning Research, 87). PMLR*, 29–31 Oct. 2018, pp. 133–145. [Online]. Available: http://proceedings.mlr.press/v87/kaufmann18a.html

[8] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Fl owNet 2.0: Evolution of optical flow estimation with deep networks," in *IEEE Conf. Comput. Vision Pattern Recognit.*, Jul. 2017. [Online]. Available: http://lmb.informatik.uni-freiburg.de//Publications/2017/IMKDB17

[9] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, Jun. 2018. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2018/html/Sun_PWC-Net_CNNs _for_CVPR_2018_paper.html

[10] A. Ranjan and M. J. Black, "Optical flow estimation using a spatial pyramid network," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2017. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2017/html/Ranjan_Optical_F low_Estimation_CVPR_2017]_paper.html

[11] B. Penin, R. Spica, P. R. Giordano, and F. Chaumette, "Vision-based minimum-time trajectory generation for a quadrotor UAV," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2017, pp. 6199–6206. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8206522

[12] V. Murali, I. Spasojevic, W. Guerra, and S. Karaman, "Perception-aw are trajectory generation for aggressive quadrotor flight using differential flatness," in *Proc. Amer. Control Conf.*, Jul. 2019, pp. 3936–3943. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8814697

[13] T. Ngeli, J. Alonso-Mora, A. Domahidi, D. Rus, and O. Hilliges, "Real-Time motion planning for aerial videography with dynamic obstacle avoidance and viewpoint optimization," *IEEE Robot. Autom. Lett.*, vol. 2, no. 3, pp. 1696–1703, Jul. 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7847361

[14] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "PAMPC: Perception-Aware model predictive control for quadrotors," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2018, pp. 1–8. [Online]. Available: http://rpg.ifi.uzh.ch/docs/IROS18_Falanga.pdf

[15] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 2018 *arXiv*. [Online]. Available: https://arxiv.org/abs/1804.02767

[16] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-Time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst. 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 91–99. [Online]. Available: https://arxiv.org/abs/1506.01497

[17] G. Williams *et al.*, "Information theoretic MPC for model-based reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2017, pp. 1714–1721. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7989202

[18] W. Guerra, E. Tal, V. Murali, G. Ryou, and S. Karaman, "FlightGoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual Reality," 2019 *arXiv*. [Online]. Available: https://arxiv.org/abs/1905.11377

[19] G. Farnebck, "Two-Frame motion estimation based on polynomial expansion," in *Proc. Scand. Conf. Image Anal.*, vol. 2749, 06 2003, pp. 363–370. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-45103-X_50

[20] G. Bradski, "The OpenCV Library," *Dr. Dobb's J. Softw. Tools*, pp. 122–125, 2000.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[22] E. Ilg, T. Saikia, M. Keuper, and T. Brox, "Occlusions, Motion and depth boundaries with a generic network for disparity, optical flow or scene flow estimation," in *Eur. Conf. Comput. Vision*, 2018. [Online]. Available: http://openaccess.thecvf.com/content_ECCV_2018/papers/Eddy_Ilg_Occlu sions_Motion_and_ECCV_2018_paper.pdf