Tracking and Planning with Spatial World Models

Barış Kayalıbay[*]   BKAYALIBAY@ARGMAX.AI
Atanas Mirchev[*]   ATANAS.MIRCHEV@ARGMAX.AI
Patrick van der Smagt
Justin Bayer   BAYERJ@ARGMAX.AI
Machine Learning Research Lab, Volkswagen Group, Munich

Editors: R. Firoozi, N. Mehr, E. Yel, R. Antonova, J. Bohg, M. Schwager, M. Kochenderfer

Abstract

We introduce a method for real-time navigation and tracking with differentiably rendered world models. Learning models for control has led to impressive results in robotics and computer games, but this success has yet to be extended to vision-based navigation. To address this, we transfer advances in the emergent field of differentiable rendering to model-based control. We do this by planning in a learned 3D spatial world model, combined with a pose estimation algorithm previously used in the context of TSDF fusion, but now tailored to our setting and improved to incorporate agent dynamics. We evaluate over six simulated environments based on complex human-designed floor plans and provide quantitative results. We achieve up to 92% navigation success rate at a frequency of 15 Hz using only image and depth observations under stochastic, continuous dynamics.
Keywords: navigation, planning, model-based control, state estimation, differentiable rendering

1. Introduction

Autonomous systems need to understand their environment to make good decisions. Optimal control and reinforcement learning therefore hinge on learning or engineering accurate models of both dynamics and observations, which allow us to plan into the future and find optimal actions. The paradigm of learning models has been successfully applied to Atari games (e.g. Kaiser et al., 2020), walking with complex simulated robots (e.g. Hafner et al., 2019), controlling unmanned aerial vehicles (e.g. Becker-Ehmck et al., 2020) and meta-reinforcement learning for robotics (e.g. Zhao et al., 2020).

Prior work has attempted to learn such world models of spatial environments (Fraccaro et al., 2018; Mirchev et al., 2019). These early models were limited to simple scenes, which restricted their practical use in control to toy sce-
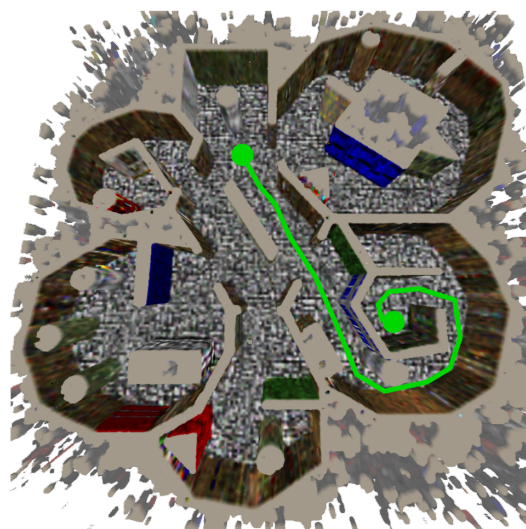

Figure 1: Planning in a world model. Isometric model view with planned trajectory.

---

[*] Equal contribution.

narios (Kayalibay et al., 2018). Their limitations were a result of attempting to model the complex interaction of a camera or depth sensor with its environment using a combination of neural networks and simple inductive biases. More recently, this balance has shifted in favour of the inductive bias, with an increased focus on engineering and domain knowledge (Mildenhall et al., 2020; Mirchev et al., 2021; Sitzmann et al., 2020; Müller et al., 2019). The result is the field of differentiable rendering, which models camera sensing by applying multiple-view geometry and adapting concepts from computer graphics, with neural networks being relegated to the relatively simple task of predicting properties like colour or occupancy over 3D space.

Differentiable rendering allows to model spatial environments with outstanding fidelity. It is then natural to use such models for control, an idea that was voiced as early as the 1990s (Oore et al., 1997). This has been done by Li et al. (2021) in the context of multi-link robots and Adamkiewicz et al. (2021) in the context of navigation, albeit without a quantitative evaluation of success.

Here we present a practical navigation algorithm for planar robots which uses a learned world model. Navigating based on visual observations alone requires inferring the agent's pose. We will therefore address the problem of state estimation as well. Our exact contributions are:

- We present a 15-Hz real-time algorithm for navigation under noisy agent dynamics and only local RGB-D observations in world models obtained with differentiable rendering.

- We test our method on simulated yet challenging indoor environments based on actual floor plans and provide a thorough quantitative evaluation.

- We improve an existing method for tracking camera poses by using the dynamics of the agent and apply it to our setting. This runs about five times faster than propagating gradients through the renderer, and is thus well-suited for vision-based control.

2. Related Work

Generative models and maps. Various generative models of space have been proposed (Fraccaro et al., 2018; Planche et al., 2019; Gregor et al., 2019; Mirchev et al., 2019). Among these, the work of Mirchev et al. (2019) was later adapted to a navigation task with the addition of A*-based planning (Kayalibay et al., 2018). The method was demonstrated on simple 2D mazes with $360°$ depth range observations. In contrast, others have proposed models aimed at tasks like navigation and exploration, which do not require generating new observations from the model. These approaches define read and write operations for distilling information from camera images into a 2D top-down map of the environment. Notable contributions here include the work of Parisotto and Salakhutdinov (2017), where a 2D neural map with abstract features is written to and read from to solve searching problems. Gupta et al. (2019) proposed a similar algorithm, extended by a planning module for navigation. The write operation was improved using projective geometry in (Chen et al., 2019) and (Chaplot et al., 2020), the latter achieving impressive navigation performance. Ramakrishnan et al. (2020) extended the setup by learning to predict occupancy beyond visible locations.

Differentiable rendering, pose estimation and control. Various methods for differentiably rendering images have been proposed in recent years. Of these, the method of Mildenhall et al. (2020) relies on modeling colour and spatial density with neural nets. Sitzmann et al. (2020) used an LSTM to find the point where a ray would hit the scene geometry. Lombardi et al. (2019) proposed using voxel grids and warping fields to encode and decode scenes. Mirchev et al. (2021) employ voxel

grids as well, focusing on localisation and mapping with unmanned aerial vehicles (UAV). Taking a different route, Niemeyer et al. (2020) and Yariv et al. (2020) rely on implicit differentiation to define the surface of the scene.

Several works deal with pose estimation in the context of differentiable rendering. Wang et al. (2021) jointly optimise pose, camera and scene parameters in a set of simple scenes. Assuming that the scene parameters have been learned in advance, Yen-Chen et al. (2021) propose finding camera poses using gradient descent and an efficient pixel subsampling scheme. Notably, Sucar et al. (2021) propose a real-time algorithm for joint tracking and mapping, though their evaluation is limited to small scenes. Mirchev et al. (2021) introduce a SLAM algorithm based on differentiable rendering, capable of tracking a UAV in photorealistically rendered simulations, though not in real time.

To the best of our knowledge, only two papers exist that apply differentiable rendering in a control context. Of these, (Li et al., 2021) focuses on manipulation tasks involving a robot arm. Adamkiewicz et al. (2021) investigate using neural radiance fields to solve navigation tasks. Their work is close to ours, though we differ in a couple of points. We focus on navigation with planar robots in indoor environments, and provide a quantitative evaluation on a set of complex scenes based on human-designed floor plans which were converted into levels for the Vizdoom simulator (Wydmuch et al., 2018). In contrast, they provide experiments with different kinds of robots, though they do not report any quantitative results indicating large-scale navigation success. Our work also approaches the problem of pose estimation differently. They approach pose estimation by combining the approach of Yen-Chen et al. (2021) with a transition loss and obtain a Bayesian filter. The benefit of our approach is its speed over computing gradients through the rendering pipeline. Finally, they propose a gradient-based algorithm for obstacle avoidance, which is initialised using A*-search. We similarly use A*-search, and track the planned trajectory using a simple low-level controller intended for a planar robot.

3. Background

3.1. Differentiable Rendering

The backbone of most differentiable rendering approaches is a parametric function that maps 3D space to geometric information (e.g. occupancy) and colour. Formally, we have two maps $f : \mathbb{R}^3 \to \mathbb{R}$ and $g : \mathbb{R}^3 \to \mathbb{R}^3$, where one learns the occupancy (alternatively, density or opacity) and the other corresponds to the RGB value. The colour model might also take a viewing angle as input, which allows modeling reflections. We can then generate an RGB-D image from any camera pose in the special Euclidean group $\mathrm{SE}(3)$ by defining a differentiable rendering procedure.

The first step of the rendering procedure involves finding the point where a ray intersects the scene geometry for the first time, the latter being implicitly defined by the occupancy model. We follow the approach of Mirchev et al. (2021), where the occupancy function is linearised around the point where it first exceeds a threshold, and the intersection point is then found using that linear approximation. This way of rendering is well-suited to our purposes, since it directly defines the occupancy of a point in space, which defines where the obstacles are when navigating. Formally, given a ray parameterised by the camera centre and an offset vector: $\mathbf{r} = \mathbf{c} + k\mathbf{v}$, we evaluate the occupancy function for $k \in \{\Delta, 2\Delta, \ldots, n\Delta\}$, where Δ gives the distance between two points along the ray, and $n\Delta$ is the maximum range of the camera. Then, if $\mathbf{p}^+ = \mathbf{c} + k^+\mathbf{v}$ is the first point for which the occupancy function returns a value above some threshold τ, and $\mathbf{p}^- = \mathbf{c} + k^-\mathbf{v}$

is the point that precedes it, the intersection point $\mathbf{p}$ is defined as $\mathbf{p}^* = \mathbf{c} + k^*\mathbf{v}$ for:

$$k^* = \alpha k^+ + (1 - \alpha)k^-, \quad \alpha = \frac{\tau - f(p^-)}{f(p^+) - f(p^-)}. \tag{1}$$

The setup of Mirchev et al. (2021) is also set apart from other differentiable rendering approaches as voxel grids are used instead of neural nets to capture the scene. Making use of trilinear interpolation, differentiability is maintained and hence the use of gradient-based techniques possible. We follow this decision as well, since indexing a voxel grid via interpolation is faster than evaluating a neural net. This reduces the time it takes to render a full image, a step required by our tracking method (we discuss rendering speed in section 5). We note that the voxel grid can be seen as an ensemble of small neural networks with partially fixed weights, where the networks are distributed over space in a grid-fashion, resembling the work of Reiser et al. (2021).

Formally the occupancy map is a 3D tensor $\mathcal{M}^{\text{occ}} \in \mathbb{R}^{h \times w \times d}$ and the colour map is a 4D tensor $\mathcal{M}^{\text{col}} \in \mathbb{R}^{h \times w \times d \times 3}$. The occupancy and colour functions f and g are then given by extracting the eight neighbor cells of the input point and trilinearly interpolating between them. The density of an RGB-D image $\mathbf{x}$ taken from the camera pose $\mathbf{z}$ under the differentiably rendered model is then:

$$p(\mathbf{x} \mid \mathbf{z}) = \prod_{i,j} p(\mathbf{x}_{ij} \mid \mathbf{z}, \mathcal{M}^{\text{occ}}, \mathcal{M}^{\text{col}}) = \prod_{\mathbf{x}_{ij} \in \mathbf{x}} \text{Laplace} \left(\begin{bmatrix} \mathbf{x}_{ij}^{\text{rgb}} \\ \mathbf{x}_{ij}^{\text{d}} \end{bmatrix} \middle| \begin{bmatrix} g(\mathbf{p}^*) \\ k^* \end{bmatrix}, \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix} \right). \tag{2}$$

Here, $\mathbf{x}_{ij}$ is a pixel from the image, and $\mathbf{x}_{ij}^{\text{rgb}}$ and $\mathbf{x}_{ij}^{\text{d}}$ are its RGB and depth readings respectively. $\mathbf{p}^*$ and k^* are the intersection point of the ray and its distance from the camera. The Laplace distribution's scales for colour and depth are parameterised by σ_1 and σ_2, which are treated as learnable parameters. The ray for a pixel is given by: $\mathbf{t}(\mathbf{z}) + d\mathbf{R}(\mathbf{z})\mathbf{K}^{-1} \begin{bmatrix} i & j & 1 \end{bmatrix}^T$, with $\mathbf{t}(\cdot)$ and $\mathbf{R}(\cdot)$ translation and rotation defined by the camera pose, $\mathbf{K}$ the camera intrinsic matrix, (i, j) the 2D coordinates of the pixel and d a depth reading.

This differentiable renderer was proposed by Mirchev et al. (2021) in the context of online simultaneous localisation and mapping. Here the map is learned offline on a set of RGB-D images with known camera poses. Given a dataset of images and camera poses $\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{z}^{(i)})\}_{i=1}^N$, the occupancy and colour parameters $\mathcal{M}^{\text{occ}}$ and $\mathcal{M}^{\text{col}}$ are obtained by gradient-based minimisation of:

$$\mathcal{L}(\mathcal{M}^{\text{occ}}, \mathcal{M}^{\text{col}}) = -\sum_{\mathbf{x},\mathbf{z} \in \mathcal{D}} \sum_{i,j} \log p(\mathbf{x}_{ij} \mid \mathbf{z}, \mathcal{M}^{\text{occ}}, \mathcal{M}^{\text{col}}). \tag{3}$$

We approximate the gradients by Monte-Carlo sampling images and pixels uniformly for speed.

3.2. Pose Estimation with Differentiable Renderers

Agents moving about freely normally cannot directly observe their state. Instead, they need to rely on estimating it from observations. Pose estimation in prior approaches with differentiable rendering has so far focused on backpropagating pixel-wise reconstruction errors through the renderer (Yen-Chen et al., 2021; Sucar et al., 2021; Adamkiewicz et al., 2021):

$$\arg\min_{\mathbf{z}} -\log p(\mathbf{x} \mid \mathbf{z}, \boldsymbol{\mathcal{M}}^{\text{occ}}, \boldsymbol{\mathcal{M}}^{\text{col}}).$$

We additionally consider a different route for state estimation, optimising both photometric and point-to-plane reprojection errors (Chen and Medioni, 1992). Given an RGB-D image $\hat{\mathbf{x}} = (\hat{\mathbf{x}}^{\mathrm{rgb}}, \hat{\mathbf{x}}^{\mathrm{d}})$ rendered by the model, a new observation $\mathbf{x} = (\mathbf{x}^{\mathrm{rgb}}, \mathbf{x}^{\mathrm{d}})$ and a relative pose $\mathbf{z}$ we minimise:

$$\arg\min_{\mathbf{z}} \sum_k \left\| \hat{\mathbf{x}}^{\mathrm{rgb}}[\pi(\mathbf{T_z}\mathbf{p}^k)] - \mathbf{x}^{\mathrm{rgb}}[\pi(\mathbf{p}^k)] \right\|_1 + \sum_k \left\| \langle \hat{\mathbf{p}}^k - \mathbf{T_z}\mathbf{p}^k, \hat{\mathbf{n}}^k \rangle \right\|_1 . \qquad (4)$$

Here, $\mathbf{p}^k$ is a 3D point in the observed camera frame[1] and $\hat{\mathbf{p}}^k$ is a 3D point in the camera frame of the model prediction, related via projective association (Blais and Levine, 1995; Stotko, 2016). π denotes perspective projection from 3D into the image plane, using the known intrinsic matrix $\mathbf{K}$. The dot product $\langle \cdot , \hat{\mathbf{n}}^k \rangle$ with the corresponding normal vector $\hat{\mathbf{n}}^k$ defines the point-to-plane objective. First, normals are computed based on image gradients of the rendered prediction $\hat{\mathbf{x}}^{\mathrm{d}}$. Then the difference $\hat{\mathbf{p}}^k - \mathbf{T_z}\mathbf{p}^k$ is projected onto the normal of the rendered surface, therefore reflecting the distance between the transformed point $\mathbf{T}\mathbf{p}^k$ and a hyperplane tangent to the surface. It drives points from the second camera frame to align to the surface implicitly defined by the rendered depth $\hat{\mathbf{x}}^{\mathrm{d}}$. The L1-norm (Laplace assumption) is used for increased robustness to outliers.

This method is commonly referred to as point-to-plane ICP (Chen and Medioni, 1992) combined with a photometric objective (Steinbrücker et al., 2011; Audras et al., 2011). Variants of this approach have been used to good effect for tracking new RGB-D observations w.r.t. a fused scene model (Newcombe et al., 2011; Nießner et al., 2013; Whelan et al., 2015). Traditionally, this is well-explored for maps obtained via TSDF fusion. We show the same concept is applicable to maps obtained by optimisation through differentiable rendering. In section 4.2 we will discuss its advantages compared to gradient descent directly through the renderer.

4. Method

4.1. Motion Planning

We focus on planar indoor robots trying to reach a goal position, the target. Given a starting location and target coordinates, we plan a trajectory using A^*-search in the space of 2D coordinates (*planning* in fig. 2, top). Here, we discretise the environment using a uniform grid. Moving from one cell to another is possible only if a) there is no occupied point on the line connecting the two cells according to the occupancy model, and b) the cell we step into maintains a minimum safety distance to any obstacle. For the former, we sample a fixed (e.g. one hundred) number of points along the movement vector. For the latter, we extract obstacles from the occupancy model by evaluating it on each grid point, where the occupied points then give us a set of point obstacles. This requires many evaluations of the map, which in our case is computationally cheap due to the voxel grid parameterisation of the map, in contrast to a neural net. The cost of stepping from one cell to the other is the length of the step and we likewise use the Euclidean distance to the target as our A^* heuristic.

Our A^*-search yields a shortest path to the target without satisfying any constraints on the agent's movement in this phase. To follow this path, given as a sequence of waypoints, while obeying the limitations of the agent's dynamics, we use a low-level controller (fig. 2, top right). We assume that the agent can rotate in-place up to a maximum angular velocity, and always moves in the direction of its current heading subject to some maximum velocity. This allows using a simple

---

1. In practice, we obtain $\mathbf{p}^k$ from pixel coordinates, depth d^k and the inverse intrinsic matrix $\mathbf{K}^{-1}$ in the observed image. We use it directly in eq. (4) for brevity.
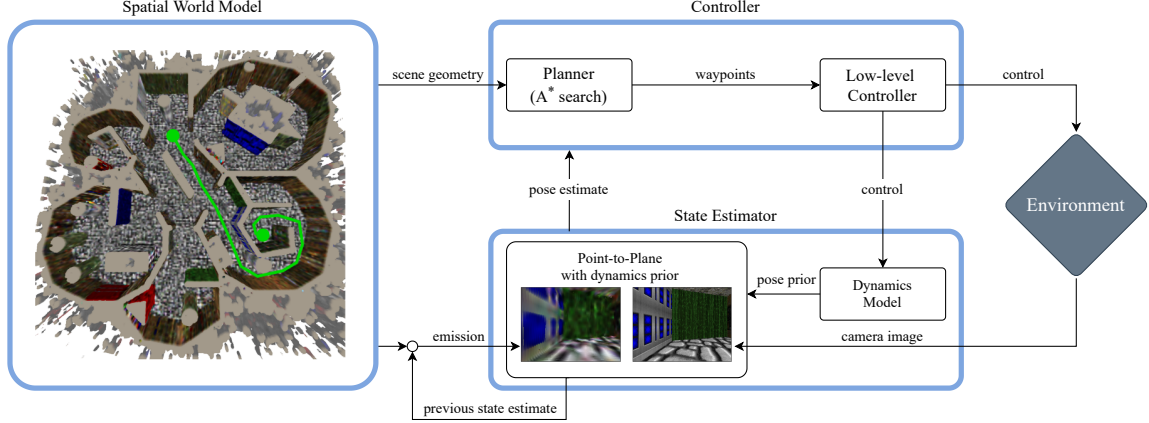
Figure 2: Overview of our method. The planner produces a set of waypoints based on the scene geometry learned by the world model. The low-level controller picks an action to move to the next waypoint. We estimate the agent's state by combining the agent's dynamics with the current camera image, which is aligned against an image predicted by the model.

low-level controller, which turns the agent at the nearest waypoint and then moves towards it. Our method can be applied to other systems, as long as a low-level controller capable of following the A* plan is available.

4.2. State Estimation

Our control rule requires us to continuously track the orientation and 2D location of the agent. We do so using a tracker that combines a transition model with the RGB-D images observed by the agent (*state estimator* in fig. 2). In the following, we will denote actions with $\mathbf{u}$, see section 5 for a description of the assumed noise model for the transition $p(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{u}_{t-1})$. Assuming a point-estimate of $\boldsymbol{\mathcal{M}}$ we can combine it with the transition into a state-space model:

$$p(\mathbf{z}_{2:T}, \mathbf{x}_{1:T} \mid \mathbf{u}_{1:T-1}, \mathbf{z}_1, \boldsymbol{\mathcal{M}}) = p(\mathbf{x}_1 \mid \mathbf{z}_1, \boldsymbol{\mathcal{M}}) \prod_{t=2}^{T} p(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{u}_{t-1}) p(\mathbf{x}_t \mid \mathbf{z}_t, \boldsymbol{\mathcal{M}}).$$

We are interested in the filtering posterior $p(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{x}_{1:t-1}, \mathbf{u}_{1:t-1}, \mathbf{z}_1, \boldsymbol{\mathcal{M}})$, shortened with $p_{\text{filter}}^t(\mathbf{z}_t)$:

$$\begin{aligned} p_{\text{filter}}^t(\mathbf{z}_t) &\propto p(\mathbf{x}_t \mid \mathbf{z}_t, \boldsymbol{\mathcal{M}}) p(\mathbf{z}_t \mid \mathbf{x}_{1:t-1}, \mathbf{u}_{1:t-1}, \mathbf{z}_1, \boldsymbol{\mathcal{M}}) \\ &= p(\mathbf{x}_t \mid \mathbf{z}_t, \boldsymbol{\mathcal{M}}) \mathbb{E}_{\mathbf{z}_{t-1} \sim p_{\text{filter}}^{t-1}(\cdot)} \left[p(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{u}_{t-1}) \right], \end{aligned} \tag{5}$$

for which we employ a maximum a-posteriori (MAP) approximation:

$$\begin{aligned} \arg\max_{\mathbf{z}_t} \ & \log p(\mathbf{x}_t \mid \mathbf{z}_t, \boldsymbol{\mathcal{M}}) + \log \left(\mathbb{E}_{\mathbf{z}_{t-1} \sim p_{\text{filter}}^{t-1}(\cdot)} \left[p(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{u}_{t-1}) \right] \right) \\ & \geq \log p(\mathbf{x}_t \mid \mathbf{z}_t, \boldsymbol{\mathcal{M}}) + \mathbb{E}_{\mathbf{z}_{t-1} \sim p_{\text{filter}}^{t-1}(\cdot)} \left[\log p(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{u}_{t-1}) \right] \\ & \approx \log p(\mathbf{x}_t \mid \mathbf{z}_t, \boldsymbol{\mathcal{M}}) + \log p(\mathbf{z}_t \mid \mathbf{z}_{t-1}^*, \mathbf{u}_{t-1}). \end{aligned} \tag{6}$$

The second line follows from Jensen's inequality. In the last line we approximate the expectation using the previous MAP estimate $\mathbf{z}_{t-1}^*$.

Integrating the agent's desired angular and forward velocities from the action $\mathbf{u}_{t-1}$, we can arrive at a prediction for the current state through $p(\mathbf{z}_t \mid \mathbf{z}_{t-1}^*, \mathbf{u}_{t-1})$, reflected in the second term of eq. (6). This prediction is imperfect due to noisy dynamics. We can refine it by comparing the agent's RGB-D observation against the map by optimising for $p(\mathbf{x}_t \mid \mathbf{z}_t, \mathcal{M})$. In the literature of differentiable rendering, this is typically done by minimising the difference between the RGB-D observation $\mathbf{x}_t$ and an image $\hat{\mathbf{x}}_t$ rendered from the pose variable $\mathbf{z}_t$ that is subject to optimisation. This is equivalent to maximising the log-likelihood term $\log p(\mathbf{x}_t \mid \mathbf{z}_t, \mathcal{M})$ above, and we will refer to it as emission-based tracking going forward.

While eq. (6) is well-aligned with the assumed generative model we follow (Mirchev et al., 2021), it is weighed down by the optimisation of $\log p(\mathbf{x}_t \mid \mathbf{z}_t, \mathcal{M})$ due to the high computational cost of propagating gradients through the renderer. We provide an empirical analysis of this aspect in section 5. With this in mind, we opt for one more approximation, substituting the first term in eq. (6) for the prediction-to-observation objective introduced in section 3.2:

$$\arg\min_{\mathbf{z}_t} \quad \sum_k \left\| \hat{\mathbf{x}}_{t-1}^{\mathrm{rgb}}[\pi(\mathbf{T}_{\mathbf{z}_t}^{\mathbf{z}_{t-1}^*} \mathbf{p}_t^k)] - \mathbf{x}_t^{\mathrm{rgb}}[\pi(\mathbf{p}_t^k)] \right\|_1$$
$$+ \sum_k \left\| \langle \hat{\mathbf{p}}_{t-1}^k - \mathbf{T}_{\mathbf{z}_t}^{\mathbf{z}_{t-1}^*} \mathbf{p}_t^k, \hat{\mathbf{n}}_{t-1}^k \rangle \right\|_1 - \log p(\mathbf{z}_t \mid \mathbf{z}_{t-1}^*, \mathbf{u}_{t-1}). \tag{7}$$

Here, $\mathbf{T}_{\mathbf{z}_t}^{\mathbf{z}_{t-1}^*}$ denotes the relative pose between step t and $t-1$ (going backward in time), which is a function of the optimised $\mathbf{z}_t$ and the previous MAP estimate $\mathbf{z}_{t-1}^*$. With that pose, we reproject points between $\hat{\mathbf{x}}_{t-1}^{\mathrm{rgb}}$, a rendered mean prediction of the RGB image at the previous time step, and the current RGB observation $\mathbf{x}_t^{\mathrm{rgb}}$. For the point-to-plane objective, $\mathbf{p}_t^k$ is a 3D point in the camera frame of the current observation, computed based on the observed depth $\mathbf{x}_t^{\mathrm{d}}$, and $\hat{\mathbf{p}}_{t-1}^k$ and $\hat{\mathbf{n}}_{t-1}^k$ are respectively the corresponding 3D point and normal in the previous camera frame, found through projective data association (Blais and Levine, 1995). The optimised states $\mathbf{z}_t$ are parameterised in the $\mathfrak{se}(3)$ Lie-algebra for the special Euclidean group $\mathrm{SE}(3)$, and we assume an L1 norm (Laplace assumption) for robustness to outliers. We uniformly sample a constant number of pixels for which the objective is evaluated, approximating gradients in expectation. We perform 100 steps of gradient descent for each new time step using the Adam optimizer (Kingma and Ba, 2015).

By substituting the rendering log-likelihood from eq. (6) for the photometric and point-to-plane loss terms in eq. (7), we maintain the necessary geometric constraints in addition to satisfying the assumed transition model. Note that in the revised objective new observations are still anchored to the map, as the projective transformation happens between a *rendered* prediction $\hat{\mathbf{x}}_{t-1}$ and the new incoming observation $\mathbf{x}_t$. The chosen objective reflects a conditional independence assumption $\mathbf{x}_t \perp\!\!\!\perp \mathcal{M} \mid \mathbf{x}_{t-1}, \mathbf{z}_{t-1}, \mathbf{z}_t$ between the map $\mathcal{M}$ and current observation $\mathbf{x}_t$ given a previous $\mathbf{x}_{t-1}$ and relative pose offset, which is a reasonable approximation for consecutive time steps. We opt for this tracking approach in favour of optimising through the emission model as it is faster and just as accurate in our setting, as we will show in experiments. To the best of our knowledge combining photometric, point-to-plane and dynamics constraints in one objective for state estimation has not been explored before for differentiable rendering.
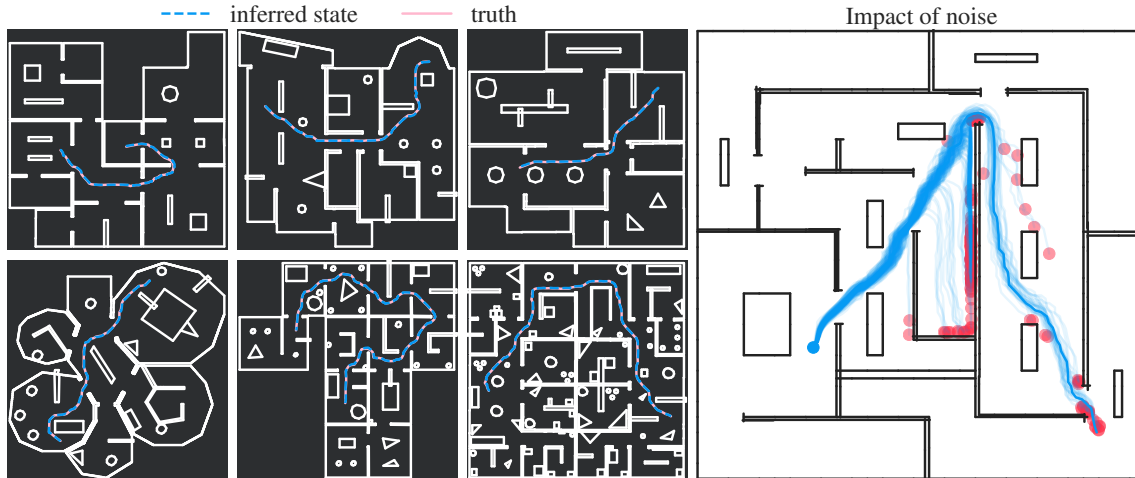
Figure 3: Evaluation levels with successful navigation runs (left) and the training maze (right). The solid blue line is a reference trajectory. The stochasticity of the dynamics is illustrated by showing multiple samples based on the same control inputs: since each trajectory ends up in a different place, state estimation and closed-loop control are crucial.

5. Experiments

We use the Vizdoom simulator for our quantitative experiments (Wydmuch et al., 2018). Despite its limited visual fidelity, Vizdoom allows us to focus on environments that are large and complex, i.e. with multiple rooms and realistic floor plans, as we can easily modify the simulation maps. To that end, we extracted six floor plans from the HouseExpo dataset Li et al. (2020), which we converted to Vizdoom levels with added obstacles. Figure 3 shows the evaluation environments, along with a training level taken from (Savinov et al., 2018). An agent body length of 0.2m, as suggested by Anderson et al. (2018), corresponds to an environment size of up to a $11.4 \times 11.4\text{m}^2$ sized square. A voxel map is learned for each environment from 5000 RGB-D images with pose labels. Our extended technical report describes the setup and models in detail (Kayalibay et al., 2022).

To analyze navigation performance under stochastic dynamics, we add noise to the agent motion in the simulator. Here, we make two considerations: noise should only be applied when the agent tries to move and it should not invert the direction of an action. The latter means that if the agent tries to rotate left, the noise should not make it rotate right. Likewise, if the agent tries to move forward, noise should not make it move backward. Guided by these principles, we add clipped Gaussian distributed noise to the angular velocity and speed, where the clipping ensures that the direction of turning (left/right) or movement (forward/backward) is not inverted.

Formally, the controls of the agent are $\mathbf{u} = (\dot{\alpha}, o, s)$, where $\dot{\alpha}$ is the angular velocity, o the angle of the movement direction and s the speed along that direction. The movement angle is distinct from the current heading angle α but must align with it up to a threshold of $5°$, and there are likewise maximum values defined for the angular velocity and speed at $11.5°$ and 0.8 times the agent's own body length. If we take l to be the 2D location of the agent, the noisy dynamics then amount to:

$$\alpha_{t+1} = \alpha_t + \text{sign}(\dot{\alpha}_t)\max(0, |\dot{\alpha}| + \epsilon_t^\alpha), \qquad \epsilon_t^\alpha \sim \mathcal{N}(0, \sigma_\alpha)$$
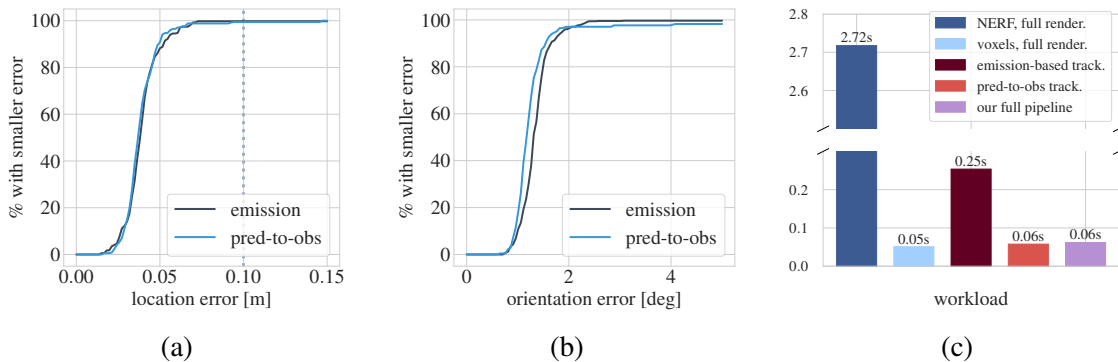
(a)   (b)   (c)

Figure 4: (a,b) CDF of the location & orientation RMSEs. The vertical line shows $0.5 \times$ agent_size. (c) Average runtimes of a single step for different computations: NeRF rendering, voxel map rendering, emission–based tracking, tracking with our scheme, tracking and control.

for the orientation and

$$l_{t+1} = l_t + \begin{bmatrix} \cos(\epsilon_t^\alpha + o_t) \\ \sin(\epsilon_t^\alpha + o_t) \end{bmatrix} \max(0, s_t + \epsilon_t^s), \;\; \epsilon_t^s \sim \mathcal{N}(0, \sigma_s)$$

for the location, where the angular velocity noise is added to the direction of movement and t denotes time. In addition to the noise, there is another source of error in the agent's dynamics, which results from the simulator. Vizdoom only allows for setting an integer angle in $[0°, 360°]$ degrees and the 2D location can only be specified up to two decimal points of precision. Together, these factors make for enough perturbation to pose a non-trivial challenge against navigation. This effect is illustrated in the right plot of fig. 3.

State estimation in real-time   First, we compare the tracker described in section 4.2 to pose estimation via gradient backpropagation through the rendering emission. We use $3°$ and 5% of the agent body length for the rotational and translational Gaussian scales of the noisy simulator dynamics. Figures 4(a) and 4(b) show the accuracy of the two approaches, displaying the percentage of trajectories with an RMSE smaller than the number on the x-axis. Both estimators have adequate performance, with a location RMSE smaller than 0.1m more than 99% of the time. Similarly, the orientation RMSEs are roughly on par.
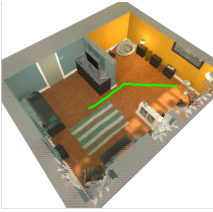
However, the main advantage of the state estimator from section 4.2 is its speed. As shown in fig. 4(c), a single step takes 0.06s (16.7Hz) on average, compared to 0.25s (4Hz) for the emission-based tracking. This is due to avoiding the computational cost of propagating gradients through the renderer (cf. section 4.2). We also note the major runtime difference when rendering from a NeRF map (Mildenhall et al., 2020) at 2.72s (0.4Hz) and from a voxel map (Mirchev et al., 2021) at 0.05s (20Hz) per image (fig. 4(c), left) respectively. The aim for on-line control made us employ the latter approach. The overall runtime of our pipeline is 0.062s (16.1Hz) per time step, with the bulk of it occupied by the state estimator. The proposed approach amounts to stable real-time tracking under noisy dynamics with only RGB-D observations. We believe this is a necessary component in the context of on-line control.

Navigation   Next we focus on our overarching objective – successful point-to-point navigation. Since our tracker works by combining transition estimates with RGB-D observations that are compared against the map, we conduct an ablation study to check how much each part is contributing.

9

noise	ours	no map	dynamics
high	**0.46**	0.37	0.33
mid	**0.79**	0.51	0.52
low	**0.92**	0.61	0.61

(a)

(b)

Figure 5: (a) Average SPL over six environments. Higher is better. **No map** tracks by optimising eq. (7) using the previous camera image instead of the model prediction. **Dynamics** merely integrates the dynamics. (b) Isometric views of the learned maps and successful navigation trajectories from AI2-THOR. The **low** noise setting was used.

Here, we compare our pipeline to a) tracking by using the transition model only, i.e. path integration; b) tracking by optimising a pose offset between two consecutive RGB-D observations (instead of using an emitted image from the map). We repeat our evaluation under three different levels of noise: **high** ($\sigma_a = 9°$, $\sigma_s = 30\%$ of agent size), **medium** ($\sigma_a = 6°$, $\sigma_s = 15\%$) and **low** ($\sigma_a = 3°$, $\sigma_s = 10\%$).

In our evaluation, we sample random pairs of starting positions and targets from the free space of the level, with the constraint that the start and target must be farther apart than three times the agent's body length. We consider a navigation attempt as successful if the agent's final position is closer to the target than two times the agent's body length, and use 200 navigation tasks per environment. The same navigation tasks are used for each noise level and method.

Our central evaluation metric is *success weighted by path length* (SPL) (Anderson et al., 2018), which is defined as:

$$\text{SPL} = \sum_{s_i \in \mathcal{S}} s_i \frac{l_i}{\max(p_i, l_i)}, \tag{8}$$

where $\mathcal{S}$ is a set of navigation tasks, s_i is a binary variable indicating the success of a task, and p_i and l_i are the length of the path taken by the agent and the length of the optimal path. We have found that SPL and success rate are almost the same for all three approaches. This indicates that successful navigation runs align well with the optimal trajectory.

The results of our evaluation can be found in fig. 5(a). We find that our proposed navigation approach performs best under all noise levels, yet it also degrades at higher levels of noise. Finally, we qualitatively demonstrate that our work extends to AI2-THOR (Kolve et al., 2017) in fig. 5(b), which is visually much more realistic than Vizdoom.

6. Conclusion

We have introduced a method for real-time navigation based on a differentiably rendered world model. Our approach is able to solve navigation tasks in complex environments using only RGB-D observations and under different levels of noise. We demonstrated that the state estimation problem can be solved by applying a point-to-plane metric to the differentiable rendering setting and combining it with a dynamics model, which allows for faster tracking compared to differentiating through the renderer. In future work, we will extend our method to environments with a higher visual fidelity and more complex dynamics.

References

Michal Adamkiewicz, Timothy Chen, Adam Caccavale, Rachel Gardner, Preston Culbertson, Jeanette Bohg, and Mac Schwager. Vision-only robot navigation in a neural radiance world, 2021. URL https://mikh3x4.github.io/nerf-navigation/assets/NeRF_Navigation.pdf.

Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir R. Zamir. On evaluation of embodied navigation agents, 2018.

Cedric Audras, A Comport, Maxime Meilland, and Patrick Rives. Real-time dense appearance-based slam for rgb-d sensors. In Australasian Conf. on Robotics and Automation, volume 2, pages 2–2, 2011.

Philip Becker-Ehmck, Maximilian Karl, Jan Peters, and Patrick van der Smagt. Learning to fly via deep model-based reinforcement learning. CoRR, abs/2003.08876, 2020. URL https://arxiv.org/abs/2003.08876.

Gérard Blais and Martin D. Levine. Registering multiview range data to create 3d computer objects. IEEE Transactions on Pattern Analysis and Machine Intelligence, 17(8):820–824, 1995.

Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam, 2020.

Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning exploration policies for navigation, 2019.

Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. Image and vision computing, 10(3):145–155, 1992.

Marco Fraccaro, Danilo Jimenez Rezende, Yori Zwols, Alexander Pritzel, S. M. Ali Eslami, and Fabio Viola. Generative temporal models with spatial memory for partially observed environments, 2018.

Karol Gregor, Danilo Jimenez Rezende, Frederic Besse, Yan Wu, Hamza Merzic, and Aaron van den Oord. Shaping belief states with generative environment models for rl, 2019.

Saurabh Gupta, Varun Tolani, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation, 2019.

Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, pages 2555–2565, 2019. URL http://proceedings.mlr.press/v97/hafner19a.html.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari, 2020.

Baris Kayalibay, Atanas Mirchev, Maximilian Soelch, Patrick van der Smagt, and Justin Bayer. Navigation and planning in latent maps, 2018. URL http://reinforcement-learning.ml/papers/pgmrl2018_kayalibay.pdf.

Baris Kayalibay, Atanas Mirchev, Patrick van der Smagt, and Justin Bayer. Tracking and planning with spatial world models. arXiv, 2022. URL https://arxiv.org/abs/2201.10335.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015. URL http://arxiv.org/abs/1412.6980.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. arXiv, 2017.

Tingguang Li, Danny Ho, Chenming Li, Delong Zhu, Chaoqun Wang, and Max Q. H. Meng. House-expo: A large-scale 2d indoor layout dataset for learning-based algorithms on mobile robots, 2020.

Yunzhu Li, Shuang Li, Vincent Sitzmann, Pulkit Agrawal, and Antonio Torralba. 3d neural scene representations for visuomotor control, 2021.

Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes. ACM Transactions on Graphics, 38(4):114, Jul 2019. ISSN 1557-7368. doi: 10.1145/3306346.3323020. URL http://dx.doi.org/10.1145/3306346.3323020.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020.

Atanas Mirchev, Baris Kayalibay, Maximilian Soelch, Patrick van der Smagt, and Justin Bayer. Approximate bayesian inference in spatial environments, 2019.

Atanas Mirchev, Baris Kayalibay, Patrick van der Smagt, and Justin Bayer. Variational state-space models for localisation and dense 3d mapping in 6 dof. In International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=XAS3uKeFWj.

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novk. Neural importance sampling, 2019.

Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In 2011 10th IEEE international symposium on mixed and augmented reality, pages 127–136. IEEE, 2011.

Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision, 2020.

Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. ACM Transactions on Graphics (ToG), 32(6):1–11, 2013.

Sageev Oore, Geoffrey E. Hinton, and Gregory Dudek. A mobile robot that learns its place. Neural Comput., 9(3):683–699, 1997. doi: 10.1162/neco.1997.9.3.683. URL https://doi.org/10.1162/neco.1997.9.3.683.

Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning, 2017.

Benjamin Planche, Xuejian Rong, Ziyan Wu, Srikrishna Karanam, Harald Kosch, YingLi Tian, Jan Ernst, and Andreas Hutter. Incremental scene synthesis, 2019.

Santhosh K. Ramakrishnan, Ziad Al-Halah, and Kristen Grauman. Occupancy anticipation for efficient exploration and navigation, 2020.

Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps, 2021.

Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation, 2018.

Vincent Sitzmann, Michael Zollhfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations, 2020.

Frank Steinbrücker, Jürgen Sturm, and Daniel Cremers. Real-time visual odometry from dense rgb-d images. In 2011 IEEE international conference on computer vision workshops (ICCV Workshops), pages 719–722. IEEE, 2011.

Patrick Stotko. State of the art in real-time registration of rgb-d images. In Central European Seminar on Computer Graphics for Students (CESCG 2016), 2016.

Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew J. Davison. imap: Implicit mapping and positioning in real-time, 2021.

Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. Nerf--: Neural radiance fields without known camera parameters, 2021.

Thomas Whelan, Stefan Leutenegger, R Salas-Moreno, Ben Glocker, and Andrew Davison. Elasticfusion: Dense slam without a pose graph. Robotics: Science and Systems, 2015.

Marek Wydmuch, Michał Kempka, and Wojciech Jaśkowski. Vizdoom competitions: Playing doom from pixels. IEEE Transactions on Games, 2018.

Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Ronen Basri, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance, 2020.

Lin Yen-Chen, Pete Florence, Jonathan T. Barron, Alberto Rodriguez, Phillip Isola, and Tsung-Yi Lin. Inerf: Inverting neural radiance fields for pose estimation, 2021.

Tony Z. Zhao, Anusha Nagabandi, Kate Rakelly, Chelsea Finn, and Sergey Levine. MELD: meta-reinforcement learning from images via latent state models. CoRR, abs/2010.13957, 2020. URL https://arxiv.org/abs/2010.13957.