Socially-Aware Robot Navigation

David Betancourt
School of Computational Science & Engrg.
Georgia Institute of Technology
david.betancourt@gatech.edu

Nithin Shrivatsav Srikanth
Georgia Institute of Technology
nithinshri@gatech.edu

Kapil Vuthoo
School of Computer Science
Georgia Institute of Technology
kapilv@gatech.edu

***Abstract*—Socially-aware navigation is an important aspect of Human-Robot interaction that is gaining popularity in the research community. Deep reinforcement learning is a framework which looks promising to solve the problem of socially-aware navigation in a way where we can generalize to different crowds without hand-coded rules, by modeling pairwise interaction features between humans-robot and human-human and constructing a crowd state representation vector. We show that our reinforcement learning algorithm performs better than other recently released algorithms in simulation space. For the robot implementation in Turtlebot2, the reinforcement learning algorithms resulted in actions which did not correspond to the current state. Nonetheless, a separate non-reinforcement learning algorithm for socially-aware navigation was successfully implemented in Turtlebot and tested with humans.**

***Index Terms*—HRI, social navigation, Deep Reinforcement Learning, Turtlebot2**

I. Introduction

The main motivation behind this project is to learn about mobile robot navigation algorithms in environments crowded by pedestrians. In these situations, we need to make use of concepts from the field of human-robot interaction and enable the robots to respect the social norms of humans.

Today, an increasing number of mobile robots are being built to interact with humans which will naturally result in the development of autonomous navigation algorithms that consider social norms to avoid collisions. Enabling robots to follow social conventions will guarantee the comfort of the surrounding humans and will in turn ensure that the humans can also infer the robot's intentions. This can be dealt with using a reaction-based or a trajectory-based approach. The reaction-based approach uses a reciprocal velocity obstacle—a reaction based method that adjusts each agent's velocity vector to ensure collision-free navigation. However, it does not consider evolution of the neighboring agents' future states, they are short-sighted in time and have been found to create oscillatory and unnatural behaviors in certain situations. In contrast, trajectory-based methods explicitly account for evolution of the joint (agent and neighbors) future states by anticipating other agents' motion. For this project we have decided to explore the later and use deep reinforcement learning algorithms to teach robots to follow social conventions and carry out human-aware navigation. Some common examples of social norms are passing on the right side of a pedestrian, over-taking a pedestrian on the left side, and not crossing a queue if there is space to bypass it. This type of navigation would be the focus of this class project and we believe it would be interesting to learn how to model the navigation of robots with the additional social enhancements that we have learned in class.

II. Related Work

A few recent methodologies to develop socially-aware navigation algorithms for ground robots have been published. One approach [6], [7] decouples perception and planning such that the robot uses two different algorithms for each task. This approach has led into a problem known as the *freezing robot* problem, where the robot cannot find an optimal path. An enhanced approach stems from considering the instantaneous cooperation that occurs to avoid collisions in human-to-human interactions, which some researchers seek to adapt to this HRI task. In particular, recent work [2]–[4] uses deep reinforcement learning with reward shaping in a game theoretic setting to minimize the number of collisions with humans. In such reinforcement learning setting, rewards account for the social norms and are shaped (biased) to account for cultural-specific behavior, like passing on the right. Our initial algorithm was based on the seminal work of [2], [3]. However, other recent approaches subsumed this work and in addition better handle crowd-robot interaction (CRI) by also modeling human-human interactions with multiple agents [1], [5]. In particular, the work in [1] explicitly models the crowd dynamics using a deep neural network with an attention mechanism.

III. Background

a) MPDs and POMDPs: A Markov Decision Process (MDP) is a model for sequential decision-making and planning, and is a paradigm regularly used in robot motion [10]. MDP's assume that the state of the environment is fully observable to the agent. In most real-world applications, the Markov Property in MDPs does not hold because the agent cannot observe the full state of the environment. When this happens, the MPD becomes a Partially Observable Markov Decision (POMDP) and the agent only receives observations $o \in \mathcal{Z}$, instead of complete states $s \in \mathcal{S}$. A POMDP is defined by the following experience tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{Z}, \mathcal{O}\}$. At each time step t, the environment is in state $s_t \in \mathcal{S}$, and the agent takes an action $a_t \in \mathcal{A}$ according to a policy $\boldsymbol{\pi} : \mathcal{O} \times \mathcal{A} \mapsto [0, 1]$, which causes the environment to transition to the new state s_{t+1} according to the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}$. The agent

then receives an observation $o_t \in \mathcal{Z}$, which depends on the new state of the environment with probability $\mathcal{O}(o_t|s_{t+1}, a_t)$. The agent then receives rewards according to its actions and the state $r : \mathcal{S} \times \mathcal{A} \mapsto \Re$. In MDPs and POMDPs, the goal of the agent is to maximize the expected future cumulative discounted rewards $\mathcal{R} = \mathbb{E}\big(\sum_{t=0}^{T} r^t \gamma^t\big)$, where γ is the discount factor.

b) Value Iteration Networks: The value of a state s following a policy $\boldsymbol{\pi}$ is $V^{\boldsymbol{\pi}}(s) = \mathbb{E}_{a\sim\boldsymbol{\pi}}\big[\sum_{t=0}^{\infty} r(s,a)\gamma\big]$. The optimal value function $V^*(s)$ is the maximal expected return from a state s, which equals $\max_{\boldsymbol{\pi}} V^{\boldsymbol{\pi}}(s)$. A policy is optimal ($\boldsymbol{\pi}^*$) if $V^{\boldsymbol{\pi}^*}(s) = V^*(s)$. Value Iteration is an algorithm which allows to calculate $V^*(s)$ and $\boldsymbol{\pi}^*$. In a value iteration network, the value function is evaluated by a deep neural network.

c) Prioritized Experience Replay: Prioritized experience replay (PER) developed in [9], uses the Temporal-Difference (TD) error and importance sampling to prioritize the order in which experiences are replayed by the RL agent. The work in [9] outperformed previous deep RL implementations, with uniform experience replay, in almost all the games in the Atari benchmark.

IV. Methodology

A. Main Algorithm

We adopted the algorithm of [1] and enhanced it by modifying the way that the RL agent samples past experiences. The algorithm consists of using value iteration initialized with demonstrations. The algorithm, SARL, seeks to model the relative importance of humans and encode the collective impact of the crowd. The multi-agent framework of SARL is composed of three modules, as follows:

a) Interaction Module: This module embeds the motion information of each of the agents (humans and robot) with respect to their neighbors. This is achieved by constructing a neighborhood map of size $L \times L \times 3$ around each human i, which encodes the velocities of existing neighbors, as

$$M_i(\alpha,\beta) = \sum_{j\in\mathcal{Q}_i} \mathbb{1}[(\Delta x, \Delta y) = (\alpha,\beta)] s_j^h \tag{1}$$

Where (α,β) are the cell's coordinates; $(\Delta x, \Delta y)$ are the difference in planar coordinates between humans i and j; Q_i is the set of neighboring humans around the i^{th} human; s_j^h is the local state vector for human j; and $\mathbb{1}[\cdot]$ is the indicator function.

For each human i the state is a 7-tuple, defined as $s_i^{h,(t)} = [p_x, p_y, v_x, v_y, r_i, d_i, r_i + r]^{(t)}$, where p are positions, v are velocities, r are radii, and d separation distance at time t. For the robot, the state is a 5-tuple, defined as $s^{(t)} = [d_g, v_{pref}, v_x, v_y, r]^{(t)}$; where d_g is distance to goal, v_{pref} is preferred velocity, v are velocities, and r is the radius at time r. The embedding of the state of the robot s, the states of humans s_i^h, and neighborhoods maps M_i, into a high-dimensional tensor e_i is achieved with a neural network Φ_e parameterized by Θ_e, as

$$e_i = \Phi_e(s, s_i^h, M_i; \Theta_e) \tag{2}$$

Finally, the embedding vector e_i is then trained end-to-end with a deep neural network Ψ_h parameterized by Θ_h giving the pair-wise interaction h_i between the robot and human i, as

$$h_i = \Psi_h(e_i; \Theta_h) \tag{3}$$

b) Pooling Module: In order to handle crowds of varying sizes, the pooling module allows the network to process an arbitrary number of inputs into a fixed-sized output action space. To this end, SARL uses an attention mechanism in order to learn the relative importance of each neighbor, as well as the collective force of the crowd. The attention score is achieved through hidden layers with ReLu activations Ψ_α, parameterized by Θ_α, as follows

$$\alpha_i = \Psi_\alpha(e_i, e_m; \Theta_\alpha) \tag{4}$$

Where e_i is defined in Eq. 2 and $e_m = \frac{1}{n}\sum_{k=1}^{n} e_k$. Then, the final representation of the crowd is given by c, as

$$c = \sum_{i=1}^{n} \mathbf{softmax}(\alpha_i) h_i \tag{5}$$

Where h_i is defined in Eq. 3.

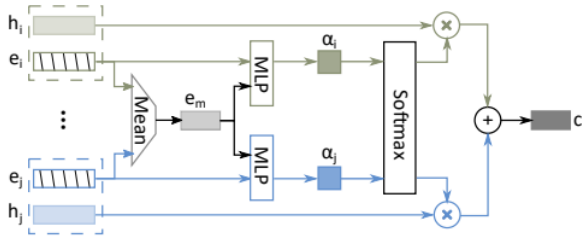

Fig. 1. Pooling Module Architecture from [1].

c) Planning Module: Using the compact representation of the crowd c, the planning module estimates the state value V for cooperative planning, parameterized by Θ_V, as

$$V = f_V(s, c; \Theta_V) \tag{6}$$

1) Model Details: The hidden units of the network functions are as follows: $\Phi_e = (150, 100); \Psi_h = (100, 50); \Psi_\alpha = (100, 100); f_V = (150, 100, 100)$.

2) Reward Function: The reward function is defined as follows:

$$R_t(s_t^{joint}, a_t) = \begin{cases} -0.25 & if \quad d_t < 0 \\ -0.1 + d_t/2 & elif \quad d_t < 0.2 \\ 1 & elif \quad p_t = p_g \\ 0 & otherwise. \end{cases}$$

Where d_t is the minimum separation distance between the robot and the crowd during each time interval.

3) Training Details: We implemented prioritized experience replay (PER) [9] as an algorithmic enhancement for [1] and also implemented reward shaping as in [3]. Adding PER reduces the training time significantly and we noticed that surprisingly none of the algorithms in the literature used PER, but rather only sampled uniformly from the experience buffer. Additionally, PER helps to infer the POMDP hidden states in our setting—the intentions of humans. The model is trained in PyTorch using Adam with a mini-batch size of 32. The learning rate for imitation learning is 0.01 and 0.001 for RL. The action space contains 80 actions (5 speeds exponentially spaced between (0,v_{pref}] and 16 headings evenly spaced between [0,2π).

B. Secondary Algorithm

We also implemented and tested the algorithm of [8] and successfully implemented it on TurtleBot2. The algorithm is a planning-based approach to address the problem of socially-aware navigation. It proceeds by making use of a layered social cost function to generate collision-free paths. Some of the social norms hard-coded in the robot are avoiding the personal space of humans, waiting for a human to pass and moving out of the way when blocking a human's path. The algorithm involves the use of time-dependent, search-based path planning with dynamic, social cost maps containing costs based on predicted human trajectories and Gaussian social cost model. One of the main reasons for choosing this algorithm as an alternative is its ability to consider motion of humans over time. We tested this algorithm on Turtlebot in a narrow and cluttered environment. The navigation framework consists of the following modules.

a) Search Graph: To generate robot trajectories that are feasible and take into account the robot's inertial constraints, the state space is modeled in the following way:

$$C = (x, y, \phi, v, \omega, t)$$

—this corresponds to the unicycle model of the robot. Then, a discrete time model with a constant sampling time (Δt) and a finite set of directly executable actions u_i is used to build the Search Graph. The action set is as follows: $U = \{(e_x a_x, e_\phi a_\phi) | e_x, e_\phi \in \{-1, 0, 1\}\}$. The actions are a combination of forward and angular accelerations resulting in motions which can have constant acceleration, constant deceleration or zero acceleration. This set of state space and actions ensures the generation of feasible motions through the use of a feasibility tree.

b) Social Cost Model: The social space preference of humans can be modeled using either potential functions or social costs and this approach makes use of a Gaussian cost model with an amplitude A and σ_x,σ_y along the person's front and side directions. To induce the social norm of moving the robot on the right side of the human and preventing it from moving in the sensitive area in front of the person, the Gaussian distribution is perturbed from the person's geometric center by a small value. In addition, to respect the proxemics of the person, a non-traversable area of radius r_0 is created around the person.

c) Dynamic Cost Map: A layered dynamic cost map encodes the static environment constraints with the dynamic social constraints where each layer represents one snapshot of the predicted human-trajectory from time-step i to $i + 1$. The cost map has a single static layer and multiple dynamic layers with each layer being a two-dimensional grid that represents the navigation constraints of the robot.

d) Optimization: The parameters being optimized in the A^* search algorithm are path execution time, path length, static environment constraints and social constraints with each parameter being represented by a cost c_i. Each of these costs are described below:

- Path execution time is proportional to the required time-steps to reach goal.
- The path length cost function is a function of the linear and diagonal transitions associated with a planned path in the cost map grid.
- The static environment and social constraints cost function is obtained from the dynamic cost map according to the grid cell that the path traverses.

The Cost Function is as follows:

$$C = \Sigma w_i . c_i \tag{7}$$

e) A^* Heuristic: This paragraph describes the robot's behavior when no people are around. To generate the shortest path of the robot a planar eight-connected Dijkstra expansion is imposed on the static layer of dynamic cost map.

f) Planning Timeout: To ensure that the planning frequency is constant the time allocated for planning is restricted. Once the maximum planning time is reached, the algorithm returns the path to the best expanded state which can be the goal state in an ideal scenario. If the goal was not reached, the algorithm returns the lowest cost path that reached the goal at any orientation and velocity or expand the path to the state that would be expanded next.

1) Model Details: The local map for TurtleBot2 is generated using Simultaneous Localization and Mapping technique

by leveraging the gmapping package of Robot Operating System(ROS). The GMapping algorithm makes use of Rao-Blackwellized particle filters to learn grid maps and localize the robot.

C. TurtleBot Implementation

We started with programming the turtlebot for normal navigation using all the sensors and built in utilities in ROS like navstack and rviz for local and the global planner. To prepare the turtlebot for addition of deep learning algorithms, we integrated the object detection for collision avoidance and subsequently added people detection. Further, the CADRL and SARL algorithms were trained and the trained model put on the tutlebot.

As discussed in the above sections a non-communicating multiagent collision avoidance problem can be formulated as a partially-observable sequential decision making problem. In this implementation the agent's state vector is a combination of two parts: $s_t = [s_t^o; s_t^h]$ where s_t^o denotes the observable part that can be measured by all other agents, and s_t^h denotes the hidden part that is only known to the agent itself. Each part of the state vector is a further combination of p - position, v - velocity , r - radius and h_a - heading angle. $s^o = [p_x, p_y, v_x, v_y, r]$ $s^h = [p_{gx}, p_{gy}, v_{pref}, h_a]$

We took the origin as the start point of the robot in the map and then by subscribing to Odometry we got the position coordinates to this origin in 2D. Own velocity was being published to cmd vel, therefore known and for the moving people (Agents) the position and velocity is calculated from the lidar data. Further, the heading was also taken from Odom data. Because of the mismatch of the training data and the expected data, the inference engine was not generating proper velocity commands. We tweeked and tried many combinations of the state vectors and the paper [1] is also silent about physical implementation as against using the gym environment.

a) Implementation Details: Throughout the entire project ROS was a very important ingredient. ROS stands for Robot Operating System and is a meta operating system for writing modular robot software. The modularity and powerful communication capabilities of ROS allowed us to create such a complex task in a simple manner.

The TurtleBot2 was abstracted as a unicycle model to pass the motion commands for navigation. The TurtleBot2 is equipped with ASUS Xtion Pro sensor which is an RGB-D sensor but is used as a Lidar for our application. This sensor is based in primesense infra-red technology. The sensor outputs data in the form of a laserscan which is a linear vector of ranges from the robot to the nearest obstacle in various directions. A ROS node extracts the middle few rows of the Asus Xtion Pro's depth image and filters the output to generate data similar to the data from a Lidar.

The TurtleBot2 is built on a Kobuki mobile base which works on differential drive mechanism. The base has sensors such as wheel encoders and gyroscope, actuators for movement and a power source. The sensors of the robot are used in generating the odometry information of the robot such as its pose. To operate it a Netbook with ROS capabilities is needed and we have used the ASUS Netbook of TurtleBot2 and this laptop is configured as a ROS_MASTER_URI.

The codes were first tested on simulation using Gazebo and then the codes were ported to hardware. Results of both the simulation and hardware are attached in the Results section.

The lattice_planner ROS package is used as the time-dependent global planner and timed_path_follower ROS package is used for local planning. The lattice_planner generates a path based on A^* search algorithm for robots with differential drive constraints. It directly uses the dynamic cost map for planning the paths of the robot. The timed_path_follower provides a trajectory tracking controller in the ROS Navigation Stack.

To test the behavior of the robot around humans, a fake detection node is executed in ROS. The navigations goals of the robot are set in RViz and the robot smoothly navigates to the desired goal while avoiding humans and following social constraints.

V. Results

A. Comparison of Algorithms

Table 1 shows the comparisons of the main reinforcement learning algorithm implemented (SARL) vs two other recent approaches (CADRL [2] and LSTM-RL [4]) during testing. For all approaches, the algorithms were trained using prioritized experienced replay, which significantly reduced the training time (from 20k episodes to 4k episodes). SARL-PER had the lowest collision rate, while CADRL-PER was faster. The LSTM-RL-PER algorithm did not perform well, perhaps, it needed more training episodes but we wanted to do a comparison with similar training times. Fig. 3 shows the navigation time and the rewards per episode of training, we see that CADRL-PER and SARL-PER are very similar during training, while LSTM-RL is not converging.

TABLE I
COMPARISON OF RL ALGORITHMS

Algorithm	Collision Rate	Success Rate	Time to Goal (sec)
SARL-PER	0.018	0.99	11.23
CADRL-PER	0.054	0.99	10.78
LSTM-RL-PER	0.63	0.58	22.94

B. TurtleBot2 Implementation Results

The RViz visualizations for the TurtleBot navigation in an environment with humans is depicted in the pictures below. As seen from the pictures, the navigation of the TurtleBot is smooth due to the even switching between the various navigation behaviors represented by the layered cost map. The re-planning of path takes place when the planned path conflicts with human motion. The red cylinder around the human is a restricted radius which the robot will never get close to. The blue layer around the human represents the Gaussian distribution with an offset to represent a region of high potential.

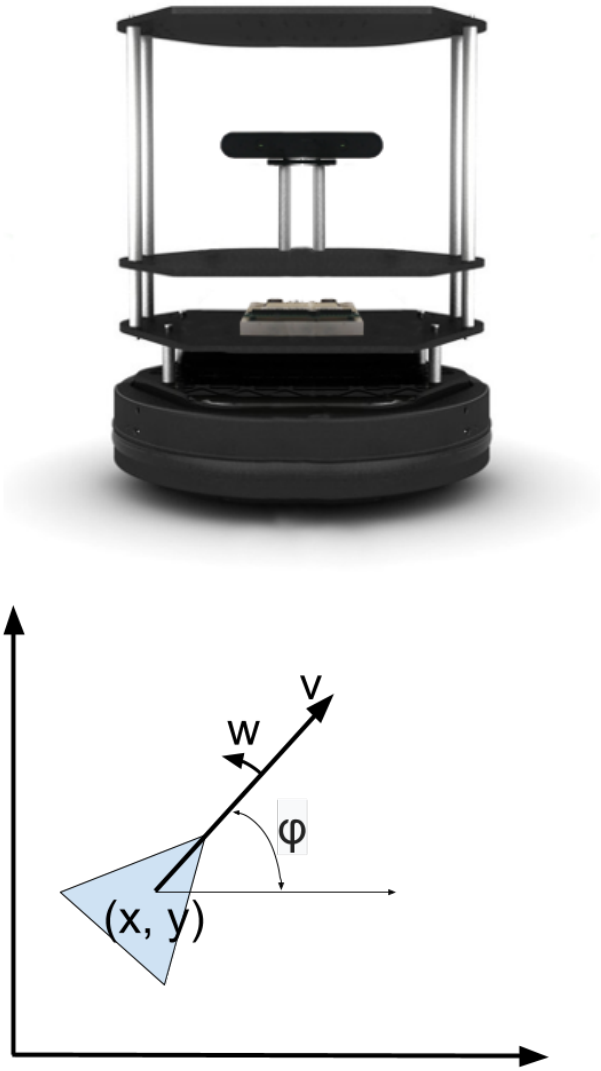

Fig. 2. TurtleBot2 and Unicycle Model

A real-world experiment with the TurtleBot2 was tested for the condition of moving to the right side of a person and is depicted in Figure 5.

VI. Conclusion

Socially-aware robot navigation research is a subject that is gaining a lot of attention, due to some immediate needs where robots must navigate through crowds—i.e., airports, shopping malls. We conclude that although SARL-PER performed very well in the Gym-AI simulations, it did not work in Turtlebot. Perhaps, the main reason for this is that SARL and CARDL were originally implemented with robots that were not Turtlebot2. Nonetheless, this gave us insight into how to implement these algorithms in the future, especially as long as our experience with ROS is not advanced. Thus, for a future implementation of RL algorithms in ROS, we envision starting with an algorithm that is already implemented in the same hardware and build from that algorithm to make it suit our goals. Such workflow is more similar to how we implemented the secondary non-RL (cost map) algorithm successfully.

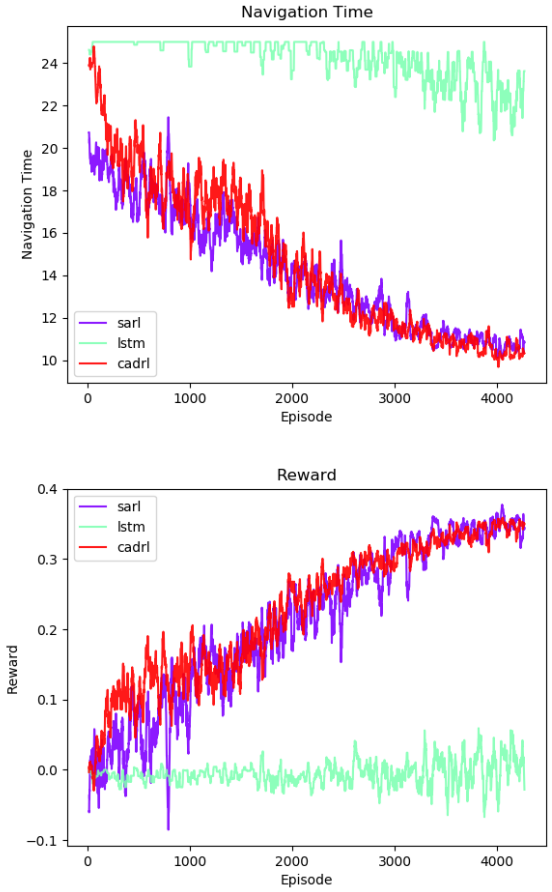

Fig. 3. Results per Episode of Training
Top: Navigation Time.
Bottom: Reward.

Acknowledgment

We wish to thank our professor, Sonia Chernova, and our Teaching assistant, Siddhartha Banerjee, for providing us Turtlebot2 and the helpful guidance.

References

[1] Changan Chen, Yuejiang Liu, Sven Kreiss, and Alexandre Alahi. Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. *arXiv preprint arXiv:1809.08835*, 2018.

[2] Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan P How. Socially aware motion planning with deep reinforcement learning. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 1343–1350. IEEE, 2017.

[3] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 285–292. IEEE, 2017.

[4] Michael Everett, Yu Fan Chen, and Jonathan P How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. *arXiv preprint arXiv:1805.01956*, 2018.

[5] Tingxiang Fan, Xinjing Cheng, Jia Pan, Dinesh Monacha, and Ruigang Yang. Crowdmove: Autonomous mapless navigation in crowded scenarios. *arXiv preprint arXiv:1807.07870*, 2018.

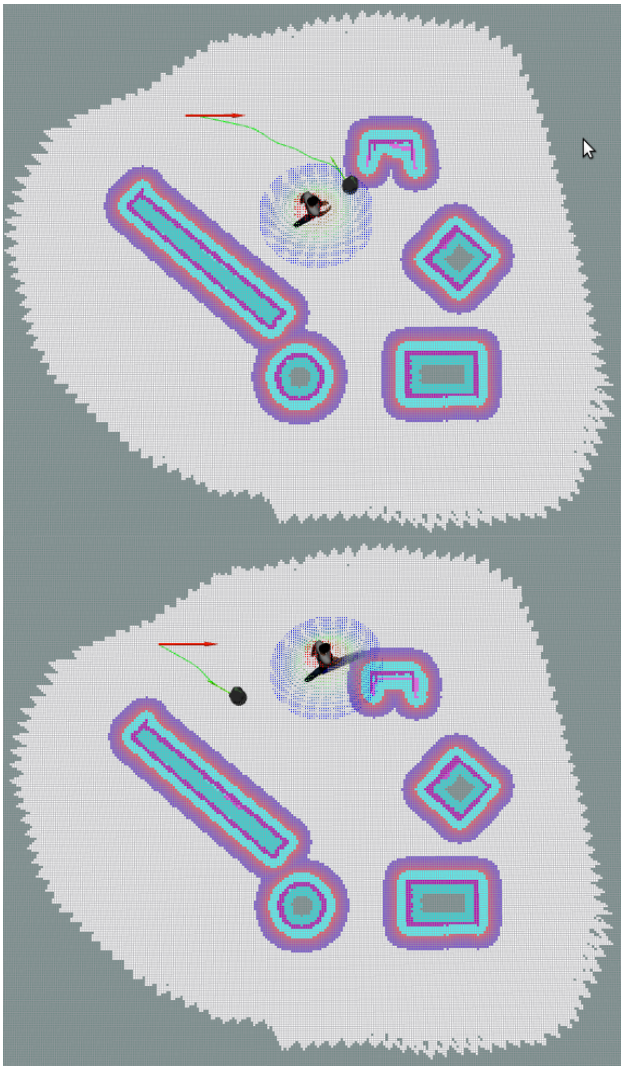

Fig. 4. Visualizations of the Layered Cost Map Algorithm

Fig. 5. Hardware Implementation

[6] Ioannis Karamouzas and Stephen J Guy. Prioritized group navigation with formation velocity obstacles. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 5983–5989. IEEE, 2015.

[7] Sujeong Kim, Stephen J Guy, Wenxi Liu, Rynson WH Lau, Ming C Lin, and Dinesh Manocha. Predicting pedestrian trajectories using velocity-space reasoning. In *Algorithmic Foundations of Robotics X*, pages 609–623. Springer, 2013.

[8] M. Kollmitz, K. Hsiao, J. Gaa, and W. Burgard. Time dependent planning on a layered social cost map for human-aware robot navigation. In *2015 European Conference on Mobile Robots (ECMR)*, pages 1–6, Sept 2015.

[9] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[10] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.