Deep Visual MPC-Policy Learning for Navigation

Noriaki Hirose, Fei Xia, Roberto Martín-Martín, Amir Sadeghian, Silvio Savarese

Abstract—Humans can routinely follow a trajectory defined by a list of images/landmarks. However, traditional robot navigation methods require accurate mapping of the environment, localization, and planning. Moreover, these methods are sensitive to subtle changes in the environment. In this paper, we propose a Deep Visual MPC-policy learning method that can perform visual navigation while avoiding collisions with unseen objects on the navigation path. Our model PoliNet takes in as input a visual trajectory and the image of the robot’s current view and outputs velocity commands for a planning horizon of N steps that optimally balance between trajectory following and obstacle avoidance. PoliNet is trained using a strong image predictive model and traversability estimation model in a MPC setup, with minimal human supervision. Different from prior work, PoliNet can be applied to new scenes without retraining. We show experimentally that the robot can follow a visual trajectory when varying start position and in the presence of previously unseen obstacles. We validated our algorithm with tests both in a realistic simulation environment and in the real world. We also show that we can generate visual trajectories in simulation and execute the corresponding path in the real environment. Our approach outperforms classical approaches as well as previous learning-based baselines in success rate of goal reaching, sub-goal coverage rate, and computational load.

I. INTRODUCTION

An autonomously moving agent should be able to reach any location in the environment in a safe and robust manner. Traditionally, both navigation and obstacle avoidance have been performed using signals from Lidar or depth sensors [1][2]. However, these sensors are expensive and prone to failures due to reflective surfaces, extreme illumination or interference [3]. On the other hand, RGB cameras are inexpensive, available on almost every mobile agent, and work in a large variety of environmental and lighting conditions. Further, as shown by many biological systems, visual information suffices to safely navigate the environment.

When the task of moving between two points in the environment is addressed solely based on visual sensor data, it is called visual navigation [4]. In visual navigation, the goal and possibly the trajectory are given in image space. Previous approaches to visually guide a mobile agent have approached the problem as a control problem based on visual features, leading to visual servoing methods [5]. However, these methods have a small region of convergence and do not provide safety guarantees against collisions. More recent approaches [4][5][6] have posed the navigation problem as a reinforcement learning task. After the training phase, these methods can navigate large environments without collisions. However, they learn each trajectory from thousand of trials on simulation, and therefore they cannot be applied to novel environments or for new trajectories.

In this work, we present a novel navigation system based solely on visual information from a 360° camera. Unlike previous approaches, our method can navigate through long distances in completely novel environments in a safe and robust manner. This is possible owing to a novel neural network model that is trained to learn a MPC policy and produce efficient and safe control commands towards to goal. This network learns based on a scene-view synthesis algorithm that predicts possible images conditioned on virtual robot velocities, and a robust vision-based traversability estimation network.

We evaluate our proposed method and compare to multiple baselines both in the real world and in simulation [8]. In total, we ran 4800 tests in simulation and 110 tests in the real world. Our algorithm robustly achieves the goal while avoiding obstacles that force it to deviate from the original visual trajectory. Moreover, we demonstrate that our method is capable of bridging the gap between simulation and the real world and can execute visual trajectories that have been purely generated in simulation, without the need for retraining. With our method we grant mobile agents the ability to navigate between any two locations connected by a visual trajectory. Our datasets, trained models and supplementary videos are released on the project website.

II. RELATED WORK

A. Visual Servoing

Visual Servoing is the task of controlling an agent’s motion so as to minimize the difference between a goal and a current image (or image features) [5][9][10][11][12]. The most common approach for visual servoing involves defining an Image Jacobian that correlates robot actions to changes in the image space [5] and then minimizing the difference between the goal and the current image (using the Image
There are three main limitations with visual servoing approaches: first, given the greedy behavior of the servoing controller, it can get stuck in local minima. Second, direct visual servoing requires the Image Jacobian to be computed, which is costly and requires detailed knowledge about the sensors, agent and/or environment. Third, visual servoing methods only converge well when the goal image can be perfectly recreated through agents actions. In the case of differences in the environment, visual servoing methods can easily break [13].

The method we present in this paper goes beyond these limitations: we propose to go beyond a pure greedy behavior by using a model predictive control approach. Also, our method does not require expensive Image Jacobian computation but instead learns a neural network that correlates actions and minimization of the image difference. And finally, we demonstrate that our method is not only robust to differences between subgoal and real images, but even robust to the large difference between simulation and real so as to allow sim-to-real transfer.

B. Visual Model Predictive Control

Model predictive control (MPC) is a multivariate control algorithm that is used to calculate optimum control moves while satisfying a set of constraints. It can be used when a dynamic model of the process is available. Visual model predictive control studies the problem of model predictive control within a visual servoing scheme [15, 16, 17, 6, 18]. Sauvée et al. [19] proposed an Image-based Visual Servoing method (IBVS, i.e. directly minimizing the error in image space instead of explicitly computing the position of the robot that would minimize the image error) with nonlinear constraints and a non-linear MPC procedure. In their approach, they measure differences at four pixels on the known objects at the end effector. In contrast, our method uses the differences in the whole image to be more robust against noise and local changes and to capture the complicated scene.

Finn and Levine [6] proposed a Visual MPC approach to push objects with a robot manipulator and bring them to a desired configuration defined in image space. Similar to ours, their video predictive model is a neural network. However, to compute the optimal next action they use a sampling-based optimization approach. Compared to Finn and Levine [6], our work can achieve longer predictive and control horizon by using a 360° view image and a more efficient predictive model (see Section III). At execution time, instead of a sampling-based optimization method, we use a neural network that directly generates optimal velocity commands; this reduces the high computational load of having to roll out a predictive model multiple times. We also want to point out that due to the partial observable nature of our visual path following problem, it is more challenging than the visual MPC problem of table-top manipulation.

C. Deep Visual Based Navigation

There has been a surge of creative works in visual-based navigation in the past few years. These came with a diverse set of problem definitions. The common theme of these works is that they don’t rely on traditional analytic Image Jacobians nor SLAM-like systems [20, 21, 22, 23] to map and localize and control motion, but rather utilize recent advances in machine learning, perception and reasoning to learn control policies that link images to navigation commands [23, 24, 25, 26, 4].

A diverse set of tools and methods have been proposed for visual navigation. Learning-based methods including Deep Reinforcement Learning (DRL) [41, 27, 28] and Imitation Learning (IL) [29] have been used to obtain navigation policies. Other methods [23, 30] use classical pipelines to perform mapping, localization and planning with separate modules. Chen et al. [31] proposed a topological representation of the map and a planner to choose behavior. Low level control is behavior-based, trained with behavior cloning. Among those works, the subset of visual navigation problems most related to our work is the so-called visual path following. [7] focuses on following a visual trajectory defined by a list of images.

Compared with [7], our work is validated in the real world and can record a trajectory in a 3D-reconstructed environment and follow the trajectory in real environment (sim-to-real transfer). Also, we are able to achieve higher success rate than [7] in similar settings.

III. Method

In this section, we introduce the details of our deep visual model predictive navigation approach based on 360° RGB images.

The input to our method is a visual trajectory and a current image, both from a 360° field-of-view RGB camera. The trajectory is defined as consecutive images (i.e. subgoals or waypoints) from a starting location to a target location sampled at a constant time interval. We represent the 360° images as two 180° fisheye images (see Fig. 2). Thus, the trajectory can be written as a sequence of $K$ subgoal image pairs, $\{(I_0^f, I_0^b), (I_1^f, I_1^b), \ldots, (I_{K-1}^f, I_{K-1}^b)\}$, where the superindex $f$ indicates front and $b$ indicates back. This trajectory can be obtained by teleoperating the real robot or moving a virtual camera in a simulator.

The goal of our control policy is to minimize the difference between the current 360° camera image at time $t$, $(I_t^f, I_t^b)$, and the next subgoal image in the trajectory, $(I_{t+1}^f, I_{t+1}^b)$, while avoiding collisions with obstacles in the environment. These obstacles may be present in the visual trajectory or not, being completely new obstacles. To minimize the image difference, our control policy moves towards a location such that the
image from onboard camera looks similar to the next subgoal image.

A simple heuristic determines if the robot arrived at the current subgoal successfully and switches to the next subgoal. The condition to switch to the next subgoal is the absolute pixel difference between current and subgoal images: \( |I^f_i - \hat{I}^f_i| + |I^b_j - \hat{I}^b_j| < d_{\text{th}} \), where \( j \) is the index of the current subgoal and \( d_{\text{th}} \) is a threshold adapted experimentally.

The entire process is depicted in Fig. 2. Transitioning between the subgoals our robot can arrive at the target destination without any geometric localization and path planning on a map.

A. Control Policy

We propose to control the robot using a model predictive control (MPC) approach in the image domain. However, MPC cannot be solved directly for visual navigation since the optimization problem is non-convex and computationally prohibitive. Early stopping the optimization leads to suboptimal solutions (see Section V). We propose instead to learn the MPC-policy with a novel deep neural network we call PoliNet. In the following we first define the MPC controller, which PoliNet is trained to emulate, and then describe PoliNet itself.

PoliNet is trained to minimize a cost function \( J \) with two objectives: following the trajectory and moving through traversable (safe) areas. This is in contrast to prior works that only care about following the given trajectory. We propose to achieve these objectives minimizing a linear combination of three losses, the Image Loss, the Traversability Loss and the Reference Loss. The optimal set of \( N \) next velocity commands can be calculated by the minimization of the following cost:

\[
J = J^{\text{img}} + \kappa_1 J^{\text{trav}} + \kappa_2 J^{\text{ref}}
\]

with \( \kappa_1 \) and \( \kappa_2 \) constant weights to balance between the objectives. To compute these losses, we will need to predict future images conditioned on possible robot velocities. We use a variant of our previously presented approach VUNet [32] as we explain in Section III-B. In the following, we will first define the components of the loss function assuming predicted images, followed by the description of our VUNet based predictive model.

Image Loss: We define the image loss, \( J^{\text{img}} \), as the mean of absolute pixel difference between the subgoal image \( (I^f_j, I^b_j) \) and the sequence of \( N \) predicted images \( (\hat{I}^f_{t+i}, \hat{I}^b_{t+i})_{i=1\ldots N} \) as follows:

\[
J^{\text{img}} = \frac{1}{2N \cdot N_{\text{pix}}} \sum_{i=0}^{N} w_i (|I^f_j - \hat{I}^f_i| + |I^b_j - \hat{I}^b_i|)
\]

with \( N_{\text{pix}} \) being the number of pixels in the image, \( 128 \times 128 \times 3 \), \( (\hat{I}^f_{t+i}, \hat{I}^b_{t+i})_{i=1\ldots N} \) are predicted images generated by our predictive model (Section III-B) conditioned on virtual velocities, and \( w_i \) weights differently the contributions of consecutive steps for the collision avoidance.

Traversability Loss: With the traversability loss, \( J^{\text{trav}} \), we aim to penalize areas that constitute a risk for the robot. This risk has to be evaluated from the predicted images. Hirose et al. [33] presented GONet, a deep neural network-based method that estimates the traversable probability from an RGB image. Here we apply GONet to our front predicted images such that we compute the traversability cost based on the traversable probability \( \hat{p}^{\text{trav}}_{t+i} = \text{GONet}(\hat{I}^f_{t+i}) \).

To emphasize the case of lower traversable probability in \( J^{\text{trav}} \), we kernelize the traversable probability from GONet as \( \hat{p}^{\text{trav}}_{t+i} = \text{Clip}(\hat{p}^{\text{trav}}_{t+i}, \kappa_{\text{trav}}) \), which effectively scales all the probabilities by the constant value \( \kappa_{\text{trav}} \) and clips all probabilities over \( 1/\kappa_{\text{trav}} \) to 1. Based on the kernelized traversable probability, the traversability cost is defined as:

\[
J^{\text{trav}} = \frac{1}{N} \sum_{i=0}^{N-1} (1 - \hat{p}^{\text{trav}}_{t+i})^2
\]

Reference Loss: The image loss and the traversability loss suffice to follow the visual trajectory while avoiding obstacles. However, we observed that the velocities obtained from solving the MPC problem can be sometimes non-smooth and non-realistic since there is no consideration of acceptableness to the real robot in the optimizer. To generate more realistic velocities we add the reference loss, \( J^{\text{ref}} \), a cost to minimize the difference between the generated velocities \( (v_i, \omega_i)_{i=0\ldots N-1} \) and the real (or simulated) velocities \( (v^\text{ref}_{t+i}, \omega^\text{ref}_{t+i})_{i=0\ldots N-1} \). The reference loss is defined as:

\[
J^{\text{ref}} = \frac{1}{N} \sum_{i=0}^{N-1} (v^\text{ref}_{t+i} - v_i)^2 + \frac{1}{N} \sum_{i=0}^{N-1} (\omega^\text{ref}_{t+i} - \omega_i)^2
\]

This cost is only part of the MPC controller and training process of PoliNet. At inference time PoliNet does not require any geometric information (or the velocities), but only the images defining the trajectory.

B. Predictive Model, VUNet-360

The previously described loss function requires a forward model that generates images conditioned on virtual velocities. Prior work VUNet [32] proposed a view synthesis approach to predict future views for a mobile robot given a current image and robot’s possible future velocities. However, we cannot use VUNet as forward model within the previously defined MPC to train PoliNet because: 1) the original VUNet uses and predicts only a front camera view, and 2) the multiple step prediction process in VUNet is sequential, which not only leads to heavy computations but
also to vanishing gradients if applied directly to train PoliNet. We solve these problems with VUNet-360, a modified version of VUNet that uses as input one 360° image and \( N \) virtual velocities and generates \( N \) predicted future images in parallel.

The network structure of VUNet-360 is depicted in Fig. [3a]. Similar to the original VUNet, our version uses an encoder-decoder architecture with robot velocities concatenated to the latent vector. One novelty of VUNet-360 is that the computation is now in parallel for all \( N \) images. We also introduced the blending module that generates virtual front and back images by fusing information from the input front and back images.

Fig. [3b] shows \( i \)-th blending module for the prediction of \( I_t^i+1 \). Similar to [34], we use bilinear sampling to map pixels from input images to predicted images.

The blending module receives 2 flows, \( F_{t+i}^f, F_{t+i}^b \) and 2 visibility masks \( W_{t+i}^f, W_{t+i}^b \) from the decoder of VUNet-360. This module blends the sampled front and back images \( I_{t+i}^f, I_{t+i}^b \) by \( F_{t+i}^f, F_{t+i}^b \) with the masks \( W_{t+i}^f, W_{t+i}^b \) to produce both front and back image pixels to predict \( I_{t+i}^i \).

To train VUNet-360 we input a real image \((I_t^f, I_t^b)\) and a sequence of real velocities \((v_t, \omega_t)_{i=0\ldots N-1} = (v_{t+i}, \omega_{t+i})_{i=0\ldots N-1} \) collected during robot teleoperation (see Sec. IV) and we minimize the following cost function:

$$J_{VUNet} = \frac{1}{2N \cdot N_{pix}} \sum_{i=1}^{N} \left( |I_t^f - I_t^f|^2 + |I_t^b - I_t^b|^2 \right)$$

(5)

where \((I_t^f, I_t^b)_{i=1\ldots N}\) are the ground truth future images and \((\hat{I}_t^f, \hat{I}_t^b)_{i=1\ldots N}\) are the VUNet predictions.

C. Neural Model Control Policy, PoliNet

The optimization problem of the model predictive controller with the predictive model described above cannot be solved online within the required inference time due to the complexity (non-convexity) of the minimization of the cost function. We propose to train a novel neural network, PoliNet in the dashed black rectangle of Fig. 4 and only calculate PoliNet online to generate the velocities instead of the optimization problem. The network structure of PoliNet is simply constructed with 8 convolutional layers to allow fast online computation on the onboard computer of our mobile robot. In the last layer, we have tanh(·) to limit the linear velocity within \( \pm v_{max} \) and the angular velocity within \( \pm \omega_{max} \).

Similar to the original MPC, the input to PoliNet is the current image, \((I_t^f, I_t^b)\), and the subgoal image, \((\hat{I}_t^f, \hat{I}_t^b)\), and the output is a sequence of \( N \) robot velocities, \((v_t, \omega_t)_{i=0\ldots N-1}\), that move the robot towards the subgoal in image space while keeping it away from non-traversable areas. By forward calculation of PoliNet, VUNet-360, and GONet as shown in Fig. 4, we can calculate the same cost function \( J \) as MPC-policy to update PoliNet. Note that VUNet-360 and GONet are not updated during the training process of PoliNet. VUNet-360 and GONet are only used to calculate the gradient to update PoliNet. To train PoliNet, we need \((I_t^f, I_t^b)\) as current image, \((\hat{I}_t^f, \hat{I}_t^b)\) as the subgoal image, and the tele-operator’s velocities \((v_t, \omega_t)_{i=0\ldots N-1} = ((v_{t+i}, \omega_{t+i})_{i=0\ldots N-1} \) for \( J_{ref} \). We randomly choose the future image as the target image like \( (I_t^f, I_t^b) = (\hat{I}_{t+k}^f, \hat{I}_{t+k}^b) \) from GS4. Here, \( k \) is the random number within \( N_r \).

IV. EXPERIMENTAL SETUP

We evaluate our deep visual navigation approach based on our learned visual MPC method as the navigation system for a Turtlebot 2 with a Ricoh THETA S 360 camera on top. We will conduct experiments both in real world and in simulation using this robot platform.

To train our VUNet-360 and PoliNet networks we collect new data both in simulation and in real world. In real world we teleoperate the robot and collect 10.30 hours of 360° RGB images and velocity commands in twelve buildings at the Stanford University campus. We separate the data from the different buildings into data from eight buildings for training, from two buildings for validation, and from two other buildings for testing.

In simulation we use the GibsonEnv [8] simulator with models from Stanford 2D-3D-S [35] and Matterport3D [36] datasets. The models from Stanford 2D-3D-S are office buildings at Stanford reconstructed using a 3D scanner with texture. This means, for these buildings we have corresponding environments in simulation and real world. Differently, Matterport3D mainly consists of residential buildings. Training on both datasets gives us better generalization performance to different types of environments.

In the simulator, we use a virtual 360° camera with intrinsic parameters matching the result of a calibration of the real Ricoh THETA S camera. We also teleoperate the virtual robot in 36 different simulated buildings for 3.59 hours and split the data into 2.79 hours of training data from 26 buildings, 0.32 hours of data as validation set from another 5 buildings, and 0.48 hours of data as test set from another 5 buildings.

To train VUNet-360 and PoliNet, we balance equally simulator’s and real data. We train iteratively all networks using Adam optimizer with a learning rate of 0.0001 on a Nvidia GeForce RTX 2080 Ti GPU. All collected images are resized into 128×128 before feeding into the network.

Parameters: We set \( N = 8 \) steps as horizon and 0.333 s (3 Hz) as inference time. This inference time allows to use our method in real time navigation. The values correspond to a prediction horizon of 2.667 s into the future. These values are a good balance between long horizon that is generally better in MPC setups, and short predictions that are more reliable with a predictive model as VUNet-360 (see Fig. 5).

\( v_{max} \) and \( \omega_{max} \) are given as 0.5 m/s and 1.0 rad/s. Hence, the maximum range of the prediction can be ±
1.333 m and ± 2.667 rad from the current robot pose, which are large enough to allow the robot avoiding obstacles.

In \( J^{\text{img}} \), \( w_i = 1.0 \) for all time steps except \( i = N \), \( w_N = 5.0 \) to allow the deviation from the original visual trajectory and to arrive at the target image at the last step. \( \kappa_{\text{trav}} \) for \( J^{\text{trav}} \) is set to 1.1 and \( d_{\text{th}} \) to between 0.1 and 0.2 according to empirical performance. The weight of the different terms in the cost function (Eq. [1]) are \( \kappa_1 = 0.5 \) for the traversability loss and \( \kappa_2 = 0.1 \) for the reference loss. The optimal \( \kappa_1 \) is found through ablation studies (see Table [V]). Setting \( \kappa_2 = 0.1 \), we limit the contribution of the reference loss to the overall cost because our goal with this additional term is not to learn to imitate the teleoperator’s exact velocity but to regularize and obtain smooth velocities. In addition, we set \( N_r = 12 \) to randomly choose the subgoal image for the training of PoliNet.

V. EXPERIMENTS

We conduct three sets of experiments to validate our method in both simulation and real world. We first evaluate the predictive module: VUNet-360. Then we evaluate the performance of PoliNet by comparing it against a set of baselines, both as a MPC-learned method and as a core component of our proposed deep visual navigation approach. Finally, we perform ablation studies to understand the importance of the traversability loss computed by GONet in our loss design.

A. Evaluation of VUNet-360

Quantitative Analysis: First we evaluate the quality of the predictions from our trained VUNet-360 on the images of the test set, and compare to the original method, VUNet. We use two metrics: the pixel difference (lower is better) and SSIM (higher is better). As is shown in Fig. 5 [c], VUNet-360 clearly improves over the original method in all stages of the prediction.

We also compare the computation efficiency of VUNet and VUNet-360, since this is relevant for efficient training of PoliNet. VUNet-360 has smaller memory footprint (901MB vs. 1287MB) and higher frequency (19.28 Hz vs. 5.03 Hz) than VUNet. This efficiency gain is because VUNet-360 predicts multiple images in parallel with a single forward pass of the encoder-decoder network.

Qualitative Analysis: Fig. 5 [a,b] shows predicted images from VUNet and VUNet-360 for two representative scenarios. In the images for each scenario, on the left side we show the current real front and back images that compose the 360\(^\circ\) image. From second to fourth column, we show the ground truth image, VUNet prediction and VUNet-360 prediction for 1, 4 and 8 steps. In the first scenario, the robot is turning in place. In the second scenario, the robot is moving forward while slightly turning left. We observe that VUNet-360 more accurately predicts future images and better handles occlusions since it is able to exploit information from the back camera.

B. Evaluation of PoliNet

We evaluate PoliNet and compare it to various baselines, first as method to learn MPC-policies and then as core of our visual navigation approach. We now briefly introduce the baselines for this evaluation, while encouraging readers to refer to the original papers for more details.

MPC online optimization (back propagation) [38]: Our baseline backpropagates on the loss function \( J \) to search for the optimized velocities instead of using a neural network. This approach is often used when the MPC objective is differentiable [38] [39]. We evaluate three different number of iterations for the backpropagation, \( n_{\text{it}} = 2, 20, \) and 100.

Stochastic optimization [6]: We use a cross-entropy method (CEM) stochastic optimization algorithm similar to [6] as baseline of a different optimization approach. To do that, we sample \( M \) sets of \( N \) linear and angular velocities, and calculate \( J \) for each velocity. Then, we select \( K \) set of velocities with the smallest \( J \) and resample a new set of \( M \) from a multivariate Gaussian distribution of \( K \) set of velocities. In our comparison to our approach we evaluate three sets of parameters of this method: \((M, K, n_{\text{it}}) = (10, 5, 3), (20, 5, 5), \) and \((100, 20, 5)\).

Open loop control: This baseline replays directly to the teleoperator’s velocity commands in open loop as a trajectory.

Imitation Learning (IL) [29]: We train two models to imitate the teleoperator’s linear and angular velocity. The first model learns to imitate only the next velocity (comparable to PoliNet with \( N = 1 \)). The second model learns to imitate the next eight velocities (comparable to PoliNet with \( N = 8 \)).

PoliNet as learned MPC: Table [II] depicts the results of our quantitative evaluation of PoliNet as learned MPC approach. Since our goal is to learn to emulate MPC, we report \( J \), \( J^{\text{img}} \), \( J^{\text{trav}} \) and \( J^{\text{ref}} \) of our approach and the baselines. In addition, we show the memory consumption and the inference time, crucial values to apply these methods in a
PoliNet for Deep Visual Navigation: To evaluate PoliNet as the core of our deep visual navigation approach, we perform three sets of experiments. In the first set we evaluate navigation in simulation with simulation-generated trajectories. In simulation we can obtain ground truth of the robot pose and evaluate accurately the performance. In the second set we evaluate in real world the execution of real world visual trajectories. In the third set we evaluate sim2real transfer, whether PoliNet-based navigation can reproduce in real world trajectories generated in simulation.

For the first set of experiment, navigation in simulation, we select three simulated environments (see Fig. [a]) and create simulated trajectories that generate visual trajectories by sampling images at 0.533 Hz (dashed lines). The robot start randomly from the gray area and needs to arrive at the goal. Obstacles are placed in blue areas (on trajectory) and red areas (off trajectory). To model imperfect control and floor slippage, we multiply the output velocities by a uniformly sampled value between 0.6 and 1.0. With this large noise execution we evaluate the robustness of the policies against noise.

Table II depicts the success rate, the coverage rate (ratio of the arrival at each subgoal images), and SPL [40] of 100 trials for each of the three scenarios and the average. Here, the definition of the arrival is that the robot is in the range of ±0.5 m of the position where the subgoal image was taken (note that the position of the subgoal image is only used for the evaluation).

![Image 60x559 to 181x738]

Fig. 6: Simulation environments for quantitative evaluation and robot trajectories of two examples. Top image in (a), (b) and (c) is the map of each environment. Bottom images show images from the visual trajectory, which is indicated in dashed line on the map. (d) shows robot trajectories in (i) case A and (ii) case C in simulator with and without obstacle. The superimposed blue lines are the robot trajectories from 10 different initial poses without the obstacle. The red and green lines are the robot trajectories with the obstacle shown as the rectangular red and green region. The grey circle is the goal region, which is used to determine if the robot arrived at the goal.

Our method achieves 0.997 success rate without obstacles and 0.850 with obstacles on average and outperform all baseline methods. In addition, SPL for our method is close to the success rate, which means that the robot can follow the subgoal images without large deviation.

![Image 199x559 to 320x737]

![Image 338x559 to 459x737]
TABLE II: Navigation with PoliNet and baselines in simulation (navigation success rate/subgoal coverage rate/SPL)

|                  | (a) Backprop.  
|------------------|---------------|---------------|
|                  | average       | Case A: 9.6 m | Case B: 5.2 m | Case C: 5.5 m |
| wo ob.           | 0.000 / 0.211 | 0.000 / 0.175 | 0.000 / 0.264 | 0.000 / 0.193 |
| (n_{itr} = 2)   | 0.000 / 0.205 | 0.000 / 0.169 | 0.000 / 0.260 | 0.000 / 0.187 |
| w/ ob.           | 0.000 / 0.147 | 0.000 / 0.102 | 0.000 / 0.184 | 0.000 / 0.156 |
| (M = 10, K = 5, n_{itr} = 3) | 0.000 / 0.150 | 0.000 / 0.104 | 0.000 / 0.181 | 0.000 / 0.164 |
| (g) Open Loop    | 0.023 / 0.366 | 0.040 / 0.206 | 0.030 / 0.394 | 0.000 / 0.497 |
| w/ ob.           | 0.010 / 0.287 | 0.000 / 0.353 | 0.030 / 0.274 | 0.000 / 0.234 |
| (h) Imitation learning | 0.320 / 0.721 | 0.000 / 0.501 | 0.960 / 0.981 | 0.000 / 0.683 |
| (N = 1)          | 0.297 / 0.659 | 0.000 / 0.501 | 0.890 / 0.944 | 0.000 / 0.534 |
| w/ ob.           | 0.103 / 0.592 | 0.110 / 0.618 | 0.120 / 0.531 | 0.080 / 0.627 |
| (N = 8)          | 0.310 / 0.689 | 0.100 / 0.619 | 0.800 / 0.922 | 0.030 / 0.528 |
| w/ ob.           | 0.850 / 0.916 | 0.990 / 0.981 | 0.910 / 0.963 | 0.740 / 0.805 |
| (i) Our method   | 0.997 / 0.999 | 0.990 / 0.996 | 1.000 / 1.000 | 1.000 / 1.000 |
| w/ ob.           | 0.850 / 0.850 | 0.900 / 0.899 | 0.910 / 0.910 | 0.740 / 0.805 |

In the second set of experiments we evaluate PoliNet in real world navigation with and without previously unseen obstacles. We record a trajectory with the robot and evaluate it different days and at different hours so that the environmental conditions change, e.g., different position of the furniture, dynamic objects like the pedestrians, and changes in the lighting conditions. We normalize $\left( I_f^J, I_t^J \right)$ to have the same mean and standard deviation as $\left( I_f^J, I_t^J \right)$. Table II shows the success rate and the coverage rate with and without obstacles in the original path. Our method can achieve high success rate and coverage rate for all cases and outperforms the baseline of imitation learning with eight steps by a large margin. (Note: other baselines cannot be used in this real time setup).

The first three rows of Fig. 7 depict some exemplary images from navigation in real world. The figure depicts the current image (left), subgoal image (middle) and the predicted image at the eighth step by VUNet-360 conditioned by the velocities from PoliNet (right). There are some environment changes between the time the trajectory was recorded (visible in the subgoal image) and the testing time (visible in current image). For example, the door is opened in first example (top row), the light in one room is turned on and the brown box is placed at the left side in second example (second row), and the lighting conditions are different and a pedestrian is visible in the third example (third row). Even with these changes, our method generates accurate image predictions, close to the subgoal image. This indicates that PoliNet generates velocities that navigate correctly the robot towards the position where the subgoal image was acquired.

In the third set we evaluate sim-to-real transfer: using visual trajectories from the simulator to navigate in the corresponding real environment. We perform 10 trials for the trajectories at different days and times of the day. The results of the sim-to-real evaluation are summarized in Table IV. The performance of our navigation method is worse than real-to-real, which is expected because there is domain gap between simulation and real world. Despite of that, our method can still arrive at the destination without collision in most of the experiments, indicating that the approach can be applied to generate virtual visual trajectories to be executed in real world.

The second three rows of Fig. 7 depict some exemplary images from navigation in the sim-to-real setup. The discrepancies between the simulated images (subgoals) and the real images (current) are dramatic. For example, in 4th row, the black carpet is removed; in 5th and 6th row, there are big color differences. In addition, the door is opened in 6th row. To assess whether the velocities from PoliNet are correct, we compare the predicted 8th step image and subgoal image. Similar images indicate that the velocities from PoliNet allow to minimize visual discrepancy. Despite the changes in the environment, our deep visual navigation method based on PoliNet generates correctly velocities to minimize the visual differences.

TABLE III: Navigation in real world

|                  | Case 1: 24.1 m | Case 2: 16.1 m |
|------------------|---------------|---------------|
|                  | wo ob.        | w/ ob.        | wo ob.        | w/ ob.        |
| (b) IL (N=8)     | 0.10 / 0.493  | 0.10 / 0.617  | 0.90 / 0.967  | 0.40 / 0.848  |
| (i) Our method   | 1.00 / 1.000  | 0.80 / 0.907  | 1.00 / 1.000  | 0.80 / 0.910  |

TABLE IV: Navigation in sim-to-real

|                  | Case 3: 6.6 m | Case 4: 8.4 m | Case 5: 12.7 m |
|------------------|---------------|---------------|---------------|
|                  | wo ob.        | w/ ob.        | wo ob.        |
| (i) Our method   | 0.90 / 0.983  | 0.80 / 0.867  | 0.80 / 0.933  |

C. Ablation Study of Traversability Loss Generation

In our method, $J_{trav}$ is one of the most important components for navigation with obstacle avoidance. Table V is the ablation study for $J_{trav}$. We evaluate $J$, $J_{img}$, $J_{trav}$ and $J_{ref}$ for each weighting factor $q_g = 0.0, 0.5, 1.0$ of $J_{trav}$ on the test data of both simulator’s images and real images. In addition, we evaluate the model’s navigation performance in simulator.

We test our method 100 times in 3 different environments with and without obstacle. Average of the success rates (the ratio which the robot can arrive at the goal) are listed in the most right side.

Bigger $q_g$ can lead to smaller $J_{trav}$ and bigger $J_{img}$. However, the success rate of $q_g = 1.0$ is almost zero even for the environment without obstacle. Because too big $q_g$ leads the conservative policy in some cases. The learned policy tend to avoid narrow paths to keep $J_{trav}$ high, failing to arrive
at target image. As a result, we decide to use the model with \( q_g = 0.5 \) for all evaluations for our method.

### TABLE V: Ablation Study of Traversability Loss Generation

| Method       | \( q_g \) | \( \text{sim.} \) | \( \text{real} \) | With ob. | Wo ob. |
|--------------|----------|-----------------|-----------------|----------|--------|
| Our method   | \( q_g = 0.0 \) | 0.244 | 0.217 | 0.161 | 0.261 | 0.810 | 0.780 |
|              | \( q_g = 0.5 \) | 0.277 | 0.221 | 0.062 | 0.245 | \textbf{0.997} | \textbf{0.850} |
| Our method   | \( q_g = 1.0 \) | 0.308 | 0.236 | 0.041 | 0.306 | 0.000 | 0.003 |

### VI. CONCLUSION AND FUTURE WORK

We presented a novel approach to learn MPC policies with deep neural networks and apply them to visual navigation with a 360\(^\circ\)RGB camera. We presented PoliNet, a neural network trained with the same objectives as an MPC controller so that it learns to generate velocities that minimize the difference between the current robot’s image and subgoal images in a visual trajectory, avoiding collisions and consuming less computational power than normal visual MPC approaches. Our experiments indicate that a visual navigation system based on PoliNet navigates robustly following visual trajectories not only in simulation but also in the real world. However, our method is not without limitations. Our method occasionally fail to avoid large obstacles if they occupy a large part of a subgoal image. In addition, the current predictive horizon may be not enough to plan long detours to avoid these large obstacles. We plan to explore increasing the prediction horizon of VUNet-360 in future work. Also, occasionally the simple heuristic to switch between subgoal images fails. We plan to replace the heuristic with a neural network to recognize the next subgoal image.

### REFERENCES

[1] F. Flacco, T. Kröger, A. De Luca, and O. Khatib, “A depth space approach to human-robot collision avoidance,” in 2012 IEEE International Conference on Robotics and Automation. IEEE, 2012, pp. 338–345.
[2] D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” IEEE Robotics & Automation Magazine, vol. 4, no. 1, pp. 23–33, 1997.
[3] R. Martín-Martí, M. Lorbach, and O. Brock, “Deterioration of depth measurements due to interference of multiple rgb-d sensors,” in 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2014, pp. 4205–4212.
[4] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, “Target-driven visual navigation in indoor scenes using deep reinforcement learning,” in 2017 IEEE international conference on robotics and automation (ICRA). IEEE, 2017, pp. 3357–3364.
[5] S. Hutchinson, G. D. Hager, and P. I. Corke, “A tutorial on visual servo control,” IEEE transactions on robotics and automation, vol. 12, no. 5, pp. 651–670, 1996.
[6] C. Finn and S. Levine, “Deep visual foresight for planning robot motion,” in 2017 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2017, pp. 2786–2793.
[7] A. Kumar, S. Gupta, D. Fouhey, S. Levine, and J. Malik, “Visual memory for robust path following,” in Advances in Neural Information Processing Systems, 2018, pp. 773–782.
[8] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese, “Gibson env: Real-world perception for embodied agents,” in CVPR, 2018.
[9] B. Espiau, F. Chaumette, and P. Rives, “A new approach to visual servoing in robotics,” IEEE Transactions on Robotics and Automation, vol. 8, no. 3, pp. 313–326, 1992.
[10] F. Chaumette and S. Hutchinson, “Visual servo control. i. basic approaches,” IEEE Robotics & Automation Magazine, vol. 13, no. 4, pp. 82–90, 2006.
[11] E. Malis, F. Chaumette, and S. Boudet, “2 1/2 d visual servoing,” IEEE Transactions on Robotics and Automation, vol. 15, no. 2, pp. 238–250, 1999.
[12] F. Chaumette and S. Hutchinson, “Visual servo control. ii. advanced approaches [tutorial],” IEEE Robotics & Automation Magazine, vol. 14, no. 1, pp. 109–118, 2007.
[13] N. J. Cowan, J. D. Weingarten, and D. E. Koditschek, “Visual servoing via navigation functions,” IEEE Transactions on Robotics and Automation, vol. 18, no. 4, pp. 521–533, Aug 2002.
[14] J. B. Rawlings and D. Q. Mayne, Model predictive control: Theory and design. Nob Hill Pub. Madison, Wisconsin, 2009.
[15] B. Calli and A. M. Dollar, “Vision-based model predictive control for within-hand precision manipulation with underactuated grippers,” in 2017 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2017, pp. 2839–2845.
[16] V. Tolani, “Visual model predictive control,”
Master’s thesis, EECS Department, University of California, Berkeley, May 2018. [Online]. Available: [http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-69.html](http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-69.html)

[17] Z. Li, C.-Y. Su, and W. Zhang, “Vision-based model predictive control for steering of a nonholonomic mobile robot,” IEEE Transactions on Control Systems Technology, vol. 24, no. 2, pp. 553–564, 2016.

[18] G. Allibert, E. Courtial, and Y. Touré, “Real-time visual predictive controller for image-based trajectory tracking of a mobile robot,” IFAC Proceedings Volumes, vol. 41, no. 2, pp. 11244–11249, 2008.

[19] M. Sauvéé, P. Poignet, E. Dombre, and E. Courtial, “Image based visual servoing through nonlinear model predictive control,” in Proceedings of the 45th IEEE Conference on Decision and Control. IEEE, 2006, pp. 1776–1781.

[20] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: part i,” IEEE robotics & automation magazine, vol. 13, no. 2, pp. 99–110, 2006.

[21] S. Thrun, W. Burgard, and D. Fox, “A probabilistic approach to concurrent mapping and localization for mobile robots,” Autonomous Robots, vol. 5, no. 3-4, pp. 253–271, 1998.

[22] ——, “A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping,” in ICRA, vol. 1, 2000, pp. 321–328.

[23] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, “Cognitive mapping and planning for visual navigation,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 2616–2625.

[24] S. Gupta, D. Fouhey, S. Levine, and J. Malik, “Unifying map and landmark based representations for visual navigation,” arXiv preprint arXiv:1712.08125, 2017.

[25] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sündherauf, I. Reid, S. Gould, and A. van den Hengel, “Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 3674–3683.

[26] S. Brahmbhatt and J. Hays, “Deepnav: Learning to navigate large cities,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 5193–5202.

[27] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, “Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation,” in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 1–8.

[28] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, “Learning agile and dynamic motor skills for legged robots,” Science Robotics, vol. 4, no. 26, p. eaau5872, 2019.

[29] 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). IEEE, 2018, pp. 1–9.

[30] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, “A survey of motion planning and control techniques for self-driving urban vehicles,” IEEE Transactions on intelligent vehicles, vol. 1, no. 1, pp. 33–55, 2016.

[31] K. Chen, J. P. de Vicente, G. Sepulveda, F. Xia, A. Soto, M. Vázquez, and S. Savarese, “A behavioral approach to visual navigation with graph localization networks,” 2019.

[32] N. Hirose, A. Sadeghian, F. Xia, R. Martín-Martín, and S. Savarese, “Vunet: Dynamic scene view synthesis for traversability estimation using an rgb camera,” IEEE Robotics and Automation Letters, 2019.

[33] N. Hirose, A. Sadeghian, M. Vázquez, P. Goebel, and S. Savarese, “Gonet: A semi-supervised deep learning approach for traversability estimation,” in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2018, pp. 3044–3051.

[34] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros, “View synthesis by appearance flow,” in European conference on computer vision. Springer, 2016, pp. 286–301.

[35] I. Armeni, S. Sax, A. R. Zamir, and S. Savarese, “Joint 2d-3d-semantic data for indoor scene understanding,” arXiv preprint arXiv:1702.01105, 2017.

[36] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang, “Matterport3d: Learning from rgb-d data in indoor environments,” arXiv preprint arXiv:1709.06158, 2017.

[37] N. Hirose et al., “Gonet++: Traversability estimation via dynamic scene view synthesis,” arXiv preprint arXiv:1806.08864, 2018.

[38] N. Wahlström, T. B. Schön, and M. P. Deisenroth, “From pixels to torques: Policy learning with deep dynamical models,” arXiv preprint arXiv:1502.02251, 2015.

[39] N. Hirose, R. Tajima, K. Sukigara, and M. Tanaka, “Personal robot assisting transportation to support active human life reference generation based on model predictive control for robust quick turning,” in 2014 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2014, pp. 2223–2230.

[40] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva et al., “On evaluation of embodied navigation agents,” arXiv preprint arXiv:1807.06757, 2018.
APPENDIX I
NETWORK STRUCTURE

The details of the network structures are explained in the appendix.

A. VUNet-360

VUNet-360 can predict 8 steps future images by one encoder-decoder network, as shown in Fig. 8. Different from the previous VUNet, VUNet-360 needs to have the network structure to merge the pixel values of the front and back side view for more precise prediction. The concatenated front image \( f_i \) and horizontally flipped back image \( b_f \) are fed into the encoder of VUNet-360. The encoder is constructed by 8 convolutional layers with batch normalization and leaky relu function. Extracted feature by the encoder is concatenated with \( N \) steps virtual velocities \((v_i, \omega_i)_{i=0 \cdots N-1}\) to give the information about the robot pose in the future. By giving 8 de-convolutional layers in the decoder part, \( 128 \times 128 \times 128 \) feature is calculated for \( N \) steps prediction. 

\( 12 \times 128 \times 128 \) feature is fed into each blending module to predict the front and back side images by the synthesis of the internally predicted images. We explain the behavior of \( i \)-th blending module as the representative one. For the prediction of the front image \( f_i \), we internally predict \( f_i^{ff} \) and \( f_i^{bf} \) by the bilinear sampling of \( f_i \) and \( b_f \). Then, we synthesize \( f_i^{t+i} \) by blending \( f_i^{ff} \) and \( f_i^{bf} \) as follows,

\[
\begin{align*}
\hat{I}_t^{f+i} = & \hat{I}_t^{ff} \otimes W_{ff} + \hat{I}_t^{bf} \otimes W_{bf}, \\
\end{align*}
\]

where \( W_{ff} \) and \( W_{bf} \) are 1D probabilistic \( 1 \times 128 \times 128 \) selection mask, and \( \otimes \) is the element-wise product. We use a softmax function for \( W_{ff} \) and \( W_{ff} + W_{bf} \) to satisfy \( W_{ff}(u, v) + W_{bf}(u, v) = 1 \) for any same image coordinates \((u, v)\). \( f_i^{t+i} \) can be predicted by same manner as \( f_i^{t+i} \), although the explanation is omitted.

B. PoliNet

PoliNet can generate \( N \) steps robot velocities \((v_i, \omega_i)_{i=0 \cdots N-1}\) for the navigation. In order to generate the appropriate velocities, PoliNet needs to internally understand i) relative pose between current and target image view, and ii) traversable probability at the future robot pose. On the other hand, computationally light network is required for the online implementation. Concatenated
Fig. 10: Predicted image by VUNet-360. Most left column shows current image. The other column shows the ground truth (GT), the predicted image by VUNet, and the predict image by VUNet-360 for each step. Top 6 rows are for test data of the real environment. Bottom 6 rows are for test data of the simulator’s environment.

Fig. 11: Visualization of the navigation performance of PoliNet in real environment and simulation. First and second column show current and subgoal image. The other column shows the predicted images by VUNet-360 with the virtual velocities from PoliNet. Top 3 rows are for real-to-real. Bottom 3 rows are for sim-to-real.