Learn to Navigate Maplessly with Varied LiDAR Configurations: A Support Point Based Approach

Wei Zhang, Ning Liu, and Yunfeng Zhang

Abstract—Deep reinforcement learning (DRL) demonstrates great potential in mapless navigation domain. However, such a navigation model is normally restricted to a fixed configuration of the range sensor because its input format is fixed. In this paper, we propose a DRL model that can address range data obtained from different range sensors with different installation positions. Our model first extracts the goal-directed features from each obstacle point. Subsequently, it chooses global obstacle features from all point-feature candidates and uses these features for the final decision. As only a few points are used to support the final decision, we refer to these points as support points and our approach as support-point based navigation (SPN). Our model can handle data from different LiDAR setups and demonstrates good performance in simulation and real-world experiments. It can also be used to guide the installation of range sensors to enhance robot navigation performance.

I. INTRODUCTION

Deep reinforcement learning (DRL) [1], which utilizes deep neural networks (DNN) to approximate navigation policy with reinforcement learning, has been widely applied in robot navigation domain. The DRL model shows great advantage in mapless navigation because it can directly handle the range data without any hand-engineered navigation rules compared to fuzzy reactive control [2]. It can also drive the robot out of small local-minimum areas [3] compared to artificial potential field approach [4]. Moreover, compared to the commonly used local planner in SLAM [5], such as dynamic window approach (DWA) [6], the DRL model requires neither local map nor time-consuming forward simulation, and it can adjust its policy quickly when facing a moving obstacles [7].

The most used sensors for DRL-based navigation are range sensors. With a range sensor, such as lidar, the model trained in simulation can be directly deployed to a real robot without any fine-tuning. Tai et al. [8] trained their model in simulation scenarios, and the trained model can generate navigation policy using only ten laser beams. Xie et al. [9] introduced a PID controller to accelerate the training of DRL in simulation. This model is built with convolutional neural networks (CNN), which takes 512 laser beams as input. Pfeiffer et al. [10] used min pooling to down-sample 36 distance values from 1080 laser beams. This operation can handle more laser points and enhance safety during navigation. However, it sacrifices much local information and may lead to a suboptimal policy.

Although these approaches work well in robot navigation applications, the trained DNN model requires a fixed parameter of the employed sensor, i.e., sensor position, field of view (FOV), and angular resolution. These parameters may change due to different task requirements. Fig. 1 shows examples of potential parameter changes of the lidar sensor. Once these parameters change, the original model cannot directly handle the new sensor readings. Although the missing distance values in some direction can be estimated by interpolation, it is difficult to ensure the accuracy of the interpolated value, especially when the new sensor has a smaller FOV and low angular resolution. Moreover, a false estimation of a short distance may cause a collision. In addition, if the input data of the model comes from lidar with low configuration, the navigation performance of the model cannot be improved with the upgrading of the lidar because the input dimension is fixed and much useful local information has to be abandoned. Building a local grid map and inputting the map into a CNN model may allow the change of lidar configuration. However, map building requires additional computation, and the dimension of a high-resolution map is huge. Training a network with high-dimensional and sparse inputs is much harder than the low-dimensional dense inputs. Existing works only operate maps with a low-resolution [12], [13], such resolution is not enough for local obstacle avoidance in crowded scenarios.

In this paper, our model operates on each obstacle point and takes accurate positional information of these points as input. It is inspired by Pointnet [14], which handles semantic segmentation problems of the point cloud. Different from [14], our model incorporates the goal-reaching objective into the point-wise feature extraction. Specifically, we introduce a gate mechanism to guide the network to extract obstacle features that are critical for goal-reaching. Subsequently,
global features are selected from all the feature candidates. The number of global features decides the maximum number of selected points. As only a few points are used to support the final decision, we refer to these points as support points and our approach as support-point based navigation (SPN). This operation is similar to human navigation, in which only a few features, such as sharp corners on the way to the destination, are paid attention to. As our model only operates on each obstacle point, the LiDAR specifications and position can be changed, provided that it can return the obstacle point position in robot frame. Besides, the navigation performance can be improved with the increase of LiDAR resolution because more candidate points can be chosen from. Moreover, our model can be used to guide where to mount the LiDAR sensor such that a better navigation performance can be achieved. To sum up, the contributions of this paper are:

- A human-like navigation algorithm is proposed, which automatically focuses on critical obstacle points and ignores the non-critical ones.
- Our model can generate reliable navigation policy for robot with a new LiDAR setup without any retraining.
- The performance of our model can be improved with the upgrading of the LiDAR sensor.
- The robot navigation performance can be improved through changing the position of the LiDAR under the guidance of the trained model.

The remaining of this paper is organized as follows. A brief introduction of our problem is given in Section II. The proposed SPN approach is described in Section III, followed by the implementation and results in Section IV. Last, we draw the conclusions in Section V.

II. BACKGROUND

A. Problem Formulation

The illustration of mapless robot navigation with varied LiDAR configurations is given in Fig. 2. As shown, a circular robot is required to reach its target without colliding with any obstacles. It carries a LiDAR sensor to perceive its surroundings. The LiDAR can be described using its pose and the specifications. As shown, in the robot frame (the y-axis is robot heading direction), the pose of the LiDAR is described with $[x_l, y_l, \varphi_l]$. Besides, we define the LiDAR angular resolution as $\varphi_a$, the field of view (FOV) as $\varphi_f$ and maximum detection range as $L_{\text{max}}$. We call all the parameters above as LiDAR configurations. In this paper, the robot is required to perform navigation tasks when its LiDAR configurations are changed.

Our problem can be modeled as a sequential decision-making process. During navigation, similar to most mapless navigation problem [8], [9] and [10], the target position $s_t^g = [d_t^g, \varphi_t^g]$ in robot frame is given or directly obtained by sensors [19]. At time $t$, the input $s_t \in S$ of the model contains the LiDAR readings, current robot velocities and the goal position. The dimension of $s_t$ may change with different LiDAR configurations. The action $a_t \in A$ of the robot comprises the commands of linear and angular velocities. Given $s_t$, the robot takes action $a_t$ controlled by the current policy $\pi_t$. With new observations received at time $t+1$, the robot updates its input $s_{t+1}$ and obtains a reward $r_t(s_t, a_t, s_{t+1})$ from the reward function. The objective of this paper is to find an optimal policy $\pi^*$ that maximizes the discount total rewards $G_t = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_{\tau}$, where $\gamma \in [0, 1)$ is a discount factor.

III. APPROACH

We use DRL to learn a DNN model $\pi_\theta$ to approximate the optimal policy $\pi^*$. The DRL algorithm used in this paper is soft actor critic (SAC) [15]. It contains an actor network for approximating the policy and three critic networks (one value network, and two Q networks) for approximating expected total returns. The structures for these networks are given in Fig. 3. As shown, the actor network takes point-wise information as input, while the critic networks share a similar structure with our previous work [16], which take 1D laser scans as input.

A. Actor Network

1) Input representation: To learn DNN-based navigation controller from laser scans, previous works usually format the laser scans as a one-dimensional vector $d = [d_1, \cdots, d_n]$, where $d_i$ denotes the $i$-th scanned distance value and $n$ ($n = 1080$ in this paper) is the number of laser beams. The direction of each laser beam is included by its location in $d$. However, such representations restrict the trained model from handling the data obtained by LiDARs with different FOVs or angular resolutions. To address this problem, in this paper, the laser scans are represented by an $n$-element set $P = \{p_i|i=1,\cdots,n\}$, where $p_i = [\sin \alpha_i/d_i, \cos \alpha_i/d_i]$ and $\alpha_i$ is the relative angle of the obstacle point in robot frame. The same as our previous work [16], we use the reciprocal of distance to highlight the small-distance values in laser scans. According to such representation, the position of each obstacle point is explicitly described, and hence there is no ordering requirement for elements in $P$.

2) Goal-directed point-wise feature extraction: The shadow layers (labeled with orange in Fig. 3(a)) of actor network operate on each point representation $p_i$ individually. As shown, each obstacle point is fed into a dense layer and activated by LeReLU (Leaky Rectified Linear Unit). Meanwhile, the relative position of the goal $s_t^g = [d_t^g, \varphi_t^g]$ and current robot velocity $[v_t, \omega_t]$ are fed into another dense
layer and activated by a sigmoid function. Subsequently, we apply element-wise multiplication to the two features as follows,

$$h_i = lrelu(W_1p_i^T + b_1) \odot \text{sigm}(W_2g^T + b_2) \quad (1)$$

where $W_1 \in \mathbb{R}^{H \times 2}, W_2 \in \mathbb{R}^{H \times 4}, b_1, b_2 \in \mathbb{R}^{H \times 1}$ are network weights and bias of the aforementioned two layers, $\odot$ denotes the element-wise multiplication, $lrelu(\cdot)$ denotes the LReLU function, $\text{sigm}(\cdot)$ denotes the sigmoid function and $g = [d^g, \varphi^g, v_t, \omega]$. The sigmoid function $\text{sigm}(\cdot) \in (0,1)$ here serves as a gate function that helps reserve important obstacle features and remove non-critical ones given the goal position and current velocity.

3) Global feature selection and policy calculation: The following network structure is similar to Pointnet [14]. The filtered feature representation $h_i$ is fed into another dense layer with $K$ output neurons, followed by a max-pooling operation. Through max pooling, the global features of all obstacle points are extracted. The number $K$ of global features is a key factor, which decides the maximum number of obstacle points used for the final decision. The selected obstacle features, combined with goal position and current velocity, are fed into a two-layer DNN for policy calculation. As squashed Gaussian policy is used in this paper, we refer to the sampled stochastic action (used in training) as $\tilde{a}$ and the deterministic action (used in testing) as $\bar{a}$.

B. Critic Networks

As critic networks are not used in testing, their input representation and the network structure are not necessarily similar to the actor network. In this paper, the adopted structure of critic network is similar to [8] and [16]. As shown in Fig. 3(b), all critic networks are fully-connected three-layer DNNs. To reduce the dimension of laser scans and the number of network parameters, the original laser scans are compressed into $m$ ($m = 36$ in this paper) values by 1D minimum down-sampling. Similar to actor network, we choose the reciprocal of the down-sampled value as the input representation $y_i$, which is computed as,

$$y_i = \frac{1}{\min(d_{i,k+1}, d_{i,k+2}, \cdots, d_{i,k+k})} \quad (2)$$

where $i$ is the index of the down-sampled laser scans and $k$ ($k = 30$ in this paper) is the length of down-sampling window. The down-sampled representations of laser scans, together with relative goal position and robot velocity, are fed into the value network to approximate the value function. The double $Q$ networks (Fig. 3(b) top) share the same structure as the value network (Fig. 3(b) bottom) except that it also takes the agent’s action as input.

C. Training Approach

The agent’s learning objective is to reach a given goal point in the shortest time-steps without colliding with the obstacles. To guide the learning process, the reward function contains a positive part $r_s$ for encouraging goal-reaching and a negative part $r_c$ for punishing collision. As the above
In the first 100 episodes, the agent is randomly chosen obstacle-free point and aims to reach a goal scenario arranged by the selection probability. It starts from a keep the selection probability of each scenario until the end. When the selection probability of Env(3) reaches 0.7, we 50 episodes, the selection probability of the next scenario the agent’s success rate in Env(i)

At beginning, the selection probability of the four scenarios are ordered (left to right) by their training difficulty: Env(0) is to train a model that can generate optimal navigation strategies in complex scenarios such as Env(2) and Env(3). To is to train a model that can generate optimal navigation strate-gies in complex scenarios such as Env(2) and Env(3). To

A. Model Training

We train our model in four simulated scenarios using ROS Stage [17]. As shown in Fig. 4, the simulated robot is a circular robot with a radius of 0.2 meters. A LiDAR is mounted on the robot center. Its FOV is 360°, angular resolution is 0.33° and scanning range is 5m. The four training scenarios are ordered (left to right) by their training difficulty: Env(0) is the simplest, and Env(3) is the hardest. The training purpose is to train a model that can generate optimal navigation strategies in complex scenarios such as Env(2) and Env(3). To accelerate training, we adopt curriculum learning as follows. At beginning, the selection probability of the four scenarios Env(0), Env(1), Env(2), Env(3) is [0.7, 0.1, 0.1, 0.1]. When the agent’s success rate in Env(i) reaches 0.9 in the past 50 episodes, the selection probability of the next scenario Env(i+1) exchanges with the selection probability of Env(i). When the selection probability of Env(3) reaches 0.7, we keep the selection probability of each scenario until the end of training.

In each training episode, the robot is spawned in a scenario arranged by the selection probability. It starts from a randomly chosen obstacle-free point and aims to reach a goal point (not rendered). In the first 100 episodes, the agent is controlled by a PID controller [9]. Afterwards, it takes the stochastic action sampled from squashed Gaussian policy. We train the model 500k steps and test the model every 5000 training steps. During testing, in each scenario, the agent is required to reach four goals (labeled with red circles) from the original point with the deterministic policy. To investigate the effect of the number K of global features, we train six models with K ∈ {2, 3, 5, 10, 20, 50}. To compare with conventional input format, we train a model with 1D LiDAR data as input. The input and structure of this model are the same as the value network.

Each model is trained five times using different random seeds. As the dense part in the reward function cannot represent the real navigation objective, the metric used to evaluate the model’s performance in each task is similar to [9], and is represented by a score S function as follows,

where $c_1$ is a scaling constant, and $c_2$ is a small negative constant for punishing staying still.

As the DRL algorithm used for training the agent SAC [15], the expected total return is regularized by the policy entropy and as follows:

where $\mathcal{H}(\pi(\cdot|s_t)) = -\int A \pi(a|s_t) \log \pi(a|s_t) da$ is policy entropy, and $\alpha$ is a weighing factor. During training, the critic networks are optimized to approximate $J(\pi)$ using the Bellman equation, and the actor network learns a policy that maximizes the Q values approximated by the Q network.

IV. IMPLEMENTATION AND TESTS

A. Model Training

We train our model in four simulated scenarios using ROS Stage [17]. As shown in Fig. 4, the simulated robot is a circular robot with a radius of 0.2 meters. A LiDAR is mounted on the robot center. Its FOV is 360°, angular resolution is 0.33° and scanning range is 5m. The four training scenarios are ordered (left to right) by their training difficulty: Env(0) is the simplest, and Env(3) is the hardest. The training purpose is to train a model that can generate optimal navigation strategies in complex scenarios such as Env(2) and Env(3). To accelerate training, we adopt curriculum learning as follows. At beginning, the selection probability of the four scenarios Env(0), Env(1), Env(2), Env(3) is [0.7, 0.1, 0.1, 0.1]. When the agent’s success rate in Env(i) reaches 0.9 in the past 50 episodes, the selection probability of the next scenario Env(i+1) exchanges with the selection probability of Env(i). When the selection probability of Env(3) reaches 0.7, we keep the selection probability of each scenario until the end of training.

In each training episode, the robot is spawned in a scenario arranged by the selection probability. It starts from a randomly chosen obstacle-free point and aims to reach a goal point (not rendered). In the first 100 episodes, the agent is controlled by a PID controller [9]. Afterwards, it takes the stochastic action sampled from squashed Gaussian policy. We train the model 500k steps and test the model every 5000 training steps. During testing, in each scenario, the agent is required to reach four goals (labeled with red circles) from the original point with the deterministic policy. To investigate the effect of the number K of global features, we train six models with K ∈ {2, 3, 5, 10, 20, 50}. To compare with conventional input format, we train a model with 1D LiDAR data as input. The input and structure of this model are the same as the value network.

Each model is trained five times using different random seeds. As the dense part in the reward function cannot represent the real navigation objective, the metric used to evaluate the model’s performance in each task is similar to [9], and is represented by a score S function as follows,

where $T_s$ is the total number of steps spent on navigation. As shown, the robot will get a negative score when its ending status is collision or time out. The learning curves are given in Fig. 5. The training curves in Env(0) and Env(1) are not shown, the robot will get a negative score when its ending

| $S_{max}$ | 1D input | SPN model: No. of global features: K |
|---------|--------|----------------------------------|
| In Env(3) | 14.397 | 14.326 14.352 14.39 14.396 14.404 14.389 |
| In Env(0-3) | 3.616 | 3.597 3.611 3.618 3.619 3.622 3.619 |
B. Support Points Visualization

To better understand the effect of the support points, we visualize them in a simulated map shown in Fig. 6. This map is an $8 \times 8m^2$ square room, which is considered a complex scenario in [9]. Four models SPN($K = 2$), SPN($K = 3$), SPN($K = 5$) and SPN($K = 20$) are used for testing. As shown in Fig. 6, the robot is initialized at the center point (black) and facing right (black arrow). It is required to reach four goal-points (G0-G3) from the initial state respectively. We plot the support points selected by our model every 20 control steps for each task. As shown, the support points are marked with different shapes according to their goal, and their colors are consistent with the colors of the footprints of the robot. As several features may come from the same obstacle point, we amplify the size of the support point marker by $j$ times if $j$ features are extracted from this point. Notably, at beginning of each task, the LiDAR readings are the same, but the support points are different (see black markers). This observation indicates the effect of the goal-directed feature extraction operation, i.e., the model only reserves obstacle features that is valuable for goal-reaching. Moreover, our model can accurately capture the corner points that may affect its navigation in the near future and ignore the nearby points that have been passed or in the opposite direction of its goal.

C. Performance Evaluation in Simulated Scenarios

To evaluate the performance of our model, robots with seven LiDAR configurations containing different angular resolution, FOVs, and LiDAR positions are tested in two simulated scenarios. For simplicity, we label each LiDAR setup with $\{\varphi_f (\degree) | \varphi_a (\degree) | L_{\text{max}} (\text{m}) | y_l (\text{m})\}$, where the first three parameters represent the specification of the LiDAR and the last term represents the LiDAR position (see Fig. 2 for parameter definitions). The other two position parameter $x_l$ and $\varphi_l$ are both zero. The first four LiDAR setups are used in real-world experiments in [8], [9], [10] and [11] respectively, and their detailed setups are given in Fig. 7 and Fig. 8. Note that none of these configurations are used for model training. The simulation scenario Env5, Env6 and Env7 are all $10 \times 10m^2$ square rooms used in [10]. In each scenario, the robot performs 100 navigation tasks with randomly generated initial and goal positions. In Fig. 8, we plot the ratio of different ending status (success, crash or time out) of the robot using different Lidar setup in the four scenarios. As shown, all robots with high-resolution LiDARs ($\varphi_a \leq 1\degree$) can achieve near 100% success rate in Env4 an Env5, which demonstrates a high generalization performance of our model. As we haven’t trained our model with the objective of escaping local-minimum areas, our model’s performance degrades in scenarios containing many large concave areas such as Env6 and Env7. Interestingly, when the resolution of LiDAR is very low ($\varphi_a = 20\degree$), moving the LiDAR forward 0.05m from the robot center can increase the robot’s success rate in all four scenarios, and the success rate in Env5 can be increased by 14%. This observation indicates that our model can also be used to guide the installation of LiDAR to improve robot navigation performance.

D. Real-world Performance Evaluation

We test our model using a turtlebot2 robot in a real-world scenario shown in Fig. 9, in which UTM-30LX LiDAR is used for perceiving the surroundings. As shown, it is mounted at the top-center of the robot and its configuration is different from the one used the training. As target localization sensor is not available, the same as [9], we use Gmapping [18] to pre-build a map of this scenario for robot localization. With robot and goal positions in the map, we can calculate the relative position of the goal in robot frame. It should be noted that this map is only used to calculate the relative position of the goal and is not used by our SPN model. As shown in Fig. 9(b) and Fig. 10(a), the robot starts from the point “S” and is required to reach five targets (“G1” to “G4” and “S”, marked with red crosses on the floor) in succession. This testing scenario contains a very small obstacle (the cylinder) and many obstacles with sharp corners. The trajectories of the robot are recorded and plotted...
in Fig. 10(a). As shown, the robot reaches five destinations successfully, and the chosen paths are near-optimal.

1) Reaction to sudden changes: Last, we test the model’s capability of addressing a suddenly appearing obstacle. As shown in Fig. 10(b) and the attached video, the researcher (labeled with “Oi” in the map), serving as a dynamic obstacle, suddenly blocks the way of the robot. The robot in point “i” must adjust its navigation strategy to avoid potential collisions and find another path. The trajectories and video show that our robot can well adapt to such abrupt changes and quickly switch to another path.
Fig. 8. The ratios of different ending results for robots with seven different LiDAR configurations in four simulated scenarios (Env4-Env7, 100 runs in each scenario).

Fig. 9. Real-world testing setup. (a) A turtlebot2 and its LiDAR configurations marked with red crosses on the floor.

Fig. 10. Navigation trajectories of turtlebot2 robot. (a) Trajectories in the static testing scenario, (b) Trajectories in the testing scenario with sudden appearing obstacles.

V. CONCLUSIONS

In this work, we present a learning-based approach for mapless navigation with varied LiDAR configurations. Our model selects support points from the obstacle points and can make near-optimal navigation decisions. It is purely trained in simulation and can be deployed to a real robot without any fine-tuning. Extensive simulation and real-world experiments have been conducted to evaluate our model. The experimental results demonstrate that the trained model can be used for a robot with a different LiDAR setup and can achieve high navigation performance.

REFERENCES

[1] V. Mnih et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529-533, 2015.
[2] W. Li, C. Ma and F. Wahl, “A neuro-fuzzy system architecture for behavior-based control of a mobile robot in unknown environments,” Fuzzy Sets and Systems, vol. 87, no. 2, pp. 133-140, 1997.
[3] O. Zhelo, J. Zhang, L. Tai, M. Liu, and W. Burgard, “Curiosity-driven exploration for mapless navigation with deep reinforcement learning,” 2018. [Online]. Available: https://arxiv.org/abs/1804.00456.
[4] Y. Hwang and N. Ahuja, “A potential field approach to path planning,” IEEE Trans. Robot. Autom., vol. 8, no. 1, pp.23-32, 1992.
[5] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte and M. Csorba, “A solution to the simultaneous localization and map building (SLAM) problem,” IEEE Trans. Robot. Autom., vol. 17, no. 3, pp. 229-241, 2001.
[6] D. Fox, W. Burgard and S. Thrun, “The dynamic window approach to collision avoidance,” IEEE Robot. Autom. Mag., vol. 4, no. 1, pp. 23-33, 1997.
[7] Y. Wang, H. He and C. Sun, “Learning to Navigate Through Complex Dynamic Environment With Modular Deep Reinforcement Learning,” textitIEEE Trans. Games, vol. 10, no. 4, pp. 400-412, 2018.
[8] L. Tai, G. Paolo, and M. Liu, “Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation,” in IEEE Int. Conf. Intell. Robot. Syst. (IROS), pp. 31-36, Sept., 2017.
[9] L. Xie, S. Wang, S. Rosa, A. Markham, and N. Trigoni, “Learning with training wheels: Speeding up training with a simple controller for deep reinforcement learning,” in Proc. - IEEE Int. Conf. Robot. Autom. (ICRA), pp. 6276-6283, 2018.
[10] M. Pfeiffer et al., “Reinforced Imitation: Sample Efficient Deep Reinforcement Learning for Mapless Navigation by Leveraging Prior Demonstrations,” IEEE Robot. Autom. Lett., vol. 3, no. 4, pp. 4423-4430, 2018.
[11] J. Lim, S. Ha, and J. Choi, “Prediction of Reward Functions for Deep Reinforcement Learning via Gaussian Process Regression,” textitIEEE/ASME Trans. Mechatronics, vol. 25, no. 4, pp. 1739-1746, 2020.
[12] G. Chen et al., “Robot Navigation with Map-Based Deep Reinforcement Learning,” 2020. [Online]. Available: https://arxiv.org/abs/2002.06320.
[13] D.S. Chaplot, D. Gandhi, S. Gupta, A. Gupta, and R. Salakhutdinov, “Learning To Explore Using Active Neural SLAM,” in Int. Conf. Learn. Represent. (ICLR), 2019.
[14] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 652-660, 2017.
[15] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in Int. Conf. Mach. Learn. (ICML), pp. 2976-2989, 2018.
[16] W. Zhang, Y. Zhang, and N. Liu, “Map-less Navigation: A Single DRL-based Controller for Robots with Varied Dimensions,” 2020. [Online]. Available: https://arxiv.org/abs/2002.06320.
[17] “stage/Tutorials - ROS Wiki,” Wiki.ros.org, 2020. [Online]. Available: [http://wiki.ros.org/stage/Tutorials](http://wiki.ros.org/stage/Tutorials) [Accessed: 14-Oct-2020].

[18] “gmapping - ROS Wiki,” Wiki.ros.org, 2020. [Online]. Available: [http://wiki.ros.org/gmapping](http://wiki.ros.org/gmapping) [Accessed: 14-Oct-2020].

[19] C. Chen, Y. Chen, Y. Han, H. Q. Lai, F. Zhang, and K. J. R. Liu, “Achieving Centimeter-Accuracy Indoor Localization on WiFi Platforms: A Multi-Antenna Approach,” IEEE Internet Things J., vol. 4, no. 1, pp. 122-134, 2017.