RTVS: A Lightweight Differentiable MPC Framework for Real-Time Visual Servoing

M. Nomaan Qureshi*[1], Pushkal Katara*[1], Abhinav Gupta*[1], Harit Pandya[2], Y V S Harish[1], AadilMehdi Sanchawala[1], Gourav Kumar[3], Brojeshwar Bhowmick[3] and K. Madhava Krishna[1]

Abstract— **Recent data-driven approaches to visual servoing have shown improved performances over classical methods due to precise feature matching and depth estimation. Some recent servoing approaches use a model predictive control (MPC) framework which generalise well to novel environments and are capable of incorporating dynamic constraints, but are computationally intractable in real-time, making it difficult to deploy in real-world scenarios. On the contrary, single-step methods optimise greedily and achieve high servoing rates, but lack the benefits of the MPC multi-step ahead formulation. In this paper, we make the best of both worlds and propose a lightweight visual servoing MPC framework which generates optimal control near real-time at a frequency of 10.52 Hz. This work utilises the differential cross-entropy sampling method for quick and effective control generation along with a lightweight neural network, significantly improving the servoing frequency. We also propose a flow depth normalisation layer which ameliorates the issue of inferior predictions of two view depth from the flow network. We conduct extensive experimentation on the Habitat simulator and show a notable decrease in servoing time in comparison with other approaches that optimise over a time horizon. We achieve the right balance between time and performance for visual servoing in six degrees of freedom (6DoF), while retaining the advantageous MPC formulation. Our code and dataset are publicly available[†].**

I. INTRODUCTION

Visual servoing is one of the central problems in robotics which uses feedback information from the vision sensor for navigating the robot to a desired location. This involves generating a set of actions that move the robot in response to the observation from the camera, in order to reach a goal configuration in the world. The objective is to minimize difference between the features extracted from the current and the desired image. This objective is achieved by the visual servoing controller which iteratively minimises this error. Thus, feature extraction and controller designs are the vital modules for visual servoing approaches. Classically, hand-crafted features such as points [1], lines [2] and contours [3] were employed for visual servoing. However, such appearance based features result in inaccurate matching for larger camera transformations. To circumvent this bottleneck, recent data driven visual servoing approaches resort to deep neural features [4], [5]. Initial learning based approaches [4], [5] aim to learn the relative camera pose from a pair of images in a supervised fashion and as a result, they were able to achieve a sub-millimeter precision. However, as they were over-trained on a single environment, generalisation to new environments was difficult. Saxena et al. [6] trained their approach on multiple environments and thus, their approach generalised to a certain extent. But such approaches still require supervision and cannot be trained on-the-fly. Recently, a more principled approach for combining deep flow features with a classical image based controller was proposed in [7] to improve generalisation to novel environments without any retraining.

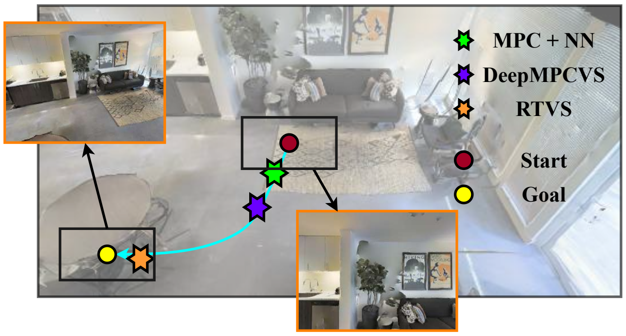


Fig. 1: RTVS shows a significant improvement in servoing frequency vis-à-vis other deep MPC based visual servoing approaches in 6DoF, without compromising its ability to attain precise alignments. Our controller generates optimal control in real-time at a frequency of 66 Hz (excluding optical flow overheads) and successfully servos to its destination, while other approaches still lag behind.

Controller design is another crucial aspect of visual servoing, classically, image based and position based controllers were presented in the literature. However, these controllers take the action greedily and do not consider a long term horizon, which could lead to problems such as the loss of features from field of view, getting stuck in local minima and larger trajectory lengths. Thus, to achieve a better controller performance, optimal control [8] and model predictive control [9] based formulations were proposed in the visual servoing domain. Another advantage of casting visual servoing in a model predictive control is to cater for additional constraints such as robot dynamics, field of view and obstacle avoidance. While the MPC formulations of visual servoing are lucrative, they assume accurate feature correspondences. Furthermore, they rely on classical MPC solvers that do not scale up to high dimensional features such as images

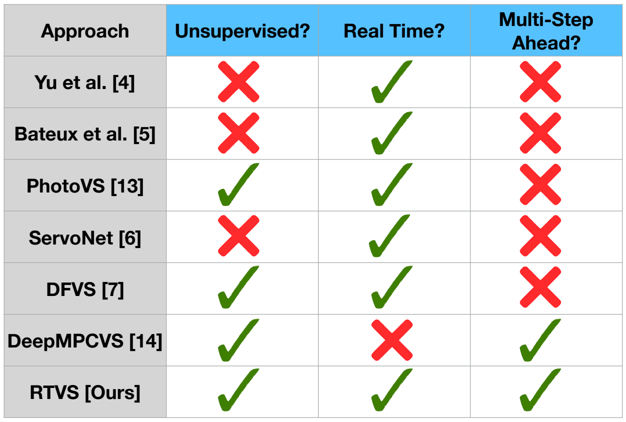
*indicates equal contribution.

[1] Robotics Research Center, IIIT Hyderabad, India

[2] Cambridge Research Laboratory, Toshiba Europe, UK

[3] TCS Research, Kolkata, India

[†] Project page: https://bonjovi1.github.io/RTVS/



Approach	Unsupervised?	Real Time?	Multi-Step Ahead?
Yu et al. [4]	✗	✓	✗
Bateux et al. [5]	✗	✓	✗
PhotoVS [13]	✓	✓	✗
ServoNet [6]	✗	✓	✗
DFVS [7]	✓	✓	✗
DeepMPCVS [14]	✓	✗	✓
RTVS [Ours]	✓	✓	✓

Fig. 2: Approaches ([4], [5], [6]), although real-time, are not multi-step ahead and require supervision. PhotoVS [13] and ServoNet [6] fail to converge on our benchmark, as shown in 6. DFVS [7] shows strict convergence in real-time, but optimises greedily and does not consider a long term horizon. DeepMPCVS [14] uses the advantageous MPC formulation and generates control in an unsupervised fashion, but is computationally expensive. Our approach makes the best of all worlds: it generates control in real-time and trains on-the-fly, whilst retaining the MPC formulation.

or flows, which makes it difficult for classical MPC based visual servoing approaches to be employed for modern deep features. Recently, several deep reinforcement based visual navigation approaches were proposed [10], [11] that present a neural path planning framework for visual goal reaching. By the virtue of being a model free reinforcement learning framework, they are sample inefficient, and do not present results in 6DoF planning. On the other hand, deep model based reinforcement learning frameworks [9], [12] aim to jointly learn controller as well as the underlying system dynamics (model) through data, thus making it difficult to generalise for new environments.

Katara et al. [14] use an approach which employs deep flow features and propose a kinematic model based on flow. For solving their deep MPC formulation, they further proposed a recurrent neural network (RNN) based optimisation routine, that generates velocities to optimise their flow based loss. As a baseline, they also employ a vanilla neural network instead of an RNN, for solving their MPC problem. While their proposed approach was able to solve the MPC problem on-the-fly in a receding horizon fashion and achieve convergence, it has costly overheads with respect to the total servoing time. DeepMPCVS [14] requires the training of a recurrent neural network online which is computationally expensive, narrowing down their scope of performance in real world scenarios. Single step methods like [7] are known to be faster in computing immediate control. However, these approaches do not include the benefits of the MPC optimisation formulation. We aim to make the best of both worlds, by improving the control generation frequency while retaining the advantageous MPC formulation.

In this paper, we extend and improve the work of [14] and present a lightweight MPC framework for real-time image based visual servoing (IBVS) which generates control in real-time and can be trained online in an unsupervised fashion. We leverage the deep flow-based MPC framework proposed in [14] and in order to make the approach faster and feasible for real-time systems, we present a novel lightweight *directional neural network* which aims to encode the relative pose between the current and final image in its parameters, thereby significantly reducing the number of iterations required in each MPC step as explained in II-B.2. Further, we utilise the differential cross entropy method (DCEM) proposed in [15] for differentiable sampling of control. Our controller can attain the desired pose around 10 times faster than the recent deep multi-step ahead formulations [14], with a 74% reduction in servoing time. This decrease in the servoing frequency leads to a significantly shorter delay between successive control commands, leading to minimal number of jerks, which is crucial for aerial robots. Fig. 2 showcases this boost obtained by our controller for an example scene in our benchmark, where our controller beats the recent deep MPC based approaches for reaching the goal. We compute immediate control at the same rate as single step methods like [7], which are also fast but lack the advantages of the MPC optimisation formulation and do not optimise over a time horizon. Our proposed architecture is sample-efficient and uses an optimal control generator network, which improves the servoing rate without a heavy compromise on its performance. Furthermore, our architecture also includes a flow normalisation layer, which reduces the error in flow depth estimates, thereby accounting for inaccurate flow predictions.

Our contributions are summarised as follows:

- We propose a lightweight and real-time visual servoing framework, which can be trained on-the-fly in an unsupervised fashion (Fig. 3) and improve the work of [14]. This work achieves a significant increase in the servoing frequency, computing control near real-time at 0.095 seconds per MPC iteration and is around 10 times faster than existing deep MPC based approaches (TABLE II).
- The decrease in time is made possible with the help of our control generator network which uses a slim and lightweight neural network that significantly reduces the number of iterations required in each MPC step. We use the differential cross-entropy method [15] which helps us sample intelligently and generate optimal control commands in 6DoF.
- Furthermore, we introduce a flow depth normalisation layer which accounts for inferior flow predictions from FlowNetC [16], thereby reducing our network's dependency on the accuracy of the flow (refer to Fig. 7).

II. Approach

Existing deep learning based MPC approaches like [14] formulate visual servoing as an MPC optimisation which consider a long term horizon and can be trained on-the-fly in an unsupervised fashion. However, [14] requires the online

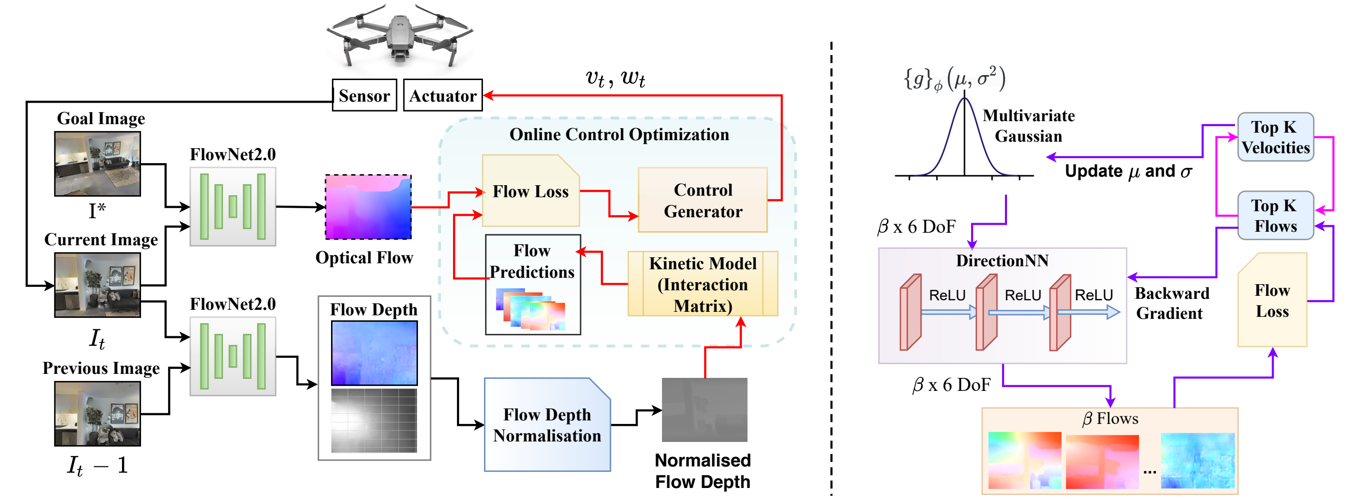


Fig. 3: **Left**: We demonstrate our MPC based optimisation framework to perform visual servoing in 6DoF. In each MPC optimisation step, we use FlowNetC to predict the optical flow between the current image I_t and goal image I_*, which acts as the target flow for control generation. The flow computed between the current and previous image acts as a proxy for depth, which after normalisation, is fed into the interaction matrix. Our control generator then optimises the flow loss to generate control commands. **Right**: This figure depicts our control generation architecture. We sample a (β x 6) vector from a multivariate Gaussian distribution and pass it through our Directional Neural Network. The kinetic model generates β pseudo-flows and the flow loss is computed for each, against the target flow. In each MPC step, we pick the top K velocities corresponding to the K least flow errors to update the Gaussian parameters, while the mean velocity is fed to the actuator.

training of an RNN, which is computationally expensive and faces challenges for real-time systems. In order to mitigate this issue, we propose a novel lightweight control generation architecture which effectively samples candidate velocities, leading to a significant improvement in the control generation time and hence, the servoing time. Our approach achieves the right trade-off between the servoing rate and performance (attainment of precise alignments) through an intelligent sampling strategy (Section II-B.1) and a slim neural network architecture trained on-the-fly (Section II-B.2). We compute the scene's depth Z_t as an inverted scale representation of the magnitude of optical flow between the current image I_t and previous image I_{t-1} as in [7], and pass it through the flow depth normalisation layer to generate effective scene-depth from optical flow, as described in (Section II-C). We illustrate our approach in Fig. 3 and summarise it through algorithm 1. In algorithm 1, the steps in red signify our modifications to the MPC algorithm proposed in [14] to make it feasible for real-time systems.

A. Problem Formulation

Given that the measurements of the robot sensor are monocular RGB images, with I^* being the desired image and I_t being the current observation at any time instant t, our aim is to generate optimal control commands $[v_t, \omega_t]$, where v_t is the linear velocity and ω_t is the angular velocity at time step t, in order to solve the fundamental IBVS objective of minimising the photometric error $e_t = \|I_t - I^*\|$ between I^* and I_t. We build our approach upon the kinetic model and MPC objective proposed in [14]. The MPC objective is given as

$$\mathbf{v}_t^* = \arg\min_{\mathbf{v}_t} \|\mathscr{F}(I_t, I^*) - \widehat{\mathscr{F}}(\mathbf{v}_t)\| \tag{1}$$

where $\mathscr{F}(I_t, I^*)$ is the target flow between I_t and I^* and $\widehat{\mathscr{F}}(\mathbf{v}_t)$ is the pseudo-flow generated through the predictive model:

$$\widehat{\mathscr{F}}(\mathbf{v}_{t+1:t+T}) = \sum_{k=1}^{T}[L(Z_t)\mathbf{v}_{t+k}] \tag{2}$$

where $L(Z_t)$ is the interaction matrix which relates the rate of change of features in the camera plane to the image plane, and $\mathbf{v}_t$ is the optimal control. The image coordinates x, y and Z_t being the scene's depth, the interaction matrix is generated as

$$L(Z_t) = \begin{bmatrix} -1/Z_t & 0 & x/Z_t & xy & -(1+x^2) & y \\ 0 & -1/Z_t & y/Z_t & 1+y^2 & -xy & -x \end{bmatrix}. \tag{3}$$

In [14], the online trajectory optimisation exploits the additive nature of optical flow and employs a recurrent neural network to generate velocity commands, which intuitively, tries to optimise over a fixed time-horizon in each MPC optimisation step. The approach tries to solve each time-step individually, exploiting the recurrence property of LSTM architectures which are computationally expensive and data-hungry. We modify the predictive model shown in eq. 2 to achieve an optimal trade-off between speed and accuracy, by connecting the MPC objective (which is to minimise the error between the predicted flow and target flow) with a differential sampling based strategy through a slim and lightweight neural network. We introduce a *horizon* parameter 'h' in

Algorithm 1 Visual Servoing MPC Framework with Fast Control Generation Optimisation.

```
Require: $I^*$, $\varepsilon$                                    ▷ Goal Image, convergence constant
Initialise $\mu$, $\sigma^2$                                      ▷ Initialise Gaussian Distribution sampling parameters
while $\|I - I^*\| \leq \varepsilon$ do                           ▷ Convergence criterion
    $I_t$ := get-current-obs()                                    ▷ Obtain the current RGB observation from sensor
    predict-target-flow ($\mathscr{F}(I_t, I^*)$)                 ▷ Predict target flow using flow network
    $L_t$ := compute-interaction-matrix($\mathscr{F}(I_t, I_{t-1})$)   ▷ Kinetic Model Generation
    for $m = 0 \rightarrow M$ do                                  ▷ On the fly training for DirectionNN
        $[\beta_i]_{i=1}^{N} \sim g_\phi(\mu, \sigma^2)$           ▷ Sampling a $\beta$ x 6 vector from Gaussian Distribution
        $[\mathbf{v}_{t,i}]_{i=1}^{N} = f_\theta([\beta_i]_{i=1}^{N})$   ▷ Predict $\beta$ velocities from DirectionNN
        $[\widehat{\mathscr{F}}(\mathbf{v}_{t,i})]_{i=1}^{N} = [L_t(Z_t)\mathbf{v}_{t,i}]_{i=1}^{N}$   ▷ Generate $\beta$ pseudo-flow using predictive model
        $[\mathscr{L}_{flow}]_{i=1}^{N} := [\|\widehat{\mathscr{F}}(\mathbf{v}_{t,i}) - \mathscr{F}(I_t, I^*)\|]_{i=1}^{N}$   ▷ Compute Flow Loss
        $\theta_{m+1} := \theta_m - \eta \nabla \mathscr{L}_{flow}$      ▷ Update DirectionNN parameters
        $\mu_{t+1} = 1/k \sum_i^k v_{t,i}$                        ▷ Update mean of the Gaussian Distribution
        $\sigma_{t+1}^2 = 1/k \sum_i^k (\mu_{t,i} - \mu_{t+1})^2$  ▷ Update standard deviation of the Gaussian Distribution
    end for
    $\widehat{\mathbf{v}}_{t+1} := \arg\min_{v \in V} \mathscr{L}_{flow}$   ▷ Execute control command in the environment
end while
```

order to scale the predicted flow to learn to predict the mean optical flow, rather than predicting over a time-horizon.

$$\widehat{\mathscr{F}}(\mathbf{v}_t) = h * L(Z_t)(\mathbf{v}_t) \tag{4}$$

Thus, we formulate our loss function as shown in eq. 5 to train our control generation network on-the-fly.

$$\mathscr{L}_{flow} = \|\widehat{\mathscr{F}}(\widehat{\mathbf{v}}_t) - \mathscr{F}(I_t, I^*)\| = \|[L(Z_t)\widehat{\mathbf{v}}] - \mathscr{F}(I_t, I^*)\| \tag{5}$$

We regress the pseudo flow $\widehat{\mathscr{F}}(\mathbf{v}_t)$ with the target flow $\mathscr{F}(I_t, I^*)$ using a mean squared error loss. We summarize the adaptive sampling and network training process of our control generation architecture in Section II-B. After each MPC optimisation step, we apply the velocity to the agent and get the new measurements from the sensor, repeating the optimisation process for each subsequent step until the IBVS objective is met. We summarize the MPC optimisation process through Fig. 3.

B. Optimal Control Generation Architecture

In this section, we provide details about the differential sampling-based strategy to generate optimal control and our lightweight neural network architecture.

1) Intelligent DCEM-based Sampling: Due to the high dimensionality of the flow representations, it becomes difficult for classical MPC solvers to optimise the objective function in equation 1. The cross entropy method (CEM) [17] is an attractive formulation to solve complex control optimisation problems involving objective functions which are non-convex, deterministic and continuous in nature by solving the equation,

$$\hat{F} = \underset{F}{\arg\min} \mathscr{E}_v(F) \tag{6}$$

where $\mathscr{E}_v(F)$ is the objective function having parameters v over the n-dimensional space. However, there are a few shortcomings to using this approach. It is a gradient-free optimisation approach where Gaussian parameters are refitted only in a single optimisation step, which does not allow adaptive sampling over consecutive MPC steps i.e. it is unable to preserve the memory of effective sampling. Moreover, the parameter optimisation is not based on the downstream task's objective function which might lead to a sub-optimal solution, which in our case would lead to an increase in the number of MPC steps, directly affecting the servoing time. In this work, we take inspiration from the differential cross entropy method (DCEM) [15] which parameterises the objective function $\mathscr{E}_v(F)$, making it differentiable with respect to v. We introduce non-linearity in our control generation process with the addition of a neural network (II-B.2), in order to retain information throughout all MPC steps. This connects the sampling strategy with the objective function, making CEM an end-to-end learning process. Hence, the Gaussian parameters are updated with subsequent MPC optimisation steps, enabling adaptive sampling over the process based on the MPC objective. This allows us to sample from a latent-space of more reasonable control solutions.

2) DirectionNN: We introduce a slim neural network called the 'DirectionNN', whose design aims to encode a sense of relative pose between the current image I_t and the goal image I_*. We keep the network lightweight in order to achieve a high servoing rate and incorporate online learning of relative pose updates in each MPC optimisation step. Since we execute small steps in each iteration, the network is capable of learning the change in relative pose quickly. Moreover, the weights of the network are also retained in each MPC iteration, which can be reused since there is a minimal change in the agent's image measurements.

Multiplying with the horizon h (eq. 4) checks the merit of direction predicted by the DirectionNN over an extended time. Hence, we can train our approach for lesser number of iterations in each MPC step, which significantly decreases the total servoing time.

The network architecture has an input layer consisting of 6 neurons followed by 4 fully connected hidden layers with 16, 32, 256, 64 neurons respectively and an output layer with 6 neurons, which represents the 6DoF velocity vector. We apply ReLU activation at the output of each hidden layer and Sigmoid activation on the last layer. We further scale the 6-D output between -1 and 1 to vectorise it as a 6DoF velocity vector. We train the network in each MPC step. The inputs to the network are intelligently sampled from a Gaussian distribution.

3) Sampling and Learning: We use the DCEM sampling strategy explained in II-B.1 along with DirectionNN explained in II-B.2 to generate optimal control. In each MPC optimisation step, the network samples a β (batch size) x 6 dimensional vector from a Gaussian distribution $g_{\phi}(\mu, \sigma^2)$ and carries out a forward pass, generating β samples of 6 DoF velocity commands. We compute the pixel-wise target flow $\mathscr{F}(I_t, I^*)$ between I_t and I^* using a pretrained FlowNetC [16] model. Moreover, we apply a kernel with a filter size of (7x7) and a stride of 7 to the target flow $\mathscr{F}(I_t, I^*)$ and pseudo-flow $\widehat{\mathscr{F}}(\mathbf{v}_t)$. The kernel K, consists of only one non-zero value with K[0, 0] = 1.

The weights of the DirectionNN are updated through gradient descent. We use the Adam optimiser with a learning rate of 0.005 to train our network. The sampling parameters of the Gaussian distribution $g_{\phi}(\mu, \sigma^2)$ are optimised for subsequent steps. We update (μ, σ) by the mean and variance of top K velocities corresponding to the top K least flow errors. For our approach, we sample 8 velocities and compute the flow loss for each, and choose the velocity corresponding to the least flow loss, which is applied to our agent. We also multiply the horizon parameter with the generated velocity before computing the flow loss. The MPC optimisation steps are repeated until the convergence criteria to reach the goal location is met.

C. Two View Depth Normalisation

We further enhance the two view depth estimation method proposed in [7] to account for inferior flow predictions, since we use FlowNetC [16] without any retraining/fine-tuning. The magnitude of flow values predicted by [16] has high variance from pixel to pixel due to the non-planar structure of the scene and as a result, there are heavy outliers while using the predicted flow as a proxy for depth. These outliers can hurt the performance of the controller, because the magnitude of velocity is very sensitive to depth values. Hence, we require a stable flow depth to guide our approach. We use the sigmoid function (equation 7) to scale and normalise the flow values before feeding them to our kinetic model - the interaction matrix. We compare the effect of flow normalisation on the mean squared error for three scenes, as shown in TABLE I.

$$Z_s(x,y) = \nu\left(\frac{1}{1+e^{-Z}} - 0.5\right) \qquad (7)$$

Here, Z is the two view depth proposed in [7], ν is the scaling factor which we have selected as 0.4 and Z_s is the scaled inverse used as proxy for depth in the interaction matrix.

Scene	MSE Flowdepth	MSE Normalised Flowdepth
Quantico	160.04	49.72
Arkansaw	469.72	63.44
Ballou	508.81	122.78

TABLE I: Comparison of effect of flow normalisation on MSE for different scenes. A significant decrease in Mean Squared Error is observed when the flowdepth is normalised

III. Experiments

The motivation behind our work is to present an online control generation strategy that can generate optimal 6DoF robot commands on-the-fly and in real-time, while not compromising on performance in terms of convergence and alignment. We benchmark our strategy on 8 indoor 3D photo-realistic baseline environments from the Gibson dataset [18] in the Habitat simulation engine [19] similar to [7]. These baseline environments span various levels of difficulty based on parameters such as the extent of overlap, the amount of texture present and the rotational and translational complexities between initial and desired image. We use a free-flying RGB camera as our agent, which can navigate in all six degrees of freedom.

Through this section, we show quantitative and qualitative results to validate our approach. This work generates control at 0.015 seconds per MPC step (excluding flow overheads) and attains photometric convergence faster than the other methods which optimise over a time horizon. While achieving a significant improvement in servoing rate, our approach does not compromise on performance in terms of pose and photometric errors, which is comparable to the established baselines. Our method makes a sacrifice on its trajectory length, but generates control in real-time and achieves precise alignments, thereby finding the right trade-off between speed and performance.

We compare our approach with (a) DeepMPCVS [14] and (b) MPC+NN [14], both of which are MPC formulations that optimise over a time horizon. The approach in [14] is optimal but computationally heavy due to its bulky architecture. We achieve a significant improvement in servoing rate vis-à-vis these baselines. Approaches like [7] are fast in computing the immediate control, but they optimise greedily and cannot incorporate additional constraints. We achieve similar servoing rates with such single-step approaches as well.

A. Convergence Study and Time Comparison

We test our approach across all environments in our simulation benchmark and report the per MPC-step time (per IBVS-step time in case of [7]) with and without overheads from flow computation, the number of steps taken for convergence and the total time for a visual servoing run to reach the goal image, averaged over all scenes as shown in TABLE II. Our lightweight control generation architecture

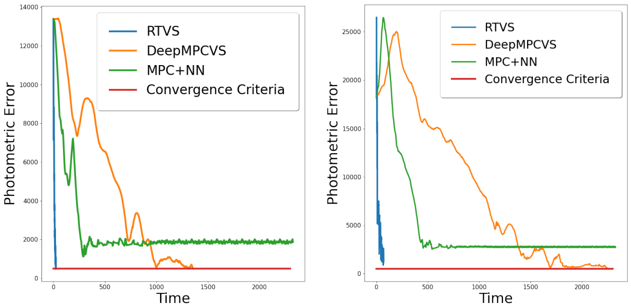


Fig. 4: **Time Comparisons**: We obtain a significant improvement in servoing rate over other deep MPC based visual servoing approaches. Here, we show the evolution of photometric error with time on two environments. Our approach is the fastest to achieve convergence, without any compromise on the convergence criteria.

and intelligent sampling strategy help achieve a per MPC-step time of 0.015 seconds (this is the time taken for the MPC optimisation step, excluding the overheads from the flow network). We only train for 1 iteration in every MPC step (as opposed to 100 required in [14]) and retain the weights of our neural network, since there is no substantial change in the velocities for immediate MPC steps, which helps lower the per MPC-step time. The entire time taken for a MPC-step, after including flow overheads, is 0.095 seconds, resulting in a near real-time control generation at a frequency of 10.52 Hz. We attain strict convergence of photometric error <500 and outperform the other multi-step ahead approaches [14] in terms of the total servoing time. We use a *Nvidia Geforce GTX 1080-Ti Pascal* GPU for our experimentation.

B. Qualitative Results

While we achieve a significant improvement in time, our approach does not compromise on photometric convergence. We test out our approach across all environments in our simulation benchmark. Our controller successfully achieves convergence and is able to servo to the desired location. The control actions are learnt unsupervised and the network is trained on-the-fly in an unsupervised fashion. We attain a strict photometric error of strictly <500 in all scenes and report the photometric error representation computed between the attained and the desired image, in Fig. 6.

C. Pose Error and Trajectory Lengths

We perform more quantitative tests and report the initial pose error, translation error(T. Error), rotational error(R. Error) and trajectory length(Traj. Length) averaged out over all environments across our simulation benchmark, as depicted in TABLE III. We achieve very low pose errors which is comparable to the other 6DoF servoing approaches. Our approach does not compromise on this and achieves precise alignments with high servoing frequency. We are able to capture a strong correlation between the photometric error and the flow loss, and a steady decrease in the velocities, as depicted in Fig. 5(Left).

Approaches	Time w.o flow	Time w. flow	Total Iters.	Total Time
MPC+NN [14]	0.75	1.10	344.22	378*
DFVS [7]	**0.001**	0.21	994.88	209
DeepMPCVS [14]	0.8	1.15	**569.63**	655
RTVS [Ours]	0.015	**0.095**	1751.25	**166**

TABLE II: **Quantitative Comparison**: We compare our approach with other deep MPC based methods from [14] and a single-step approach [7]. Apart from MPC+NN [14], all methods attain convergence on the simulation benchmark. We report the time per servoing iteration excluding overheads (Time w.o. flow), the time per servoing iteration including flow computations (Time w. flow), the average number of iterations taken to reach convergence (Total Iters.) and the average time required to servo to the destination, including overheads (Total Time). DeepMPCVS [14] is optimal and requires the least number of iterations, but has a costly per-iteration time overhead. Our method has a very low per MPC-step compute time and takes the least amount of time to attain convergence, comparable to the performance of DFVS [7]. All times are given in seconds. (* means does not converge in some scenes.)

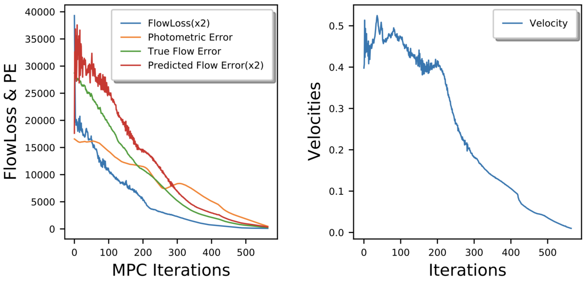


Fig. 5: **Left**: All the important errors are being reduced concurrently. The drop in FlowLoss indicates that our network can accurately predict flows as the number of MPC iterations increase. The drop in flow errors and photometric errors indicates that the agent is able to reach near the goal. **Right**: The magnitude of velocity reduces as we near the goal, resulting in a smooth and stable convergence.

Approaches	T. Error (m)	R. Error (deg.)	Traj. Length (meters)
Initial Pose Error	1.6566	21.4166	-
MPC+NN* [14]	0.1020	2.6200	**1.1600**
DeepMPCVS [14]	0.1200	**0.5500**	1.1800
RTVS [Ours]	**0.0200**	0.5900	1.732

TABLE III: **Quantitative Comparison**: We compare the average performance in terms of pose error and trajectory lengths for different approaches across all environments in our benchmark.(* means does not converge in some scenes.)

Our trajectory lengths are slightly inferior to [14], but

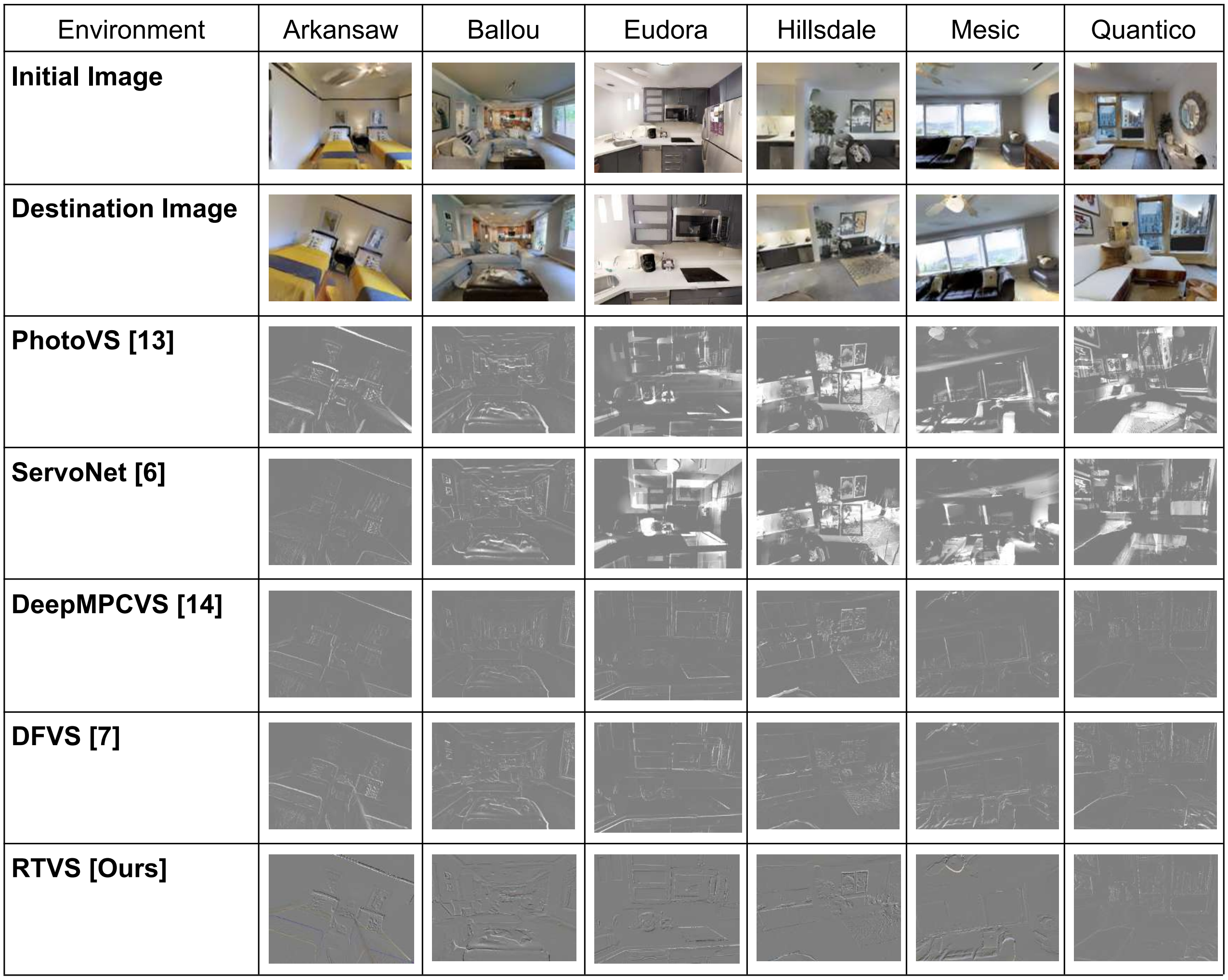


Fig. 6: **Qualitative Results**: Our controller successfully achieves convergence with a strict photometric error of <500 across all environments. Here, we show the photometric error image representation computed between the goal and attained images on termination for six environments in the simulation benchmark. PhotoVS [13] and ServoNet [6] fail to converge even with large number of iterations, thus showing large photometric errors. DFVS [7] successfully converges, but is a single-step approach that does not consider a long term horizon. DeepMPCVS [14] also meets the photometric convergence criteria, but is extremely slow since the online training of its RNN architecture is computationally expensive. Our work achieves strict convergence with control generated in real-time at a frequency of 66 Hz (excluding flow overheads), whilst optimising over a receding horizon.

shorter than single-step approaches such as [7], which signifies the importance of our control optimization steps. We compare the trajectory followed by our agent with other multi-step ahead approaches, as depicted in Fig. 7(left). Making a slight compromise on the trajectory length, we achieve superior servoing frequency as compared to other time-consuming approaches like [14], thereby achieving the right trade-off between servoing rate and performance.

D. Generalisation to Real-World Scenarios

We were unable to verify our approach for real-world scenarios due to Covid restrictions at our university and inability to access lab hardware. Previous approaches like [7] which have a similar run-time as our work and were tested on the same simulation environment [19], have shown successful deployment on drones in the real world. We achieve similar a servoing rate as [7] and are confident that our approach can also be deployed to real-life drones. In order to simulate a real world setup, we perform tests with actuation noise.

Robot commands are generally noisy in a real-world setup. In order to simulate such conditions, we add a Gaussian noise with μ=0 and σ=0.1 (m/s for translational and rad/s for rotational) to the control commands in Habitat in all six degrees of freedom, before applying it to the agent. We test this out across all scenes in our benchmark and successfully achieve convergence to a photometric error of <500. Our method adapts well and converges in an average number

Approaches	Time Per MPC-step	MPC Steps	Total Time
RTVS [Noiseless]	0.095	1774.28	168.53
RTVS [Induced Noise]	0.095	1850.22	175.77

TABLE IV: **Actuation Noise**: We achieve convergence even in the presence of actuation noise, thereby demonstrating the ability of our controller to adapt to a real world setup. All times are in seconds.

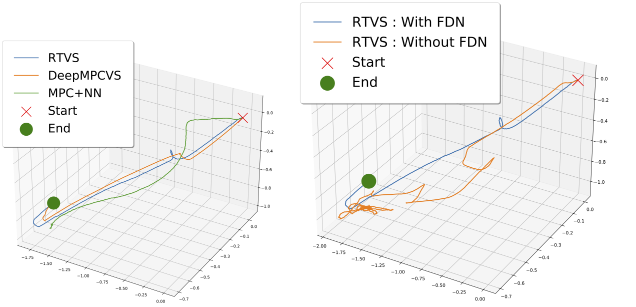


Fig. 7: **Left:** We plot the trajectories for the various approaches in our benchmark. **Right:** Our lightweight controller generates optimal velocity commands and is able to servo to the desired location in the presence of FDN (flow depth normalisation), which effectively handles inaccuracies in the flow predictions. Grid size in the plots is (0.2m X 0.2m X 0.2m).

of 1850.21 MPC steps, which is 4.28% more than those required in a noiseless setup as depicted in TABLE IV. Our approach is thus capable of handling actuation noise and generalising in real-world experiments.

E. Evaluating Flow Depth Normalisation

Since we use FlowNetC [16] without any retraining/finetuning, using a lightweight architecture is vulnerable to inaccuracies from the flow network. To counter this, we have proposed the flow depth normalisation layer. As depicted in Fig. 7(Right), our architecture is unable to achieve convergence and reach its destination when the flow depth is not normalised. With this layer in place, the flow is stabilised and we achieve robust performance in the servoing rate as well as accuracy.

IV. CONCLUSION

In this work, we have significantly improved the MPC algorithm in [14] by putting forth a lightweight and fast model predictive control framework that generates control near real-time at a frequency of 10.52 Hz. We have demonstrated a significant improvement in the total servoing time as compared to other visual servoing approaches in 6DoF. We showcase the efficiency of our control generation architecture, which uses a slim neural network architecture and an effective sampling strategy to generate optimal control in real-time, without making a heavy compromise on its performance. Our controller's ability to train in an online fashion helps it generalise and adapt well to novel environments. Our work is, to the best of our knowledge, the fastest deep MPC based approach to visual servoing in six degrees of freedom.

V. ACKNOWLEDGEMENT

This work was supported by a research grant from Tata Consultancy Services (TCS) Research, India.

REFERENCES

[1] P. I. Corke, "Visual control of robot manipulators–a review," in *Visual Servoing: Real-Time Control of Robot Manipulators Based on Visual Sensory Feedback*. World Scientific, 1993, pp. 1–31.

[2] N. Andreff, B. Espiau, and R. Horaud, "Visual servoing from lines," *The International Journal of Robotics Research*, vol. 21, no. 8, pp. 679–699, 2002.

[3] E. Malis, G. Chesi, and R. Cipolla, "212d visual servoing with respect to planar contours having complex and unknown shapes," *The International Journal of Robotics Research*, vol. 22, no. 10-11, pp. 841–853, 2003.

[4] C. Yu, Z. Cai, H. Pham, and Q.-C. Pham, "Siamese convolutional neural network for sub-millimeter-accurate camera pose estimation and visual servoing," *arXiv preprint arXiv:1903.04713*, 2019.

[5] Q. Bateux, E. Marchand, J. Leitner, F. Chaumette, and P. Corke, "Training deep neural networks for visual servoing," in *IEEE ICRA*. IEEE, 2018, pp. 1–8.

[6] A. Saxena, H. Pandya, G. Kumar, A. Gaud, and K. M. Krishna, "Exploring convolutional networks for end-to-end visual servoing," in *IEEE ICRA*. IEEE, 2017, pp. 3817–3823.

[7] Y. V. S. Harish, H. Pandya, A. Gaud, S. Terupally, S. Shankar, and K. M. Krishna, "Dfvs: Deep flow guided scene agnostic image based visual servoing," in *IEEE ICRA*. IEEE, 2020, pp. 3817–3823.

[8] S. Bansal, V. Tolani, S. Gupta, J. Malik, and C. Tomlin, "Combining optimal control and learning for visual navigation in novel environments," in *Conference on Robot Learning*. PMLR, 2020, pp. 420–429.

[9] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," 2016.

[10] 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," 2016.

[11] A. X. Lee, S. Levine, and P. Abbeel, "Learning visual servoing with deep features and fitted q-iteration," *arXiv preprint arXiv:1703.11000*, 2017.

[12] N. Hirose, F. Xia, R. Martín-Martín, A. Sadeghian, and S. Savarese, "Deep visual mpc-policy learning for navigation," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3184–3191, 2019.

[13] C. Collewet and E. Marchand, "Photometric visual servoing," *IEEE TRO*, vol. 27, no. 4, pp. 828–834, 2011.

[14] P. Katara, Y. V. S. Harish, H. Pandya, A. Gupta, A. Sanchawala, G. Kumar, B. Bhowmick, and K. M. Krishna, "Deepmpcvs: Deep model predictive control for visual servoing," *4th Annual Conference on Robot Learning (CoRL)*, 2020.

[15] B. Amos and D. Yarats, "The differentiable cross-entropy method," 2020.

[16] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," 2016.

[17] L.-Y. Deng, "The cross-entropy method: a unified approach to combinatorial optimization, monte-carlo simulation, and machine learning," 2006.

[18] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese, "Gibson env: Real-world perception for embodied agents," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9068–9079.

[19] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A Platform for Embodied AI Research," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.