2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Stochastic Neural Control using Raw Pointcloud Data and Building Information Models

Max Ferguson[1] and Kincho H. Law[2]

***Abstract*— Recently, there has been a lot of excitement surrounding the use of reinforcement learning for robot control and navigation. However, many of these algorithms encounter difficulty navigating long or complex trajectories. This paper presents a new mobile robot control system called Stochastic Neural Control (SNC), that uses a stochastic policy gradient algorithm for local control and a modified probabilistic roadmap planner for global motion planning. In SNC, each mobile robot control decision is conditioned on observations from the robot sensors as well as pointcloud data, allowing the robot to safely operate within geometrically complex environments. SNC is tested on a number of challenging navigation tasks and learns advanced policies for navigation, collision-avoidance and fall-prevention. Three variants of the SNC system are evaluated against a conventional motion planning baseline. SNC outperforms the baseline and four other similar RL navigation systems in many of the trials. Finally, we present a strategy for transferring SNC from a simulated environment to a real robot. We empirically show that the SNC system exhibits good policies for mobile robot navigation when controlling a real mobile robot.**

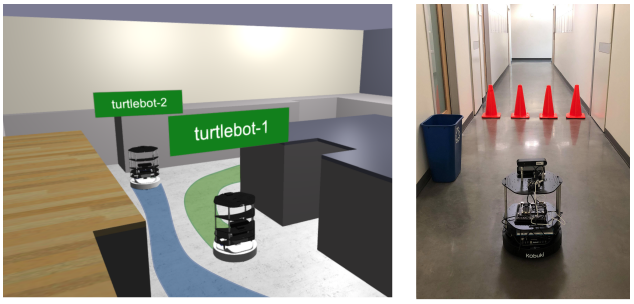
I. Introduction

The use of mobile robots is becoming increasingly common in construction, facility management and disaster recovery. However, many mobile robot systems still lack the high-level perception and decision-making skills that come naturally to human operators [2]. For example, most contemporary motion planning algorithms still have trouble choosing a safe trajectory when faced with sufficient levels of uncertainty, often leading to the frozen robot problem [26]. To overcome these challenges, an ideal mobile robot motion planning system must be able to make good control decisions under high levels of uncertainty, potentially by exploiting a rich stochastic model of the surrounding environment. More specifically, we argue that an ideal decision-making system for a mobile robot should be able to:

- Leverage multiple sources of information, possibly including lidar data, camera images, robot odometry and sensor data.
- Learn or exploit prior distributions over the movement of other objects in the building space to avoid collision scenarios.
- Exploit human-verified environment models to make better local decisions in the vicinity of known hazards like stairs or cliffs.

[1]Max Ferguson is with Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305, USA maxferg@stanford.edu

[2]Kincho H. Law is with the Faculty of the Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305, USA law@stanford.edu



(a) Simulated Navigation Trials (b) Physical Robot Trials

Fig. 1. Mobile robot navigation using stochastic neural control in (a) simulated and (b) real test environments.

Recently, some researchers have demonstrated that reinforcement learning (RL) techniques can be used to make good robot control decisions, even in uncertain environments [18], [28]. One of the main benefits of RL-based techniques is that they can condition control actions on high-dimensional input data, such as camera images or pointcloud data [18]. Another benefit of RL-based techniques is that they can implicitly learn prior distributions over the behavior of other moving objects. Despite these benefits, there are still very few practical examples where RL has been used for control or local motion planning using high-dimensional input from a real mobile robot. Robot control algorithms that use raw image or pointcloud data generally require a large amount of training data and often fail to transfer well from simulated environments to real robots [28]. Similarly, end-to-end control systems that solve the entire navigation problem using RL generally fail due to the sparse rewards problem [5].

We present a new mobile robot control system called Stochastic Neural Control (SNC), that uses a model-free stochastic policy gradient RL algorithm for local control and modified Probabilistic Roadmap (PRM) planner for global motion planning. The local motion planning system conditions each control action on the mobile robot state and raw pointcloud data from an RGB-D camera or lidar sensor. During the training process, pointcloud data is generated using a simulated building model. When SNC is deployed on a real mobile robot, we combine pointcloud data from a variety of sources including the mobile robot lidar sensor, RGB-D camera and a 3D building model. Using synthetic pointcloud data at the both the training and evaluation phases makes it easier to transfer the trained policy from a simulated environment to a real robot.

This paper introduces a mobile robot control system

that conditions control decisions on multiple sources of information, such as raw pointcloud data from an RGB-D camera. Several RL techniques are considered for mapping high-dimensional sensor data to control actions. The main contributions of the paper are:

- A mobile robot navigation system that directly conditions control decisions on mobile robot state and raw pointcloud data.
- A set of experiments illustrating how the SNC method compares to an established motion planning system, and four other prominent RL control algorithms.
- A strategy for transferring a trained SNC agent to a real mobile robot.

The remainder of this paper is organized as follows: Section 2 describes related works in robotics and computer vision. Section 3 introduces the SNC system. Section 4 introduces the global motion planning system that is used with SNC. Sections 5 and 6 describe the performance of SNC under simulated and real conditions, respectively. The paper is concluded with a discussion and a conclusion.

II. Related Work

Recent work has been proposed in visual navigation focusing on navigating both indoor [28] and outdoor [22] environments using deep RL. Researchers have presented methods that leverage 2D maps [18], pointclouds [7], and visual input alone [12]. Recently, RL has been used to successfully solve challenging problems in local robot control [8], [25]. Soft Actor-Critic [8] is a current state-of-the-art algorithm that works with high-dimensional state and action spaces and is able to learn to control robots based on unprocessed sensor observations. Similarly, RL techniques have also been used to evaluate the quality of discrete navigation options, such as the decision for an autonomous car to enter a traffic intersection [11]. SNC shares some similarities with hierarchical RL methods, which tend to perform well on maze environments [3], [7].

Several methods have emerged that use RL for local motion planning whilst relying and rely on a more established method for global motion planning [3]. The PRM-RL system uses deep deterministic policy gradient (DDPG) [27] for local control and PRM for global motion planning [3]. While PRM-RL is one of the first RL-based systems to solve the long navigation problem on a real robot, the RL system only conditions control actions on a simplified low-dimensional lidar observation. Several other local motion planning systems condition control actions on richer observations like camera images [12] or RGB-D images. Transferring information from global motion planning systems to RL control algorithms can often be challenging [20]. Systems that combine RL with global motion planning often use waypoints [10] or frequently updated target locations [20] to communicate information from the motion planner to the RL algorithm.

III. Stochastic Neural Control

In this section, we explicitly define the navigation task and introduce the SNC system. SNC is divided into two loosely coupled components: A local motion planning system that outputs a distribution over optimized control actions and a global motion planning system that outputs waypoints that connect the start position to the target position.

A. Navigation Problem

The navigation task is framed as an infinite horizon partially observed Markov decision process (POMDP) with continuous state space $\mathcal{S}$, and continuous action space $\mathcal{A}$. The mobile robot is initially located at point $\boldsymbol{x}_0 \in \mathbb{R}^3$ and must navigate to a target point $\boldsymbol{x}_{target} \in \mathbb{R}^3$, by executing a series of control actions $\boldsymbol{a} \in \mathcal{A}$. Each control action $\boldsymbol{a} = (v_r, v_l)$ has a component v_r which controls the turning speed of the robot and a component v_l which controls the linear velocity of the robot. After each action is made, the environment transitions to state $\boldsymbol{s}^{'}$ according to the state transition probability $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, \infty)$. At the same time, the agent receives an observation $\boldsymbol{o}$ and a reward $r : \mathcal{S} \times \mathcal{A} \rightarrow [r_{min}, r_{max}]$. The process then repeats. The main objective is to find a policy $\pi : S \rightarrow A$ that maximizes the expected future discounted reward: $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where r_t is the reward earned at time t and γ is the discount factor.

B. Reinforcement Learning Approach

The local motion planning system learns a policy that maps a high-dimensional observation to a distribution over the action space. By considering a distribution over action space, the control system can distinguish between situations where multiple different control actions could lead to a satisfactory result and situations where a single control action is optimal.

The local motion planning system is based on the Soft Actor-Critic (SAC) algorithm. SAC is an off-policy RL algorithm that combines several recent approaches in RL including double Q-learning [27] and entropy-regularized rewards [8]. SAC generalizes the standard RL objective by augmenting it with an entropy term $\mathcal{H}(\pi(\cdot|\boldsymbol{s}_t))$, such that the optimal policy additionally aims to maximize its entropy at each visited state:

$$\pi^* = \arg\max_{\pi} \sum_t \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim \rho_\pi}[r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \alpha \mathcal{H}(\pi(\cdot|\boldsymbol{s}_t))], \tag{1}$$

where ρ_π is the state-visitation distribution and α is the temperature parameter that determines the relative importance of the entropy term versus the reward [8]. Entropy-regularization can be particularly beneficial in motion planning systems as it encourages the RL agent to explore a range of motion planning and collision avoidance strategies during the training process [8]. In SNC, we need to compute the value of a policy π according to the maximum entropy objective. The state-value function $V(\mathbf{s}_t)$ captures the expected value of the current state $\mathbf{s}_t$, according to policy π.

$$V(\boldsymbol{s}_t) = \mathbb{E}_{\boldsymbol{a}_t \sim \pi}[Q(\boldsymbol{s}_t, \boldsymbol{a}_t) - \alpha \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t)]. \tag{2}$$

where $Q(\boldsymbol{s}_t, \boldsymbol{a}_t)$ is the Q-value for action $\boldsymbol{a}_t$. For a fixed policy, the Q-value can be computed iteratively, starting from

any function $Q : S \times A \rightarrow R$ and repeatedly applying a modified Bellman backup operator $\mathcal{T}^{\pi}$ given by:

$$\mathcal{T}^{\pi}Q(\boldsymbol{s}_t, \boldsymbol{a}_t) \triangleq r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1} \sim p}[V(\boldsymbol{s}_{t+1})], \quad (3)$$

Learning in large continuous domains requires us to derive a practical approximation for Q-function and the policy function. We consider a parameterized Q-function $Q_\theta(\mathbf{s}_t, \mathbf{a}_t)$ with parameters θ, and a tractable policy $\pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$ with parameters ϕ. The Q-function parameters can be trained by minimizing the soft Bellman residual:

$$J_Q(\theta) = \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim \mathcal{D}} \Big[\frac{1}{2} (Q_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t) - (r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1} \sim p}[V_{\bar{\theta}}(\boldsymbol{s}_{t+1})]))^2 \Big], \quad (4)$$

where $\mathcal{D}$ is the distribution of states in the replay buffer and the value function is implicitly parameterized through (2). The update makes use of parameters $\bar{\theta}$ that are obtained as an exponential moving average of θ, which has been shown to stabilize training [19]. However, our experimental trials show that the magnitude of $Q_{\bar{\theta}}$ can still explode when using large neural networks. To stabilize training process, we clip the output of $Q_{\bar{\theta}}$ to $[3R_{min}, 3R_{max}]$, where R_{min} is theoretical minimum sum of undiscounted rewards in a single trajectory and R_{max} is the theoretical maximum sum of undiscounted rewards in a single trajectory.

The policy parameters can be learned by minimizing the following objective:

$$J_\pi(\phi) = \mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{D}} \big[\mathbb{E}_{\boldsymbol{a}_t \sim \pi_\phi} [\alpha \log(\pi_\phi \boldsymbol{a}_t | \boldsymbol{s}_t)) - Q_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t)] \big], \quad (5)$$

The policy is trained by alternating between collecting data from the environment and minimizing the two objective functions shown in (4) and (5), using stochastic gradient descent.

C. Reward Function

The reward function is designed to incentivize the robot to move safely and efficiently from the current location to the target. The robot is rewarded for reaching the target location:

$$r_{target}(\boldsymbol{s}_t) = \begin{cases} 1 & \text{if } \|\boldsymbol{x}_t - \boldsymbol{x}_{target}\| < \epsilon_{target}, \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

where $\boldsymbol{x}_t \in \mathbb{R}^3$ is the current position of the robot, $\boldsymbol{x}_{target} \in \mathbb{R}^3$ is the target position, and ϵ_{target} is a distance threshold. The robot is also rewarded for reaching the waypoints:

$$r_{waypoint}(\boldsymbol{s}_t) = \begin{cases} 0.1 & \text{if } ||\boldsymbol{x}_t - \boldsymbol{x}_{waypoint}|| < \epsilon_{waypoint}, \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

where $\boldsymbol{x}_{waypoint} \in \mathbb{R}^3$ is the waypoint position and $\epsilon_{waypoint}$ is a distance threshold. Each waypoint is removed from the building after it is visited by the robot. The robot is penalized for collisions:

$$r_{collision}(\boldsymbol{s}_t) = \begin{cases} -1 & \text{if } ||\boldsymbol{x}_t - \boldsymbol{x}_{object}|| < \epsilon_{collision}, \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

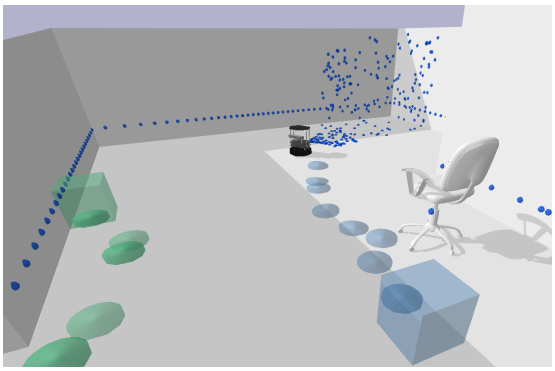


Fig. 2. Rendering of a mobile robot observation. The small blue spheres represent a single pointcloud observation, the transparent cylinders represent waypoints, and the transparent cubes represent target locations.

where $\boldsymbol{x}_{object}$ is the closest rigid-body object or wall in the environment and $\epsilon_{collision}$ is the collision tolerance. For safety and efficiency purposes, we prefer if the robot travels slower than its maximum velocity capability. For example, in this study, any linear control action v_l, with magnitude greater than 0.3 meters/second is penalized. Additionally, we choose to penalize turning speed v_r, to prevent the robot rotating aimlessly. The corresponding control penalty is:

$$r_{control}(\boldsymbol{a}_t) = -0.01 \max(|v_l| - 0.3, 0) - 0.01|v_r|. \quad (9)$$

The total reward is defined as the sum of the reward components. The episode is terminated if the robot reaches the target, or if the robot collides with another object.

D. Observations

The SNC system conditions each control action on a vector describing the robot state and a spatial observation from mobile robot sensors. The robot state vector contains the orientation and velocity of the robot, the location of the nearest k waypoints, and the target location. The global position of the robot is excluded from the observation to encourage the robot to learn a policy that generalizes to many spatial configurations. The spatial observation describes the building geometry and spatial position of objects near the mobile robot. Each spatial observation is encoded using a neural network and concatenated with the robot state vector to fully represent the robot state. Several different spatial observation methods are tested:

1) Horizontal lidar distance: The free space surrounding the robot can be represented in a vector form, where each entry denotes the length of a lidar ray projected in a pre-defined direction. Specifically, a 12-element vector is used to describe the distance between the mobile robot and other objects. During training, the lidar distances are simulated by projecting rays from the mobile robot lidar sensor along the horizontal plane, at equally spaced angles. On a real robot, these values are obtained by aggregating data from the physical lidar scanner.

2) Occupancy grid: The space surrounding the mobile robot can also be described using an occupancy grid, as is common in many mobile robot systems. In our experiments,

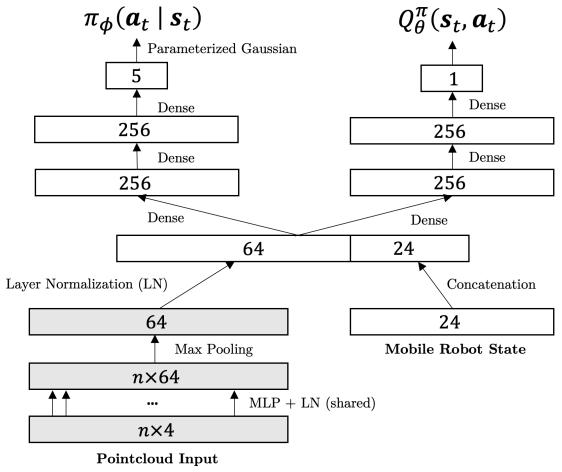


Fig. 3. Neural network architecture for Policy and Q-function when using the pointcloud encoder. The shaded boxes denote the pointcloud encoder.

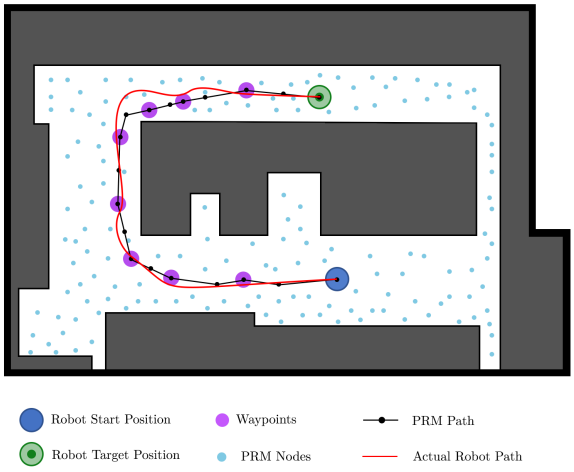


Fig. 4. A global motion planning algorithm is used to generate waypoints for the RL control system. TheRL control system guides the robot toward the target location, visiting waypoints wherever the incentive reward outweighs the cost of a potential collision.

a 64 $\times$ 64 pixel image is used to describe the occupancy of the 4 meter $\times$ 4 meter area surrounding the mobile robot. The mobile robot is always located at the center of the occupancy grid observation and the occupancy grid observation is rotated so that the top edge always represents the space directly in front of the mobile robot. The occupancy grid has three binary channels. Pixels in the first channel are set to 1 in regions that are occupied by building geometry and 0 otherwise. Pixels in the second channel are set to 1 in regions that are occupied by stationary objects and 0 otherwise. Pixels in the third channel are set to 1 in regions that are occupied by other robots and 0 otherwise.

3) Synthetic pointcloud data: Finally, the space surrounding the robot can be represented using a pointcloud from a simulated lidar sensor or RGB-D camera. Each point in the pointcloud is represented as a (x, y, z, c) tuple that specifies the location of the point relative to the mobile robot, and the class c of the object that the point belongs to. No additional structure is imposed on the position of points in the pointcloud, allowing points to be obtained from various sensors or a 3D geometry model. During training, the pointcloud is created by simulating the combined measurements from a horizontal lidar scanner and an RGB-D camera. We assume that the horizontal lidar has an angular resolution of 1° and the RGB-D camera has a resolution of 1280 $\times$ 720 pixels. To reduce the size of the pointcloud, 120 points are randomly selected from the lidar pointcloud and 240 points are randomly selected from the RGB-D pointcloud at each timestep. A rendering of the synthetic pointcloud observation is shown in Fig. 2.

E. Neural Network Encoders

Neural networks are used to model the Q-function, policy function and global cost function. Typical actor-critic implementations use a two-layer dense neural network for both the Q-function and the policy [8]. However, more complex neural networks are required to encode the spatial observations for SNC. An encoder is used to reduce each spatial observation to a single 64-element vector. The encoder for the horizontal lidar observation is a 2-layer perception with ReLU activation. The encoder for the occupancy grid is a 3-layer convolutional neural network (CNN) with alternating convolution and max-pooling layers, similar to the encoder described in [19]. The encoder for the pointcloud data is a smaller variant of the PointNet neural network [23]. In the pointcloud encoder, each point in the pointcloud is mapped into $\mathbb{R}^{64}$ using a multilayer perceptron (MLP). The points are then concatenated to form a feature tensor with size $n \times 64$. Max pooling is used to aggregate the first dimension of the feature tensor, producing a feature vector with 64 elements.

After each spatial observation has been encoded by its designated encoding function it is normalized using layer normalization [1] and then concatenated with the robot state vector, as shown in Fig. 3.

IV. Intelligent Trajectory Planning

The global motion planning system finds an admissible trajectory from the start position to the target position and communicates that trajectory to the local control system. Strictly speaking, the global motion planning system modifies the underlying POMDP, making it easier to solve with RL, whilst ensuring that the optimal RL policy π^* still corresponds to desirable robot behavior. PRM is used to find a feasible trajectory from $\boldsymbol{x}_0$ to $\boldsymbol{x}_{target}$. PRM randomly samples a fixed number of points from the configuration space and computes the Euclidian distance between each point and its nearest n neighbors, creating a roadmap [13]. The A* search algorithm is then used to find the shortest connected path between x_0 and x_{target}. The PRM algorithm is particularly efficient when training SNC, as the roadmap only needs to be constructed once for each space of interest, such as a building. The trajectory is communicated to the local motion planning algorithm using waypoints, as shown in Fig. 4.

The cost of travelling between roadmap nodes is often based on the Euclidean distance between the nodes, but this assumption can be inaccurate in stochastic or geometrically complex environments. To overcome this limitation, SNC learns a parameterized function that estimates the cost of travelling between two roadmap nodes. Let $\boldsymbol{s}_a$ denote an arbitrary state where the robot is located at node a in the PRM roadmap, and $\boldsymbol{s}_b$ denote an arbitrary state where the robot is located at node b. We define $D^\pi(\boldsymbol{s}_a\ ,\ \boldsymbol{s}_b)$ as the undiscounted cost of moving between node a and node b under policy π:

$$D^\pi(\boldsymbol{s}_a, \boldsymbol{s}_b) = V^\pi(\boldsymbol{s}_a) - V^\pi(\boldsymbol{s}_b) \tag{10}$$

Evaluating $D^\pi(\boldsymbol{s}_a, \boldsymbol{s}_b)$ during the trajectory planning process is intractable, as the future values of $\boldsymbol{s}_a$ and $\boldsymbol{s}_b$ are difficult to predict. Instead, we learn a cost function that approximates D^π using only the spacial position of node a and node b. We define a new parameterized cost function D^π_ψ as:

$$D^\pi_\psi(\boldsymbol{x}_a, \boldsymbol{x}_b) = \mathbb{E}_{\boldsymbol{s}_a \sim \rho_a}[V^\pi(\boldsymbol{s}_a)] - \mathbb{E}_{\boldsymbol{s}_b \sim \rho_b}[V^\pi(\boldsymbol{s}_b)] \tag{11}$$

where ρ_a and ρ_b are marginal distributions over $\boldsymbol{s}_a$ and $\boldsymbol{s}_b$, induced by observations at positions $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$. The cost function D^π_ψ is modelled using a three-layer MLP with parameters ψ. The cost function parameters can be trained by minimizing the following objective:

$$J_D(\psi) = \mathbb{E}\Big[\big(D^\pi_\psi(\boldsymbol{x}_i, \boldsymbol{x}_{i+1}) - V^\pi_\theta\big(\boldsymbol{s}_i\big) + V^\pi_\theta(\boldsymbol{s}_{i+1})\big)^2\Big] \tag{12}$$

where $(\boldsymbol{x}_i,\ \boldsymbol{s}_i)$ and $(\boldsymbol{x}_{i+1},\ \boldsymbol{s}_{i+1})$ are observations collected at two consecutive waypoints during a training trajectory. Relying on D^π_ψ during the training process can lead to learning instabilities. During training, half of the simulated robots generate trajectories using an Euclidean distance cost function and the other half generate trajectories using the D^π_ψ cost function. When testing D^π_ψ is used as the cost function for every robot.

V. Experiments

The SNC system is trained and evaluated using eight simulated navigation environments [15], [16]. The smallest four environments contain two robots whilst the larger environments contain 4-8 mobile robots. Each robot must move from a randomly chosen start location to a randomly chosen target location. Many of the navigation environments are challenging, requiring the navigation algorithm to avoid collisions, falls, and deadlock scenarios.

Three variants of the SNC system are trained and evaluated against a conventional motion planning baseline. The first SNC variant conditions control actions on the robot state and a horizontal lidar observation. The second SNC variant conditions on the robot state and an occupancy grid observation. Finally, the third SNC variant conditions on the robot state and a pointcloud observation from the RGB-D camera and the horizontal lidar scanner.

The SNC models are trained using the RLlib distributed RL framework [14]. The first SNC variant is trained for 1 million timesteps using a learning rate of 2×10^{-4}. The other two variants are trained for 8 million timesteps using a learning rate of 5×10^{-5}. The training process is conducted on a single Google Cloud compute node with 96 CPU cores, 384 GB of RAM, and four NVIDIA T4 GPUs. The RLlib framework is used to spawn 64 worker processes which each simulate a single building environment. The worker evaluate the current policy and store each $(\boldsymbol{s},\ \boldsymbol{a},\ r,\ \boldsymbol{s}')$ transition in a shared replay buffer. A master node samples transitions from the replay buffer and minimizes the three objective functions using the Adam optimizer. The worker nodes collectively generate about 20,000 transitions/minute and the master node samples approximately 400,000 transitions/minute from the replay buffer. The training process takes 24-48 hours.

A. Dynamic Window Baseline

The SNC models are tested against a strong baseline algorithm that uses well-accepted techniques for motion planning and collision avoidance. Specifically, the baseline uses the Dynamic Window Approach (DWA) for local motion planning [4] and PRM for global motion planning. The standard ROS implementation of DWA is used [4]. At each planning step the robot movement is forward-simulated by 1.6 seconds. The movement of other objects, including other mobile robots, is not forward-simulated. Fall-prone regions are manually added to the cost-map to prevent the baseline algorithm from falling on stairwells or similar hazards. The same parameters are used for the PRM planner in both the baseline and the SNC trials.

B. Main Results

The experimental results are presented in Table I and Table II. A video of the learned policies is available as part of the supplementary material [17]. The performance of each agent is measured using two metrics, success rate and average completion time. The success rate is defined as the percentage of trials where the robot successfully reaches the goal. Average completion time is defined as the average number of timesteps required to travel from the start to the goal, for trajectories where the goal was reached. Overall, the SNC system performs better when using the pointcloud observation rather than the simplified lidar observation or the occupancy grid. This might be because the pointcloud observation captures more information about the surrounding environment, allowing the robot to better avoid small stationary objects of fall hazards. The occupancy grid and horizontal lidar approaches perform badly on the platform and stairwell environments as they have no direct method of avoiding fall hazards. Many of the SNC models outperform the DWA baseline in completion speed as SNC can implicitly learn a model for forward-propagating the movement of other mobile robots. The SNC system can also implicitly learn a model for avoiding hazardous spatial geometry, such as corners or narrow pathways, and fall hazards.

C. Q-function clipping

In Soft Actor-Critic, the Q-value of the last step in a trajectory is estimated using $Q_{\bar{\theta}}$. This increases learning

TABLE I

COMPLETION RATE (%) FOR EACH NAVIGATION ALGORITHM IN EIGHT SIMULATED ENVIRONMENTS

Environment	DWA Baseline	SNC Lidar	SNC Occupancy grid	SNC Pointcloud
Room	82.8	85.4	86.4	**91.2**
House	**78.9**	74.2	75.1	76.1
Laboratory	57.3	63.1	63.1	**86.3**
Facility	**63.9**	19.5	61.5	51.4
Stairwell	60.3	31.3	35.3	**67.3**
Cafe	53.5	55.3	**74.5**	62.9
Platform	72.9	13.9	14.1	**82.9**
Bottleneck	14.1	52.1	57.1	**72.1**
Average	60.5	49.4	59.6	**73.0**

TABLE II

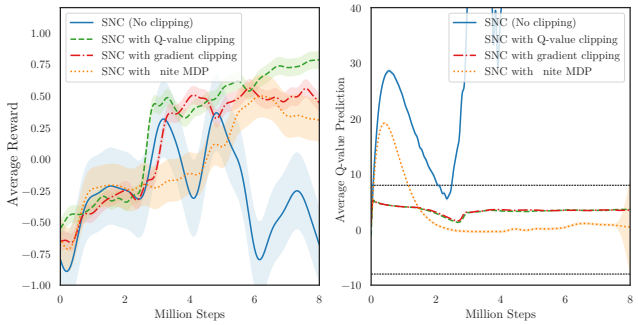
AVERAGE NUMBER OF TIMESTEPS REQUIRED TO SUCCESSFULLY COMPLETE THE SIMULATED NAVIGATION ENVIRONMENTS.

Environment	DWA Baseline	SNC Lidar	SNC Occupancy grid	SNC Pointcloud
Room	35.2	30.4	**22.1**	23.1
House	67.2	65.3	50.1	**44.1**
Laboratory	51.2	41.7	35.1	**34.3**
Facility	107.3	97.3	80.1	**76.3**
Stairwell	15.2	-	-	**8.5**
Cafe	34.2	21.0	**20.0**	22.6
Platform	40.2	-	-	**26.5**
Bottleneck	90.2	50.5	52.1	**45.2**
Average	55.1	49.0	38.0	**36.3**

stability as it prevents a trajectory arbitrarily ending with a Q-value of 0. However, bootstrapping the Q-value in this way can lead to a numerical instability where the mean Q-value grows exponentially. Several approaches are tested to overcome this instability. In the first approach, we clip the output of the target Q-network to $[3R_{min}, 3R_{max}]$. In the second approach, we solve the MDP problem with a finite horizon, ending the episode and setting the Q-value to zero after 100 steps. Finally, in the third approach, we scale the Q-value gradient so that the gradient norm does not exceed 10. Fig. 5 shows that the clipping the Q-value stabilizes the learning process and achieves the highest final reward.

D. Comparison to other learning algorithms

A number of other RL algorithms are compared to SNC, including Proximal Policy Optimization (PPO) [25], Deep Deterministic Policy Gradient (DDPG) [27], and Twin Delayed DDPG (TD3) [6]. All trials are conducted using the same pointcloud observation type, reward function, and neural network architecture that are used for SNC. Each algorithm is trained using the default hyperparameters that were reported in the original papers. The learning curves are reported in Fig. 6. The PPO algorithm learns slowly and steadily as typical on continuous control tasks [25]. The DDPG learning process is unstable and the algorithm converges to a policy where the robot remains stationary. TD3 learns quickly but does not reach the same final performance of SNC.



(a) Average Episode Reward (b) Average Q-value Prediction

Fig. 5. Comparison of learning curves using for different learning stabilization techniques. Each graph shows the mean and standard deviation of 5 trials.

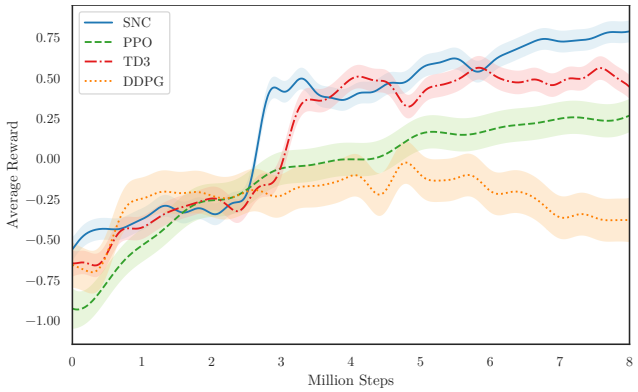


Fig. 6. Learning curves for multiple reinforcement learning methods on the *room* navigation task [15]. Each curve shows the mean and standard deviation from 5 trials.

VI. PHYSICAL ROBOT CONTROL

This section describes how the SNC algorithm was adapted to control a real mobile robot. We train the mobile robot policy using a simulator and transfer the trained policy to the mobile robot. Care is taken to ensure that the simulator dynamics closely match the real-world dynamics.

A. Robot Hardware and Control System

The physical trials in this study are conducted using a TurtleBot 2 mobile robot. The TurtleBot mobile robot is fitted with an RPLidar A2 laser scanner and Intel Realsense RGB-D camera. In addition, the TurtleBot has a three-axis accelerometer, a single-axis gyroscope, and an odometry sensor on each wheel. The global position of the robot is obtained using the Google Cartographer SLAM algorithm [9]. The Robot Operating system (ROS) is used to interface with the TurtleBot base and sensors.

A computer system is developed to control the physical mobile robot using the SNC algorithm, as illustrated in Fig. 7. Additional software components are added to the control system to protect the robot in case of a collision. A velocity smoother is added to both the robot simulator and the physical robot to ensure that the velocity and acceleration of the mobile robot do not exceed predefined limits. In

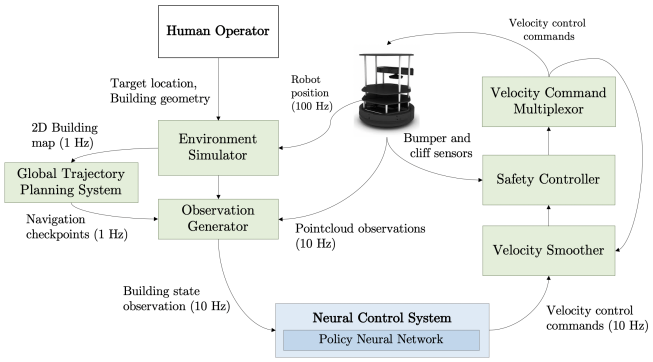


Fig. 7. TurtleBot neural control system.

addition, a safety controller is added to the mobile robot control system. The safety controller uses the Turtlebot bumper sensors to automatically stop the robot in the case of a collision and uses the Turtlebot cliff sensor to stop the robot if it reaches a cliff.

B. Environment Dynamics Matching

In RL, it is common practice to add a small amount of random noise to observations and actions to increase the robustness of the learned controller and account for uncertainty in the true system [5]. However, adding too much noise generally makes the learned controller overly conservative and can lead to degraded performance. Instead, we explicitly quantify the observation and control noise using Gaussian Process (GP) regression [24], and incorporate a similar level of random noise in our robot simulator. A small dataset is collected by gradually varying the Turtlebot velocity controls and recording the resultant velocity using the on-board odometry sensors and SLAM. The scikit-learn software package is used to fit a GP model with a radial-basis kernel to the experimental data [21]. The robot simulator is modified to sample control actions from the fitted dynamics model. For example, when simulating the movement of the robot according to an input control action (v_l, v_r), we first sample a noisy control action (v_l', v_r') from the GP model and feed the noisy velocity into the simulator. Gaussian noise with a 0.1 meter standard deviation is also added to the pointcloud observations to model sensor noise.

C. Training the Control System

Given the large number of samples required for training, it is not feasible to run the training algorithm using the real robot. Instead, the SNC system is trained using the following procedure:

- The SNC system is trained for 12 million timesteps in a simulated environment using the GPR dynamics model and noisy pointcloud data.
- The SNC models are transferred to the real robot and executed at 0.2 second intervals. The SNC observations are taken from a simulator that is synchronized with the real robot position and building geometry. Pointcloud and RGB-D data are collected during the trial but not added to the SNC observations.

(a) 1 million steps (b) 8 million steps

Fig. 8. Learning collision-avoidance policies. (a) Faced with a potential collision, the SNC agents initially slow down and slowly pass each other. (b) After extensive training, the SNC agents both move to the right and pass each other at full speed.

- The SNC system is trained offline using a mixture of simulated transitions and 100,000 real transitions from the physical trials.

In the last stage of the training process, the pointcloud observations are sourced from the RGB-D camera (240 points), the horizontal lidar sensor (120 points) and a 3D model of the building (240 points). The main benefit of this procedure is that the SNC model is gradually introduced to noisier and more complex observations, without needing to directly train the SNC models on the Turtlebot.

D. Experimental results

We empirically demonstrate that the SNC system works well on a robot through a number of physical trials. The robot is required to navigate a room whilst avoiding stationary and moving objects (people). The robot moves quickly and efficiently around static objects as shown in the accompanying video [17]. In addition, the robot can avoid moving objects, but the reaction is sometimes delayed due to latency in the control system.

VII. Discussion

When reviewing SNC navigation policies in a simulated environment, we noticed that the agents learn several interesting behaviors. Specifically, the policies generated using SNC appear to:

- Avoid stairs, cliffs, walls, furniture, and robots, even without explicit programming.
- Implicitly predicts the behavior of other robots to swiftly and safely avoid collisions.
- Resolve deadlock situations where a robot must move against a planned trajectory to let the other robot pass.

The interaction between multiple robots is particularly interesting: Initially, the agents learn to slow down when another robot is nearby to avoid a collision, as illustrated in Fig. 8a. Eventually, the agents learn to safely pass each other at speed, by conditioning on the expected behavior of the other robot, as shown in Fig. 8b.

While SNC is a promising example of how RL can be used to solve the safe navigation problem, there are still many difficulties to overcome before RL can be widely adopted on real robots. In SNC, control actions are computed from raw observations in an end-to-end fashion, often making it difficult to explain why and how certain behaviors emerge. To better understand the learning process, we can project the value function into Euclidean space, as illustrated in Fig. 9.

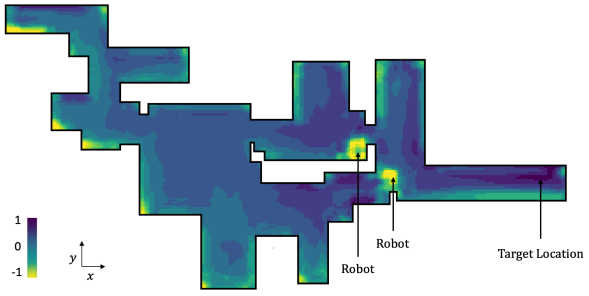


Fig. 9. The value function projected into Euclidean space. The value function is high near the target position and low in locations where collisions are likely, for example near other robots or wall corners.

This projection gives some insight into how the SNC agents learn to avoid hazard-prone regions of space. However, it would be easier to verify SNC policies if SNC produced a local motion plan rather than velocity control commands. Future work could investigate how SNC could be used for local motion planning rather than local control.

VIII. Conclusions

We presented a new method, namely Stochastic Neural Control (SNC) to solve long and challenging navigation problems with reinforcement learning. SNC agents condition their actions on a combination of sensor information and pointcloud data, allowing SNC the control system to safely operate within geometrically complex environments. Furthermore, SNC can learn prior distributions over the movement patterns of other robots, allowing the development of better collision-avoidance policies. Finally, we presented and verified a strategy for transferring SNC from a simulated environment to a real mobile robot.

Acknowledgment

The research is partially supported by the Center for Integrated Facility Engineering (CIFE) at Stanford University.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Konstantinos Charalampous, Ioannis Kostavelis, and Antonios Gasteratos. Recent trends in social aware robot navigation: A survey. *Robotics and Autonomous Systems*, 93:85–104, 2017.

[3] Aleksandra Faust, Kenneth Oslund, Oscar Ramirez, Anthony Francis, Lydia Tapia, Marek Fiser, and James Davidson. PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5113–5120. IEEE, 2018.

[4] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[5] Anthony Francis, Aleksandra Faust, Hao-Tien Lewis Chiang, Jasmine Hsu, J Chase Kew, Marek Fiser, and Tsang-Wei Edward Lee. Long-range indoor navigation with PRM-RL. *arXiv preprint arXiv:1902.09458*, 2019.

[6] Scott Fujimoto, Herke van Hoof, and Dave Meger. Addressing function approximation error in actor-critic methods. *International Conference on Machine Learning (ICML)*, 2018.

[7] Luís Garrote, João Paulo, and Urbano J Nunes. Reinforcement learning aided robot-assisted navigation: A utility and RRT two-stage approach. *International Journal of Social Robotics*, pages 1–19, 2019.

[8] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 2018.

[9] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2D lidar SLAM. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278. IEEE, 2016.

[10] Yong K Hwang and Narendra Ahuja. Gross motion planning — a survey. *ACM Computing Surveys (CSUR)*, 24(3):219–291, 1992.

[11] David Isele, Reza Rahimi, Akansel Cosgun, Kaushik Subramanian, and Kikuo Fujimura. Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2034–2039. IEEE, 2018.

[12] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations (ICLR)*, 2017.

[13] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[14] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E Gonzalez, Michael I Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. *International Conference on Machine Learning (ICML)*, 2018.

[15] Kincho H. Law Max Ferguson. Simulated environments for high-dimensional navigation. `http://digitalpoints.io/about/simulated-buildings`, 2020.

[16] Kincho H. Law Max Ferguson. Source code for stochastic neural control and RL navigation. `https://github.com/maxkferg/stochastic-neural-control`, 2020.

[17] Kincho H. Law Max Ferguson. Stochastic neural control using raw pointcloud data and building information models: Supplementary materials and video. `http://digitalpoints.io/about/stochastic-neural-control`, 2020.

[18] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 1928–1937, 2016.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[20] Ofir Nachum, Michael Ahn, Hugo Ponte, Shixiang (Shane) Gu, and Vikash Kumar. Multi-agent manipulation via locomotion using hierarchical sim2real. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *Conference on Robot Learning (CoRL)*, pages 110–121, 30 Oct 2020.

[21] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Michel, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[22] M. Pfeiffer, S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart, and J. Nieto. Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations. *IEEE Robotics and Automation Letters*, 3(4):4423–4430, 2018.

[23] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 652–660, 2017.

[24] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[26] Peter Trautman and Andreas Krause. Unfreezing the robot: Navigation in dense, interacting crowds. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 797–803. IEEE, 2010.

[27] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI Conference on Artificial Intelligence*, 2016.

[28] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3357–3364. IEEE, 2017.