Neural Autonomous Navigation with Riemannian Motion Policy

Xiangyun Meng, Nathan Ratliff, Yu Xiang and Dieter Fox

***Abstract*— End-to-end learning for autonomous navigation has received substantial attention recently as a promising method for reducing modeling error. However, its data complexity, especially around generalization to unseen environments, is high. We introduce a novel image-based autonomous navigation technique that leverages in policy structure using the Riemannian Motion Policy (RMP) framework for deep learning of vehicular control. We design a deep neural network to predict control point RMPs of the vehicle from visual images, from which the optimal control commands can be computed analytically. We show that our network trained in the Gibson environment can be used for indoor obstacle avoidance and navigation on a real RC car, and our RMP representation generalizes better to unseen environments than predicting local geometry or predicting control commands directly.**

I. INTRODUCTION

Creating autonomous robots capable of navigating in complex environments is an important research topic in robotics. Conventional autonomous robots require expensive sensors such as laser scanners to navigate, which only work in specific operating environments [1], [2], [3] due to physical and sensory constraints. In contrast, cameras are cheap, lightweight and carry rich geometric and semantic information. As a result, image-based navigation receives increasing attention recently [4], [5], [6]. Traditional image-based navigation systems are built upon simultaneous localization and mapping (SLAM) [7] techniques, where images are used to reconstruct a 3D world map, whereby a robot localizes and tracks itself. Although significant progress has been made [8], [9], [10], [11], these techniques still have trouble with textureless environments or fast-moving cameras.

By leveraging the recent advancements of deep neural networks, there exist two new paradigms for image-based autonomous navigation. The first paradigm applies deep learning to predict local geometry from images [12], [13], [14], [15]. The idea is to replace the traditional SLAM pipeline with a neural network in order to handle limitations of existing SLAM systems. Using the predicted geometry, traditional planning and control methods can be used for navigation. However, due to the complex mechanics of robots, the error in predicted geometry may result in unpredictable errors in the control commands, potentially causing catastrophic failures. The second paradigm trains a network to map images to control commands directly [16], [17], [18]. However, the lack of modelling the geometry and dynamics makes it difficult to interpret the model and also hinders its generalizability.

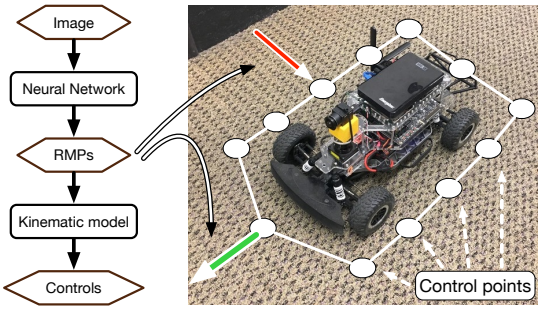


Fig. 1: We model the vehicle geometry with a set of control points and the interaction between control points and the environment with RMPs. The interaction can be repulsive (red) or attractive (green). We predict the RMPs from images with a neural network. The RMPs are then combined with the kinematic model of the vehicle to solve for the control commands.

In this paper, we introduce a new framework for image-based autonomous navigation by leveraging the recently proposed Riemannian Motion Policy (RMP) framework [19] and deep learning. RMP is a joint representation of the state of the robot and its environment. It models the interaction between a point on the robot and the environment through an acceleration policy with an associated Riemannian metric. By incorporating the kinematic model, the optimal control commands can be computed analytically. In other words, it unifies the robot geometry, robot dynamics and local obstacles into a single representation. We build this RMP structure into the design of a neural autonomous navigation framework as illustrated in Fig. 1. Our method has the following two main advantages: i) Our neural network trained to predict the RMPs generalizes better to unseen environments compared to networks trained to predict depth or control commands. ii) By examining the predicted RMPs, we are able to reason about the behavior of the vehicle precisely, achieving more explainable neural vehicle control.

We train our RMP controller in the Gibson environment [20], and evaluate our approach for indoor navigation on a RC car both in simulation and on a real hallway. Our test environments are real floorplans with presence of diverse obstacles and tight spaces, which requires precise maneuver with low tolerance for errors. In particular, the non-holonomic constraint of the vehicle makes steering in tight spaces challenging. We show that the RMP framework solves this problem elegantly and our model generalizes better than predicting control commands or predicting local geometry.

Xiangyun Meng is with Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, WA, USA xiangyun@cs.washington.edu

Dieter Fox is with NVIDIA, USA dieterf@nvidia.com and Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, WA, USA fox@cs.washington.edu

Nathan Ratliff, Yu Xiang are with NVIDIA, USA { nratliff, yux}@nvidia.com

II. RELATED WORK

Vision-based vehicle control and navigation. There has been persistent effort since [21] to map visual input to driving commands with a neural network. Thanks to the advancement of deep learning, recent works have come up with neural policies that achieved impressive driving performance. Most existing works [16], [17], [18] adopt a supervised approach to map images to control commands through imitation learning. The expert can be a human operator or a controller learned in advance using non-visual sensors. From this perspective, our approach is also supervised and the expert is a RMP controller. However, our controller differs from most existing works in that our neural network predicts a representation that models the vehicle geometry and dynamics explicitly.

[22] presents an approach that decouples perception and control by predicting a waypoint from a segmented image, whereby a PID controller is used to control the vehicle. However, it does not explicitly handle obstacles. Our work takes a step further by predicting RMPs from images. We demonstrate that our RMP representation is a principled approach for local obstacle avoidance and can potentially be used in conjunction with a waypoint predictor to achieve more robust visual autonomous navigation.

Another line of research adopts reinforcement learning (RL) [23], [24], [25] for self-supervised training. However, the behavior of reinforcement learning depends on the value function (e.g., the probability of collision), which only indirectly defines the behavior of the vehicle. This makes precise control and reasoning of a vehicle difficult. Existing works with RL on driving assume either fixed speed or routes which is impractical for real-world applications.

Model-based robot control. Model-based robot control assumes full observation of the environment. The complete knowledge of the geometric structure allows optimal local obstacle avoidance [19] and advanced planning algorithms to perform long-range trajectory optimization [26], [27]. However, the assumption that the geometric structure of the environment is known is unrealistic for a moving vehicle with a monocular RGB camera.

Our approach marries vision-based vehicle control with model-based robot control through the Riemannian metric representation of the robot and its environment. The vision system outputs a representation that can be used in the classical robot control framework. To this end, these two regimes are able to complement each other to overcome their weaknesses.

III. RMP MODELLING

In this section, we provide a brief introduction of RMP and its application to vehicle control. See [19] for a theoretical introduction in a multi-joint robotic arm setting.

A. A Brief Introduction of RMP

Consider a point agent $\mathbf{x}$ in $\mathbb{R}^n$ (usually $n = 2$ for a ground vehicle). Denote the position of the agent at time t as $\mathbf{x}(t)$. A motion policy is a mapping $\mathbf{f} : \mathbf{x}(t), \dot{\mathbf{x}}(t) \mapsto \ddot{\mathbf{x}}(t)$ that maps position and velocity of the agent to an acceleration. The agent applies this acceleration for a small time step to reach a new state $\mathbf{x}(t+1), \dot{\mathbf{x}}(t+1)$, whereby a trajectory can be generated through forward integration.

In autonomous navigation tasks, we would like to have a motion policy that guides the agent to its destination $\mathbf{g} \in \mathbb{R}^n$ while avoiding obstacles along the way. To achieve this goal, we model obstacles as a set of points $\mathbf{o}_1, ..., \mathbf{o}_m \in \mathbb{R}^n$, which are directly measurable using commodity sensors such as a laser scanner or a depth camera. Then the problem becomes designing a motion policy that keeps the agent away from the obstacles $\mathbf{o}_1, .., \mathbf{o}_m$ while moving towards the goal $\mathbf{g}$.

An intuitive way to design this motion policy is to divide it into a set of policies that model the interaction between the agent and each goal or obstacle point. For example, for each obstacle, we assign a policy that produces a repelling acceleration towards the agent, whereas for the goal point the policy produces an attractive acceleration. The optimal motion policy can be solved in a least-squares manner:

$$\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}) = \underset{\ddot{\mathbf{x}}}{\operatorname{argmin}} \sum_i \frac{1}{2} ||\mathbf{f}_i(\mathbf{x}, \dot{\mathbf{x}}) - \ddot{\mathbf{x}}||^2, \tag{1}$$

where $\mathbf{f}_i$ denotes the acceleration of the ith obstacle or goal.

Eq. (1) may result in undesirable behaviors. For example, it does not take the error direction into account. This can make a huge difference when an agent moves in the direction parallel to the obstacle surface compared to moving towards the obstacle surface. A principled solution is to assign a Riemannian metric that stretches the local space such that the cost of moving in one direction is different from another. The magnitude of a cost vector $\mathbf{v}$ w.r.t a Riemannian metric $\mathbf{A}$ is defined as $||\mathbf{v}||^2_{\mathbf{A}} = \mathbf{v}^\top \mathbf{A} \mathbf{v}$. Hence the policy becomes

$$\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}) = \underset{\ddot{\mathbf{x}}}{\operatorname{argmin}} \sum_i \frac{1}{2} ||\mathbf{f}_i - \ddot{\mathbf{x}}||^2_{\mathbf{A}_i}, \tag{2}$$

where $\mathbf{A}_i$ is the Riemannian metric associated with the policy $\mathbf{f}_i$. We define a Riemannian Motion Policy as a motion policy $\mathbf{f}$ associated with a Riemannian metric $\mathbf{A}$.

Eq. (2) only considers a point agent, but real robots have non-negligible shapes (e.g., a vehicle) and complex mechanics (e.g., a robotic arm). Considering these additional factors, we model the agent as a set of *control points* $\mathbf{x}_1, \mathbf{x}_2, ...$ with corresponding forward kinematic functions (also known as a task map) ϕ_i linked to a configuration space $\mathbf{q}$ such that $\mathbf{x}_i = \phi_i(\mathbf{q})$. To compute the optimal acceleration in the configuration space, we incorporate the Jacobian of the task map $\mathbf{J}_\phi = \frac{\partial \phi}{\partial \mathbf{q}}$ into Eq. (2) according to [19]:

$$\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}) = \underset{\ddot{\mathbf{q}}}{\operatorname{argmin}} \sum_i \frac{1}{2} ||\mathbf{f}_i - \mathbf{J}_{\phi_i} \ddot{\mathbf{q}}||^2_{\mathbf{A}_i}, \tag{3}$$

with a slight abuse of notation of $\mathbf{J}_{\phi_i}$ denoting the Jacobian of a control point involved in motion policy i. Eq. (3) can be solved analytically:

$$\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}) = \ddot{\mathbf{q}}^* = (\sum_i \mathbf{J}_i^\top \mathbf{A}_i \mathbf{J}_i)^+ (\sum_i \mathbf{J}_i^\top \mathbf{A}_i \mathbf{f}_i), \tag{4}$$

where $^+$ denotes pseudoinverse.

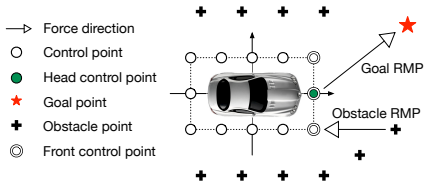
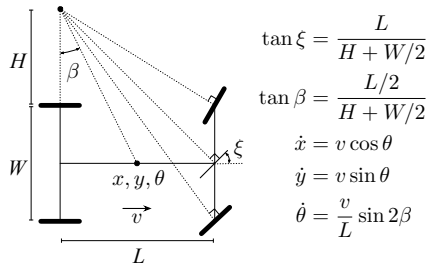


Fig. 2: We model the vehicle geometry with control points and its interaction with the environment with RMPs.



$$\tan\xi = \frac{L}{H + W/2}$$
$$\tan\beta = \frac{L/2}{H + W/2}$$
$$\dot{x} = v\cos\theta$$
$$\dot{y} = v\sin\theta$$
$$\dot{\theta} = \frac{v}{L}\sin 2\beta$$

Fig. 3: Kinematic model. v is the forward velocity and ξ is the steering angle.

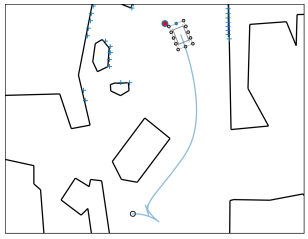


Fig. 4: A trajectory generated by our RMP controller.

To concretize, consider a turtlebot with local boundary coordinates $\mathbf{x}'_1, \mathbf{x}'_2, ...$ The configuration space $\mathbf{q}$ is its global position and orientation $[x, y, \theta]^\top$, and $\mathbf{x}_i = \phi_i(\mathbf{q})$ is the global position of each control point $\mathbf{R}(\theta)\mathbf{x}'_i + [x, y]^\top$, where $\mathbf{R}(\cdot)$ denotes a rotation matrix.

B. RMP Controller for an Ackermann Steering Vehicle

1) RMP Controller Design: Almost all automobiles and RC cars can be modeled as Ackermann steering vehicles. We model the geometry of a vehicle by defining 12 control points distributed on a rectangular bounding box around the body as in Fig. 2. There are two types of interactions between a control point and a world point:

Goal RMP. The goal point $\mathbf{g}$ pulls a control point towards it. The acceleration exerted by a goal point is defined as:

$$\alpha \frac{\mathbf{g} - \mathbf{x}}{||\mathbf{g} - \mathbf{x} + \epsilon||_2} - \beta\dot{\mathbf{x}}$$

where α, β are gain and damping factors respectively, and ϵ is a small value to prevent dividing by zero. This expression applies the maximum acceleration when the vehicle is still, and gradually decreases acceleration until the vehicle reaches its maximum speed α/β. The design of the goal RMP shows an advantage of the RMP framework: the behavior of the vehicle, such as its acceleration curve, can be precisely specified. We use the identity metric $\mathbf{I}$ for the goal RMP. Since we want the vehicle to point towards the target due to the limited field of view of sensors, we apply the goal RMP only on the three front control points. This will induce a steering force to align the vehicle's heading to the goal.

Obstacle RMP. An obstacle point pushes a control point away from it. The acceleration exerted by an obstacle $\mathbf{o}$ to a control point $\mathbf{x}$ is defined as

$$\mathbf{f}_o = -\frac{\alpha}{||\mathbf{o} - \mathbf{x}||_2}(\beta\mathbf{u} \cdot \dot{\mathbf{x}} + \gamma)\mathbf{u}$$

where α, β, γ are gain, damping and offset respectively, and $\mathbf{u} = (\mathbf{o} - \mathbf{x})/||\mathbf{o} - \mathbf{x}||_2$ which is the unit vector denoting the direction of the obstacle. Intuitively, the force increases when the obstacle is getting closer and when the velocity of the vehicle is pointing towards the obstacle. The metric for an obstacle is defined as

$$w(||\mathbf{o} - \mathbf{x}||_2)\mathbf{f}_o\mathbf{f}_o^\top,$$

where $w(\cdot)$ is a weighting function of the distance to the obstacle that is usually monotonically decreasing. $\mathbf{f}_o\mathbf{f}_o^\top$ defines a metric that penalizes directions towards the obstacle and assign zero cost to its null space (i.e., when the agent moves parallel to the obstacle surface).

Combining the goal RMP and the obstacle RMPs yields a usable but unsatisfactory policy. The vehicle tends to wiggle as it moves forward. To stabilize the vehicle, we scale up accelerations of the front left and front right control point. In addition, we add a RMP to the head control point to apply a torque that dampens excessive angular oscillation.

2) Incorporating the Kinematic Model: An Ackermann steering vehicle is non-holonomic, i.e., it cannot move sideways (assuming no drift). Using $\mathbf{q} = [x, y, \theta]$ will thus not satisfy this constraint and render the policy non-applicable in some situations. To solve this problem, we incorporate the kinematic model of the vehicle into our policy, shown in Fig. 3 [28]. We parameterize $\ddot{\mathbf{q}}$ with $[\dot{v}, \dot{\xi}]$, i.e., the forward acceleration and steering velocity. We can thus derive the following relations between $[\ddot{x}, \ddot{y}, \ddot{\theta}]$ and $[\dot{v}, \dot{\xi}]$:

$$\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{\theta}\end{bmatrix} = \underbrace{\begin{bmatrix}\cos\theta & 0\\ \sin\theta & 0\\ \frac{\sin 2\beta}{L} & \frac{4v\cos 2\beta}{3\cos^2\xi + 1}\end{bmatrix}}_{\mathbf{J}}\begin{bmatrix}\dot{v}\\ \dot{\xi}\end{bmatrix} + \begin{bmatrix}0\\ \frac{v^2}{L}\cos\theta\sin 2\beta\\ 0\end{bmatrix}.$$

Note that the relationship is not linear due to the additional term. We approximate the relationship by dropping the last term (alternatively we can subsume it into $\mathbf{f}_i$), allowing us to express Eq. (3) as

$$\dot{v}^*, \dot{\xi}^* = \underset{\dot{v},\dot{\xi}}{\operatorname{argmin}} \sum_i \frac{1}{2}||\mathbf{f}_i - \mathbf{J}_{\phi_i}\mathbf{J}[\dot{v}, \dot{\xi}]^\top||^2_{\mathbf{A}_i}. \qquad (5)$$

Fig. 4 shows a simulated trajectory of our RMP controller on a real floorplan. The vehicle starts with facing the wall and the goal is behind an obstacle. No planner is used and the motion of the vehicle is solely based on a simulated 240° laser scanner installed on the vehicle. Our RMP controller naturally considers the motion constraints of the vehicle and initially exerts a backward acceleration with a left steering signal, adjusting the heading of the vehicle towards the goal. Once the heading is good enough, it produces a forward acceleration with a right steering signal to curve around the obstacle and eventually reach the goal. This example shows that even our motion policy is based on local observations only, it is able to produce complex behaviors that are useful for real-world navigation. Interestingly, we find it resembles how human drives in this example.

IV. NEURAL RMP

The RMP controller described in Sec.III-B requires coordinates of the obstacle points, which cannot be directly measured using a monocular RGB camera. In this section, we present three neural architectures to learn RMP controllers from visual images: two baseline models that predict depth and controls respectively, and our model that predicts RMPs.

Predicting depth. In order to obtain the coordinates of the obstacle points, a straightforward way is to design a neural network that predicts a 1D laser scan (i.e., a depth map) from an input image, from which we apply the RMP controller analytically. However, predicting depth from a single image may not generalize well, as we shall see in Sec VI. Secondly, visual depth estimation can be ill-posed, such as when the vehicle is facing a wall where the image becomes completely featureless. With depth information becoming highly uncertain, the behavior of the vehicle also becomes unpredictable.

Predicting controls. An alternative scheme is to train an end-to-end network [16], [17], [18] that directly outputs the control commands. This is appealing due to its ability of learning both geometric and semantic information from visual images. However, this approach is entirely data-driven and thus lacks an interpretation of how the environment affects the behavior of the vehicle. Furthermore, without explicitly modelling the geometry and dynamics of the vehicle, it could also have an adverse effect on the generalizability of model, which we show in our experiments in Sec VI.

Predicting RMPs. To address the limitations of the abovementioned approaches, we propose a new model that predicts RMPs from visual images. RMP has merits from both schemes. The acceleration component $\mathbf{f}$ in a RMP is the command applied to a control point, whereas the metric component $\mathbf{A}$ encodes the local geometry of that control point. Moreover, by incorporating the control point Jacobians and the kinematic model, we can solve the optimal control command for the vehicle by combining the contribution of each control point in a geometrically and kinematically consistent manner, potentially having a more interpretable and generalizable model.

Fig. 5a shows the architecture of our neural RMP model. It comprises multiple feature extractors, a regressor and a solver. The image feature extractor can be any image classification network. We adopted the pretrained ResNet-50 [29] because it produces good features while not being too heavyweight to run on an embedded computer (e.g., Jetson TX2). Since a RMP may require additional information as inputs, such as velocity or goal point location, we add a feature extractor (multiple fully connected layers) for each type of odometry information. The features are then concatenated and fed into a regressor (multiple fully connected layers) to predict an array of RMPs, i.e., accelerations and metrics.

While we may define multiple RMPs for each control point, the RMP framework allows combining multiple RMPs into a single equivalent RMP [19], hence we only predict one RMP for each control point. For a land vehicle, a RMP consists of a 2-element acceleration $\mathbf{f}$ and a 2×2 Riemannian

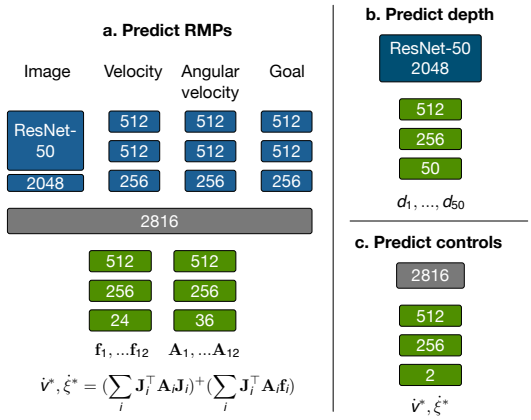


Fig. 5: Our RMP network and two baseline models. The predicting controls network reuses the feature extractor of the RMP network.

metric. Due to the symmetric nature of Riemannian metrics, we only predict three values for each metric, from which we assemble the full matrix. Finally, the solver merely computes the final control command, and is not part of the network.

We also propose two models that predict depth and controls respectively for comparison. The first model (Fig. 5b) predicts depth from image features. The second model (Fig. 5c) replaces our RMP regressor with a control command regressor. We keep the architecture and computational cost roughly the same for all models so that we can compare which representation is more effective.

V. SYSTEM IMPLEMENTATION

We designed our RMP controller for a 1/10 RC car [30] (Fig. 1). The vehicle has a dimension of approx. $40\text{cm} \times 25\text{cm}$, and is equipped with a 240° field of view laser scanner. The vehicle takes speed and steering angle as control commands while providing current speed and steering angle as odometry information. The laser scanner is used to localize the vehicle to compute waypoints [31], and also to test our expert RMP controller. Note that we do not require the waypoints to be precise or even visible (see Sec.VI-C.2), so other localization systems such as Wi-Fi based methods can also be used. We manually tuned the RMP parameters for the RC car. We found that after our expert (RMP using laser scans), works well in simulation, it requires little tuning on the real vehicle, indicating that our simulated expert transfers well to the real vehicle.

The vehicle is also equipped with a fisheye RGB camera and a Jetson TX2 computer to run our neural model. Fisheye images are rectified and cropped to produce images of 120° horizontal field of view. Since we are doing closed-loop control, minimizing latency is important. We optimized our model using the TensorRT engine to achieve 50 fps inference time from a 224×224 RGB image. Taking other factors into consideration, such as image capture, image rectification and RMP solving, our end-to-end control loop runs at 25 fps.

VI. EXPERIMENTS

We trained our neural model in the Gibson simulation environment [20]. The groundtruth trajectories are generated using the expert RMP controller (Sec.III-B) with a 240°

Agents	space8		house24		house29		house31		house57	
	reached	collision	reached	collision	reached	collision	reached	collision	reached	collision
Expert	97.9%	0.4%	59.2%	3.6%	94.5%	2.4%	97.0%	0.9%	95.0%	1.4%
Predicting RMPs	**88.1%**	**7.4%**	75.5%	**5.2%**	**93.7%**	**1.6%**	**82.1%**	**3.6%**	**89.5%**	**5.9%**
Predicting depth	85.4%	10.3%	**79.1%**	10.1%	89.0%	9.4%	73.3%	22.5%	78.1%	17.8%
Predicting controls	51.3%	19.9%	48.0%	16.3%	68.5%	14.2%	47.4%	25.5%	56.6%	21.9%

TABLE I: Statistics on the five holdout Gibson environments. Note that our neural RMP occasionally outperforms the expert (house24). This is because our neural model is less conservative in avoiding obstacles which leads to higher reach% in environments with narrow passages, but at a cost of higher collision%. The collision events for our expert controller happened mostly when the vehicle at high speed entered a narrow passage from an open space. Additional tuning could reduce such collision events.

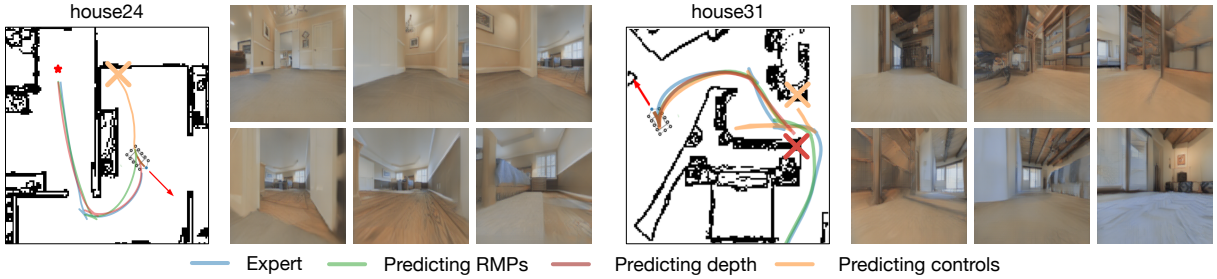


Fig. 6: Sampled trajectories in our test sets. Red arrow shows the initial location and heading. Cross shows the collision point.

laser scanner and RMP parameters tuned for the real RC car. Our training dataset consists of 60k trajectories sampled from 15 indoor spaces. During training, we randomized velocity (scale $\sim \mathcal{U}(0.0, 2.0)$, rotation $\sim \mathcal{U}(0, 2\pi)$), waypoints (rotation $\sim \mathcal{U}(0, 2\pi)$) and lighting conditions (contrast $\sim \mathcal{U}(0.5, 2.0)$ and brightness $\sim \mathcal{U}(-0.2, 0.2)$). We use the L2 loss to compute gradients. After the first epoch, we apply DAGGER [32] to augment the dataset and gradually increase the ratio of DAGGER samples for subsequent epochs.

A. Comparing Predicting Depth, Controls and RMPs

We train one agent for each approach using the same training dataset for the same number of epochs, and test them in 5 holdout Gibson environments. During testing, we randomly sample a starting point and a goal point, and use A* to compute a shortest path. At any time step, the goal point of the agent is the furthest visible point on the shortest path, hence the agent only uses the path as a coarse guidance, and does not have to follow it strictly. In fact, the shortest path is greedy (e.g., very close to obstacles) so that naively following it would cause collision.

We use two metrics to evaluate the navigation performance: the percentage of trajectories where an agent reaches the goal (reached%), and the percentage of trajectories where collision occurs (collision%). We stop an agent once collision happens, hence we have reached% + collision% + stuck% = 100%. We collected over 200 trajectories for each holdout environment and present the results in Table I. The performance of our RMP agent is the closest to our expert, with comparable high reached% and low collision%. Both *Predicting depth* and *Predicting controls* have much higher collision rate with lower reached%, with predicting controls being more likely to get stuck. This shows our RMP agent generalizes much better than predicting controls due to its explicit modelling of the vehicle geometry and dynamics. Compared to predicting depth, the RMP representation is more concise and less noisy, and thus it is more robust when a robot operates in tight spaces, where small measurement errors in the geometry would cause failures.

Fig. 6 shows two sample test trajectories. These two trajectories are challenging because they require sharp turns that cannot be completed without backing the vehicle due to its steering limit. Also the tight spaces have low tolerance for depth measurement error. *Predicting controls* backs too much in both environments and ends up hitting the walls. The *Predicting depth* agent failed the last turn in house31. In comparison, *Predicting RMPs* succeeded in both cases.

B. Visualization of the Learned RMPs

Fig. 8 visualizes the predicted RMPs when the vehicle is a) running in a straight corridor and b) doing a right turn. The learned Riemannian metrics assign high costs to the directions towards obstacles as shown by the elliptical isocontours of the metrics, and lower costs to control points far away from obstacles as shown by their larger contour sizes. Same reasoning also holds for the predicted accelerations. As a result, we are able to intuitively reason about the behavior of the vehicle by examining the predicted RMPs. This also allows us to add additional RMPs to adjust the behavior of the vehicle without retraining the network.

C. Real World Experiments

1) Hallway navigation: We also evaluated our models trained in simulation in a real hallway to test their generalizability. We perform a similar evaluation as in the Gibson environments by setting a starting point and a goal point and let each agent follow a sequence of waypoints. Fig. 7 shows the recorded trajectories for the three neural agents.

We find that there exists a domain gap between the rendered images in the Gibson environments (Fig. 6) and the real images taken from the hallway (Fig. 7). All three agents failed to pass the first left turn because they ran straightly towards the first waypoint which sits at the corner of the wall. While Gibson provides a *Goggles* mechanism [20] to

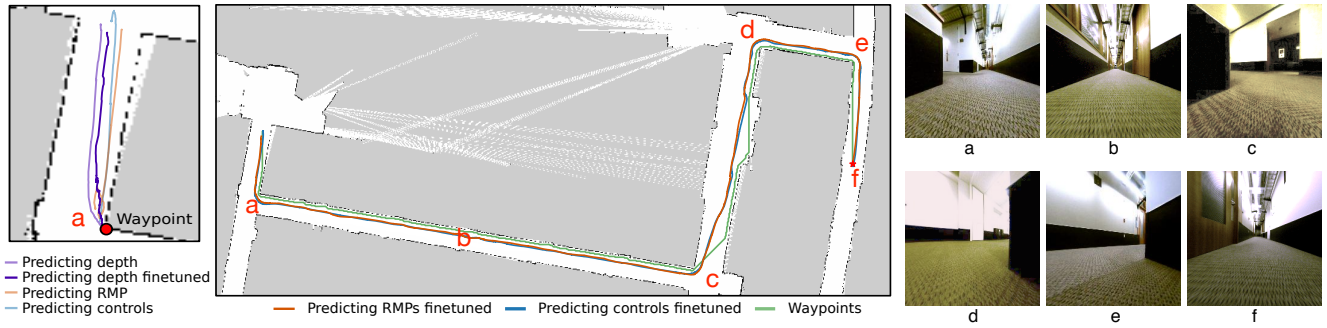


Fig. 7: Recorded trajectories in a real hallway. Left: before finetuning all agents hit the walls. Middle: finetuned *Predicting RMPs* and *Predicting controls* agents reached the goal. Right: sample images on the trajectory.

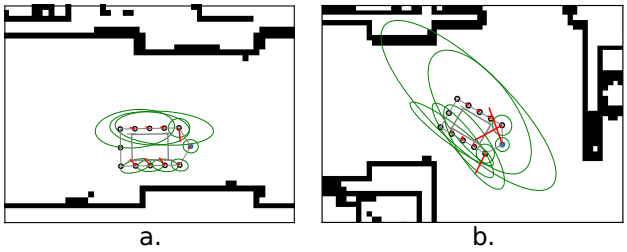


Fig. 8: Illustration of predicted RMPs. Red lines are the predicted control point accelerations. Green circles are isocontours of the predicted control point metrics.

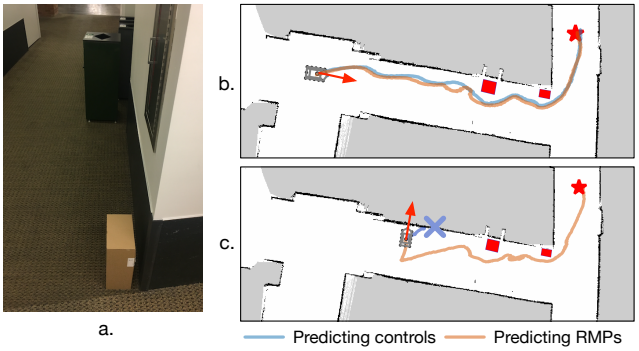


Fig. 9: Obstacle avoidance without a planner. Left: placed obstacles. Right: results for two different placements of the vehicle. Red arrow indicates the vehicle heading. Cross shows the collision point.

reduce the domain gap, we find its high computational cost introduces too much delay to our control loop. To mitigate this issue, we manually drove the vehicle in the hallway for a few minutes to collect about 13k images with associated laser scans, and annotated the data using our expert RMP controller. We finetuned the three models for a few epochs, which took about 20 minutes each. After finetuning, our *Predicting RMPs finetuned* agent and *Predicting controls finetuned* agent reached the goal successfully. Surprisingly, finetuning does not help with the *Predicting depth* agent, because it is difficult to synchronize laser scans and visual images without special hardware mechanism. This results in misalignment between the predicted depth and the ground truth depth, producing spurious high loss that makes it difficult for the network to adapt to real images.

2) Obstacle avoidance: Our hallway experiment does not show a noticeable difference between predicting RMPs and predicting controls. This is probably due to the hallway environment being simple and obstacle-free. To test them in a more challenging scenario, we disabled the planner and put the goal point in a different hallway that is invisible from the vehicle. Furthermore, we placed two obstacles that were not seen during training as shown in Fig. 9a.

When the initial heading of the vehicle was pointing in the free space direction, both *Predicting RMPs* and *Predicting controls* were able to reach the goal while avoiding obstacles (Fig. 9b). However, if we set the initial heading to point towards the wall (Fig. 9c), the *Predicting controls* agent ran straightly towards the goal and failed. In contrast, our RMP agent backed the vehicle first so that it would have sufficient headroom to do a right turn, and successfully reached the goal. The RMP agent learned that there was high cost when moving forward from the expert RMP, and hence the generated RMPs performed a conservative maneuver.

VII. CONCLUSIONS

We present a novel neural autonomous navigation framework that generates smooth obstacle avoidance behaviors while being more generalizable than directly predicting depth and control commands. The key of our approach is to utilize the RMPs to unify the geometry and dynamics of the vehicle and its interaction with the environment. Since our method models the geometry and dynamics explicitly, we believe it has strong potentials in image-based robot manipulation, policy transfer and agile robot maneuver. Future works include applying it into diverse robotic tasks, developing better neural architectures and unsupervised RMP learning.

While our RMP controller exhibits good properties, it is fundamentally a local policy and hence may get stuck in some situations (e.g., when the waypoint is behind a large concave obstacle). Thus it is more suitable to use a neural planner [33] to provide high-level waypoints, and relies on the neural RMP for handling vehicle dynamics and local reactive obstacle avoidance.

VIII. ACKNOWLEDGEMENTS

This research was funded by the Honda Curious Minded Sponsored Research Agreement. We thank Patrick Lancaster for his help working with the RC car.

REFERENCES

[1] R. C. Arkin and R. R. Murphy, "Autonomous navigation in a manufacturing environment," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 4, pp. 445–454, 1990.

[2] F. Nashashibi and M. Devy, "3-d incremental modeling and robot localization in a structured environment using a laser range finder," in *IEEE International Conference on Robotics and Automation (ICRA)*, 1993, pp. 20–27.

[3] A. Soloviev, D. Bates, and F. Van Graas, "Tight coupling of laser scanner and inertial measurements for a fully autonomous relative navigation solution," *Navigation*, vol. 54, no. 3, pp. 189–205, 2007.

[4] D. Murray and J. J. Little, "Using real-time stereo vision for mobile robot navigation," *Autonomous Robots*, vol. 8, no. 2, pp. 161–171, 2000.

[5] E. Royer, M. Lhuillier, M. Dhome, and J.-M. Lavest, "Monocular vision for mobile robot localization and autonomous navigation," *International Journal of Computer Vision*, vol. 74, no. 3, pp. 237–260, 2007.

[6] D. A. de Lima and A. C. Victorino, "A hybrid controller for vision-based navigation of autonomous vehicles in urban environments." *IEEE Trans. Intelligent Transportation Systems*, vol. 17, no. 8, pp. 2310–2323, 2016.

[7] S. Thrun, "Simultaneous localization and mapping," in *Robotics and cognitive approaches to spatial mapping*, 2007, pp. 13–41.

[8] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments," in *In the 12th International Symposium on Experimental Robotics (ISER)*, 2010.

[9] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[10] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "Elasticfusion: Real-time dense slam and light source estimation," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1697–1716, 2016.

[11] R. Wang, M. Schwörer, and D. Cremers, "Stereo dso: Large-scale direct sparse visual odometry with stereo cameras," in *International Conference on Computer Vision (ICCV)*, vol. 42, 2017.

[12] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Neural Information Processing Systems (NIPS)*, 2014, pp. 2366–2374.

[13] F. Liu, C. Shen, G. Lin, and I. D. Reid, "Learning depth from single monocular images using deep convolutional neural fields." *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 38, no. 10, pp. 2024–2039, 2016.

[14] C. Godard, O. M. Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6602–6611.

[15] S. Vijayanarasimhan, S. Ricco, C. Schmid, R. Sukthankar, and K. Fragkiadaki, "Sfm-net: Learning of structure and motion from video," *arXiv preprint arXiv:1704.07804*, 2017.

[16] F. Codevilla, M. Miiller, A. Lpez, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[17] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile off-road autonomous driving using end-to-end deep imitation learning," *Robotics: Science and Systems (RSS)*, 2018.

[18] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3530–3538, 2017.

[19] N. D. Ratliff, J. Issac, and D. Kappler, "Riemannian motion policies," *CoRR*, vol. abs/1801.02854, 2018.

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

[21] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Neural Information Processing Systems (NIPS)*, 1989, pp. 305–313.

[22] M. Mueller, A. Dosovitskiy, B. Ghanem, and V. Koltun, "Driving policy transfer via modularity and abstraction," in *CoRL*, 2018.

[23] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[24] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1714–1721.

[25] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," *Robotics: Science and Systems (RSS)*, 2017.

[26] N. Ratliff, M. Toussaint, and S. Schaal, "Understanding the geometry of workspace obstacles in motion optimization," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 4202–4209.

[27] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *Robotics: Science and Systems (RSS)*, 2013.

[28] J. M. Snider, "Automatic steering methods for autonomous automobile path tracking," *CMU-RI-TR-09-08*, 2009.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[30] "MIT racecar," 2018. [Online]. Available: https://mit-racecar.github.io/

[31] D. Fox, W. Burgard, and S. Thrun, "Markov localization for mobile robots in dynamic environments," *Journal of artificial intelligence research*, vol. 11, pp. 391–427, 1999.

[32] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011, pp. 627–635.

[33] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 7272–7281.