SOCIALGYM: A Framework for Benchmarking Social Robot Navigation

Jarrett Holtz[1] and Joydeep Biswas[1]

***Abstract*— Robots moving safely and in a socially compliant manner in dynamic human environments is an essential benchmark for long-term robot autonomy. However, it is not feasible to learn and benchmark social navigation behaviors entirely in the real world, as learning is data-intensive, and it is challenging to make safety guarantees during training. Therefore, simulation-based benchmarks that provide abstractions for social navigation are required. A framework for these benchmarks would need to support a wide variety of learning approaches, be extensible to the broad range of social navigation scenarios, and abstract away the perception problem to focus on social navigation explicitly. While there have been many proposed solutions, including high fidelity 3D simulators and grid world approximations, no existing solution satisfies all of the aforementioned properties for learning and evaluating social navigation behaviors. In this work, we propose SOCIALGYM, a lightweight 2D simulation environment for robot social navigation designed with extensibility in mind, and a benchmark scenario built on SOCIALGYM. Further, we present benchmark results that compare and contrast human-engineered and model-based learning approaches to a suite of off-the-shelf Learning from Demonstration (LfD) and Reinforcement Learning (RL) approaches applied to social robot navigation. These results demonstrate the data efficiency, task performance, social compliance, and environment transfer capabilities for each of the policies evaluated to provide a solid grounding for future social navigation research.**

I. INTRODUCTION

Deploying robot navigation safely alongside people such that they move in a social manner is one of the ultimate goals of robotics. However, training and evaluating in real-world human environments present both safety concerns and scalability challenges for many learning algorithms, and this difficulty compounds the difficulty of developing robust approaches to robot social navigation. As such, an extensible benchmark for social navigation is a critical step along the path to deploying socially compliant robots.

In order to accommodate the rapidly expanding pool of promising machine learning techniques a framework for these benchmarks should provide the infrastructure for integrating approaches. In addition, the ability to adjust the reward, observation space, and properties of the agent and environment are important for capturing social navigation scenarios and enabling varied learning approaches. Finally, such a framework should provide a representation with reasonable simulation fidelity while also abstracting away the perception task to focus on social navigation. Many different solutions for simulating robots and humans in dynamic environments have been proposed for social navigation research. However, to the best of our knowledge no existing solution satisfies all of the desired properties for a benchmark framework.

[1]Computer Science, University of Texas, Austin, TX, 78712, USA jaholtz@utexas.edu, joydeepb@cs.utexas.edu

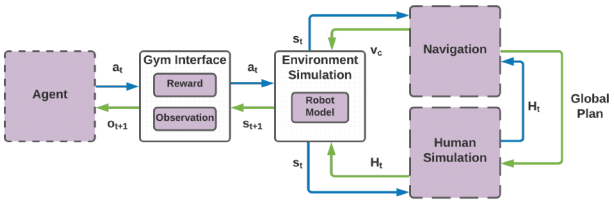

Fig. 1: Diagram of interaction between modules in SOCIALGYM. Dashed boxes represent interchangeable modules, while purple boxes contain configureable parameters. Blue arrows represent requests between modules, and green arrows represent responses.

Our proposed solution is SOCIALGYM, a 2D simulation environment built on the Robot Operating System (ROS) to provide a lightweight and configurable option for training and evaluating social navigation behaviors: https://github.com/ut-amrl/social_gym . 2D simulation is chosen to provide an option that abstracts away the perception problem to focus on interactions between navigation and people in dynamic human environments. To streamline integration in common reinforcement learning workflows, SOCIALGYM implements the OpenAI gym interface for training and evaluation. SOCIALGYM is modular such that modules can be exchanged for others to increase the range of possible benchmarks, an overview of module interactions and configurability is shown in Fig. 1, and a visualization of the simulation environment in Fig. 2. The included benchmark in SOCIALGYM focuses on the action selection problem presented in work on Multipolicy Decision Making [1] and program synthesis for social navigation [2], where social navigation is modelled as an action selection problem, and the optimal action is selected from a set of discrete sub-policies. Using this included benchmark, we compare and contrast a suite of engineered approaches, model-based symbolic learning, Learning from Demonstration (LfD), and Reinforcement Learning (RL); to provide a baseline for future comparisons; and highlight promising future directions for research. Our results demonstrate the data efficiency, scenario performance, social compliance, and generalizability of different policies, and our analysis highlights the promising strengths and weaknesses of different techniques. In addition to providing the benchmark for multipolicy decision-making in social navigation, SOCIALGYM is designed to be configurable and extensible in the hopes of providing a stable base for a variety of social navigation benchmarks.

The contributions of this work are as follows:

- a 2D simulation environment for robot social navigation,
- a complete toolkit that includes robot simulation, human simulation, navigation stack, and simple baseline behaviors,
- OpenAI gym integration to provide a standard interface for many learning approaches,
- a benchmark based on multipolicy decision making for robot social navigation, and
- evaluation results and analysis for several reinforcement learning and learning from demonstration approaches for use as baselines for future research.

II. Related work

Model-based approaches with application-specific engineered behaviors have historically been applied to social navigation [1], [3], [4], [5], [6]. More recently, approaches have sought to leverage machine learning techniques to learn general social navigation behaviors [7], [8], [9], [10], [11], [12]. Prior work has explored the use of program synthesis and learning from demonstration to learn social navigation behaviors as symbolic programs [13], [14], [2].

Existing simulators and benchmarks are either at the level of high-fidelity 3D simulation that focuses on the raw perception problem of humans in the environment or are simplified grid world models of navigation. In contrast, SocialGym provides sensor reading at the level of laser scans and human detections.

Many general-purpose high-fidelity 3D simulators have been developed for robotics. Of particular note is the Gazebo simulator [15], a powerful and extensible simulator commonly used with ROS. Other high-fidelity simulators have been developed using gaming physics and graphics engines such as Unity [16] or Unreal [17] with autonomous driving in mind, some of which can simulate other agents in the environment, such as AirSim [18] and CARLA [19]. In addition to these more general-purpose simulators, some solutions have been proposed specifically for the social navigation problem. The Social Environment for Autonomous Navigation (SEAN) [20] uses a similar Unity-based approach to more general simulators while providing social navigation-specific metrics and scenarios. While SEAN does not provide any particular benchmark, SocNavBench [21] is designed as a social benchmark that generates photo-realistic sensory input directly from pre-recorded real-world datasets to strike a balance between simulation and recorded datasets. In addition to simulation-based benchmarks, a small number of purely dataset-based benchmarks have been proposed, including SocNav1 [22] and Social Robot Indoor Navigation (SRIN) [23]. While datasets are a critical benchmark tool, they often require an initial sensor processing step and are not easily extensible. Alternatively, simplified grid-world type environments, such as MiniGrid [24], can also be used to approximate social navigation by modelling humans with dynamic obstacles, although this is not commonly used for benchmarking.

III. Social Navigation in Robotics

We frame social navigation as a discounted-reward partially observable Markov Decision Process (POMDP) $M = \langle S, A, T, R, \Omega, O, \gamma \rangle$ consisting of the state space S where $s \in S \mapsto \langle p_r, v_r, p_g, H \rangle$ and p_r is the robot pose, v_r is the robot velocity, p_g is the goal pose, and $h_i \in H \mapsto \langle p_{hi} v_{hi} \rangle$ is a list of the human poses and velocities; actions A represented either as discrete motion primitives [1] or continuous local planning actions [25]; the world transition function

$$T(s, a, s') = P(s_{t+1} = s' \mid s_t = s, a_t = a) \tag{1}$$

for the probabilistic transition to states s' when taking action a at previous state s; the reward function $R : S \times A \times S \mapsto \mathbb{R}$; the set of observations $o \in \Omega$; a set of conditional observation probabilities O; and discount factor γ. The solution to this POMDP is represented as a policy $\pi : S \times A \mapsto A$ that decides what actions to take based on the previous state-action pair. The optimal social navigation policy π^* maximizes the expectation over the cumulative discounted rewards:

$$\begin{aligned} \pi^* &= \arg_\pi \max J_\pi, \\ J_\pi &= E\left[\sum_{t=0}^{t=\infty} \gamma^t R(s_t, \pi(s_t, a_t), s_{t+1})\right] \end{aligned} \tag{2}$$

We next describe how SocialGym provides modules to simulate the components of the POMDP and interfaces with different approaches to learn π^*.

IV. SocialGym System architecture

SocialGym is built from four modules that simulate different components of the POMDP for social robot navigation and coordinate together to simulate the full POMDP. These modules are the **Gym** module, the **Environment** module, the **Human** module, and the **Navigation** module.

The Gym module is the top-level interface between the agent and the simulator that handles simulation of the agent's action selection policy to select an action $a_t \in A$ at each timestep t based on the current observation $o_t \in \Omega$. The Gym module then steps the simulation by passing a_t to the Environment module, which returns a new state $s_{t+1} \in S$ from which the Gym module calculates the reward r_t and derives a new observation $o_{t+1} \in \Omega$. The Environment module handles the representation of the state $s_t \in S$ and coordinates with the Human and Navigation modules to simulate the transition function T given a_t at each timestep. The Human module takes in s_t and a_t and controls the components of T governed by the humans' response to the environment by sending updated human positions and velocities to the Environment module. In turn, the Navigation module controls the components of T related to robot execution of a_t by providing a baseline navigation behavior for evaluation and global and local planners for use in discrete action spaces.

While our current implementation provides a single complete benchmark scenario and extensions to enable various others, it is essential to note that each module can be freely replaced with any alternative that implements a prerequisite

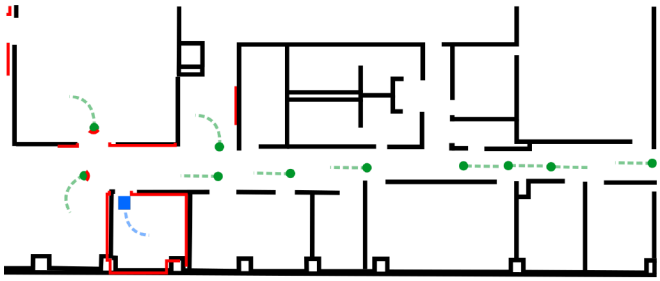

Fig. 2: Visualization of one timestep of SOCIALGYM with humans shown as green circles, the walls as black lines, the laserscan as redlines, the robot as a blue box, and recent trajectories as dotted lines.

ROS service. In this way, ROS services make our approach accessible to a wide variety of existing approaches written in ROS. The components of SOCIALGYM and their interactions are shown in Fig. 1, and in the following sections, we will describe the abstractions of each module and high-level technical details, while in § IV-E, we will describe the concrete instantiation used for our included benchmark.

A. Gym Integration

The Gym model builds on the OpenAI Gym framework to interface between our ROS-based simulation framework and the learning agent. This module has four responsibilities: coordinating the generation and resetting of scenarios, receiving actions $a \in A$ from the agent that are used to coordinate state transition of the environment, receiving states $s \in S$ from the environment and converting them into observations $o \in \Omega$, and evaluating performance by calculating metrics and the value of the reward function $R \mid s, a$.

a) Generating scenarios: A scenario is described as a tuple $\langle m, t, p_s, p_g, H \rangle$, where m is the map describing the static obstacles in the scene, t is the timestep, p_s is the start pose of the robot, p_g is the goal pose of the robot, and H is a list of pairs $\{\langle p_h^i, p_g^i \rangle\}$ that describes the initial pose p_h^i and the goal pose p_g^i for each human, i, in the scene. SOCIALGYM randomly generates new scenarios based on several parameters that describe the range of possible initial conditions and goals. These parameters are m, the maximum and minimum number of humans h_{max}, h_{min}, the number of iterations for a given scene N_r, and the maximum total iterations N_m. In addition, scenario generation requires a list of poses G that describes the list of legal start and end locations in the map. Given this configuration, a new scenario is generated as follows: 1) m and G are configured by the user, 2) a random p_s and p_g are selected from H, 3) a random n is chosen such that $n_{min} < n < n_{max}$, 4) n humans are generated by selecting a random p_h^i and p_g^i from G such that they do not overlap, 5) and finally, the environment is initialized and the initial state is sent to the agent. Each random scenario is run until it completes in either success or failure N_r times, and then a new scenario is generated up to the maximum number of iterations N_m. We define the success state to be when $|p_r - g| < \epsilon_s$, and the failure state to be when $t > t_f \wedge |p_r - g| > \epsilon_s$.

b) Updating the environment and observation: The Gym module controls the update loop of the simulation as follows: 1) the Gym module receives an action a_t from the agent, 2) a_t is sent as a request to the environment, 3) the Environment module executes T given a_t, 4) the Gym module receives a response state s_{t+1}, 5) an observation function $f(s \in S) \mapsto o \in \Omega$ is employed to map s_{t+1} to an observation o_{t+1} where o_{t+1} is an observation vector containing both discrete and continuous variables, 6) the reward is calculated using $R(s_{t+1}) \mapsto r_t$, and finally 7) r_t and o_{t+1} are passed to the agent for the next iteration. For use with, and in addition to calculating reward, the Gym module can optionally produce several metric values for use in performance evaluation. The provided Gym module supports the following metrics: 1) time to goal: t when the agent reaches p_g; 2) distance from goal: $|p_{t+n} - g|$; 3) distance traveled: $\sum_{t+1}^{t+n} |p_t - p_{t-1}|$; 4) force: approximates maximum force exerted between the robot and humans $\text{argmax}_i \, e^{|p_r - p_h^i|}$; 5) blame: takes into account velocity at times of close encounters; penalizing robot velocities towards humans. Let p_r^i be the closest point on the line segment from p_r to $p_r + v_r$ to p_h^i, then blame is calculated as: $\text{argmax}_i \, \Phi(|p_r - p_h^i|)$; 6) human collision count, and 7) static obstacle collision count.

B. Environment simulation

The Environment module is responsible for representing the state $s_t \in S$ and updating s_t based on robot actions a_t and the state transition function T.

The environment updates the robot state according to the robot motion model and a_t. For discrete actions, a_t must first be converted to continuous velocities by the Navigation module. In addition to updating the robot pose, the environment also needs to update all humans in the scene. However, since a_t does not provide velocities for the humans, we need to describe how humans will move in response to the environment. Here the Human module § IV-C utilizes the updated state from the environment to calculate new positions and velocities for each human in the scene.

C. Human simulation

The Human module is responsible for simulating the components of T concerned with how humans behave. This requires updating H based on some model of humans moving through the environment towards a goal.

Each human h^i in the scene moves back and forth between a starting position h_p^i, and a goal pose h_g^i, by planning a global path of intermediary nodes between the two that avoids static obstacles and employs a separate local planner to handle dynamic obstacles. For global planning, we utilize the Navigation module as described in § IV-D. The local planner SOCIALGYM employs is PedsimROS [26], a ROS module that models human behaviors according to the social force model [3], [4]. In brief, the Social Force Model represents each agent in the environment as exerting repelling force on each agent, while goals exert attractive forces. We refer

the reader to the original work [3] and the PedsimROS implementation [26].

D. Navigation

The Navigation module serves three purposes. For general use, the navigation module provides a global planning interface for use by the human module and implementations of baseline deterministic behaviors that do not require learning that can be used as comparisons. For our included benchmark scenario, the navigation module further provides implementations of *actions* that make up the discrete sub-policies of our action space.

The global planning component of Navigation is responsible for planning an obstacle-free path from a start location to a goal location. Global planning receives a request of the form $\langle m, p_s, p_f \rangle$, where p_s is a start position in the map m and p_f is the target position and returns a response of the form $\langle \{p_s, p_j, ...p_{j+n}, p_f\} \rangle$, where p_j represents an intermediary local goal along an obstacle-free path from p_s to p_f.

The local planner is responsible for avoiding obstacles immediately visible to the robot and calculating control commands to take the robot to the next waypoint on the global plan. The local planner is only called as part of the complete navigation solution that receives a request of the form $\langle m, p_s, p_f \rangle$, and returns a response of the form $\langle v_c \rangle$, representing command velocity to be executed by the Environment. For the local planner, we implement a trajectory rollout planner [27] that interfaces with the global planner to get the next local goal, and plans obstacle-free trajectories if possible, and comes to a stop otherwise.

E. Social Action Selection Benchmark Scenario

As part of SOCIALGYM, we include a complete benchmark for social navigation action selection in a multipolicy setting similar to the one employed for [2], [1]. At each timestep, the agent is responsible for choosing one action from a discrete set of actions representing the existing sub-policies of a navigation behavior. These sub-policies are designed as robust behaviors built on the deterministic navigation module presented in § IV-D.

A given benchmark in SOCIALGYM is defined by the choice of module implementations, action space, observation space, and reward function and can be additionally customized for complexity and difficulty via the configuration parameters of the underlying modules. Examples of these configurable parameters include 1) the environment map as defined by the environment module, 2) the parameters of the humans, 3) the robot motion model and kinematics, 4) the amount of observation or execution noise in the simulation. In the following, we will describe the observation, action, and reward spaces for the Social Action Selection Benchmark. Other benchmarks could be designed by replacing one or more of the components described here. For more information on the underlying configuration of the submodules, we refer the reader to the supplementary material and implementation.

We define the discrete action space to contain four sub-policies: stopping in place (Halt), navigating towards the next goal (GoAlone), following a human (Follow), and passing a human (Pass). The observation space $\langle p_g, p_l, v_r, H_r \rangle$ consists of the global and local goal poses p_g and p_l, respectively, the robot velocity v_r, and the relative poses and velocities of the humans visible to the robot H_r, with all non-visible human poses and velocities set to 0. As such, the observation function needs to zero out the observations of all humans that are **not** visible to the robot because they are occluded by static obstacles in the environment. We choose this observation space to minimize the environment-specific features in the observation by using only local coordinates and velocities to minimize the risk of overfitting to specific training examples.

Finally, the reward function is a weighted linear combination of three metrics from Multipolicy Decision Making [1]: d_g the distance gained towards the goal since the last timestep, f the maximum force between the robot and a human, and finally, b the maximum blame resulting from the robot's actions and the humans in the environment. Additionally, the agent receives a bonus c if the goal is reached during a timestep. This reward function can be represented using weights w_i as: $r = (w_1 * d_g + w_2 * f + w_3 * b) +$ `if (success) c else` 0.0. The weights can be used to specify user or task-specific preferences with respect to the tradeoff between social compliance and time to goal.

V. BENCHMARK AND EVALUATION

To demonstrate results obtained with SOCIALGYM we provide experimental results using the benchmark scenario described in § IV-E, and a series of baseline learned and engineered policies. Our reference implementations include two Reinforcement Learning (RL) implementations from the Stable-Baselines project [28], Proximal Policy Optimization (PPO) and Deep Q-Networks (DQN), two Learning from Demonstration (LfD) approaches, Behavior Cloning (BC) and Generative Adversarial Imitation Learning (GAIL) from the HumanCompatibleAI project [29], as well as two engineered navigation solutions, trajectory rollout based navigation and a social reference solution (Ref) and a symbolic learning from demonstration approach called Physics Informed Program Synthesis (PIPS) [2]. We evaluate the learned techniques on three criteria in three sets of experiments. First we evaluate the training efficiency of each technique by comparing the number of steps and amount of data needed for learning reasonable policies in § V-A. Then we evaluate the performance of each policy in terms of the social metrics defined in § IV-A, and compare the learned policies to the engineered policies in § V-B. Finally, we consider the generalizability of the learned policies by transferring them to a new simulated environment and comparing their performance in § V-C.

A. Data Efficiency and Learning Rates

The learning algorithms we evaluate fit into two major categories that each require different learning procedures. Reinforcement learning approaches learn via interactive interaction with the environment and Learning from Demonstration approaches require apriori demonstrations of the desired behavior. For each set of techniques, we evaluate and compare

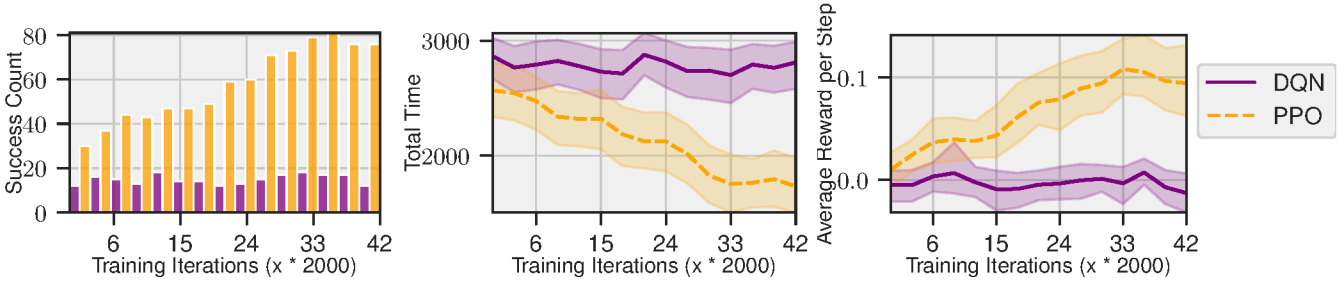

Fig. 3: Progressive performance with more training iterations. The shaded regions represent the 90% confidence interval.

the training process, learning rate, and data efficiency during learning procedures utilizing SOCIALGYM in the following paragraphs.

a) Reinforcement Learning: Training new RL algorithms is a plug-and-play process that enables the agent to interact with SOCIALGYM and iteratively update its policy. To evaluate the sample efficiency of the considered RL approaches, we trained each model for 84000 total timesteps and created a checkpoint of the learning progress every 2000 steps. For each model, we evaluate their performance on the same 100 test scenarios and report their performance in terms of the reward described in § IV-E in Fig. 3. Our evaluation results show two key results, first, both RL algorithms require 84000 or more timesteps before peaking in performance, and second the PPO approach is significantly more data efficient than the DQN approach we evaluated. In general, the DQN approach is particularly weak to the initial conditions, and performs more poorly than the other evaluated approaches on the small sample of 100 trials used for these experiments.

b) Learning from Demonstration: To evaluate the data efficiency of the considered LfD approaches, we generated a single demonstration set with 155483 timesteps from our reference behavior and trained multiple policies with decreasing quantities of data evaluating each model on 100 test scenarios, as shown in Fig. 4. We report the performance in terms of the success rate and reward earned. Notably, PIPS cannot learn policies with more than 3500 data points, as the symbolic synthesis approach used by PIPS does not scale to large numbers of demonstrations. Conversely, while the DNN-based approaches can be trained with 3500 examples or less, their performance decreases significantly as the amount of data used for training is reduced. Only performing similarly to the PIPS-based policies when 155400 samples are used. There are two likely reasons for this improved data-efficiency. First, the structure used for synthesis provides additional constraints, and second, PIPS selectively subsamples the data by windowing around transitions between actions, an optimization that is not part of the more general purpose neural approaches.

Training Size	BC		GAIL		PIPS	
	(%)	r	(%)	r	(%)	r
155400	84	0.12	59	0.06	-	-
116600	16	0.01	40	0.04	-	-
77700	37	0.04	38	0.02	-	-
38800	70	0.08	21	0.02	-	-
3500	49	0.05	27	0.01	88	0.16

Fig. 4: Performance of LfD approaches with varying numbers of demonstrations in terms of the success rate (%) and the average timestep reward **r**.

B. Performance of Learned Models

To evaluate and compare the learned behaviors, we evaluate them on the same randomly generated set of 2000 trials. We report the results in terms of four primary metrics, successful trials, force, blame, and time to goal as described in § IV-A. We report these metrics as opposed to the score or reward to better compare between LfD and RL approaches, and to better position the results with respect to task performance. We report the percentage of successful trials as a table in Fig. 7 and the metric results in Fig. 5.

In addition to the learned behaviors, Fig. 7 and Fig. 5 feature the engineered GoAlone behavior utilizing only trajectory rollout as described in § IV-D, and the engineered Reference behavior (Ref) used for comparison and generating demonstrations for the LfD algorithms. In Fig. 7, a trial is counted as successful if the goal is reached within a bounded time without any collisions between the robot and humans or walls in the environment. In terms of pure success rate, no social behavior is as efficient as the GoAlone policy, likely due to the halting robot problem, where the robot is often left waiting for humans to pass long enough that the scenario times out. When comparing the learned behaviors, we find that the LfD behaviors have a better overall success rate than the RL approaches, suggesting that continuous scenario demonstrations better convey the delayed reward of reaching the goal than the reward function. When we look at the metric performance of each approach, we see the success rates reflected in the time to goal, with the more successful approaches featuring an overall lower time to goal, while the less successful approaches are slower. An exception to this is the GAIL policy, which was neither the most or least successful while also being the slowest policy. As would be expected, we see a tradeoff between time to goal and social compliance reflected to some degree in both the force and blame graphs, with the slower behaviors exhibiting lower force and blame. In particular, BC and PIPS roughly match the performance of the Ref behavior from which demonstrations were drawn in terms of all three metrics, while in contrast, GAIL, PPO, and DQN optimized for social compliance over time to goal.

C. Environment Transfer Performance

To evaluate the ability of the learned policies to transfer between environments, we introduce a new set of 2000 scenarios using a novel map with a significantly different

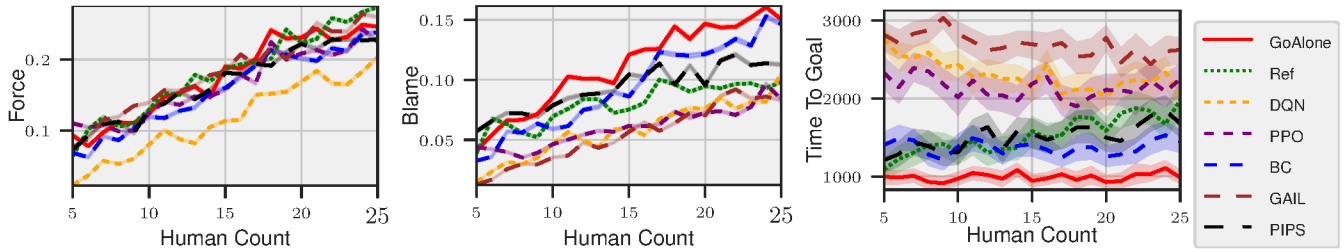

Fig. 5: Performance of evaluated policies in the training environment. The shaded regions represent the 90% confidence interval.

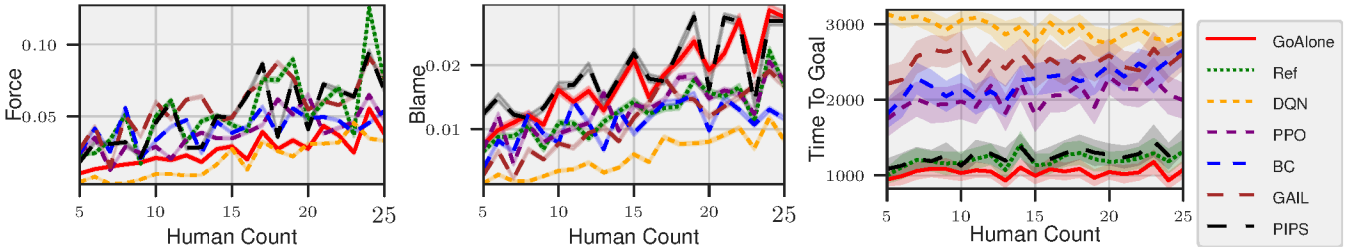

Fig. 6: Performance of evaluated policies in a novel environment. The shaded regions represent the 90% confidence interval.

Policy	Success Rates (%)	
	Test	Transfer
GoAlone	97.5	97.3
Ref	89.4	92.4
BC	92.3	49.9
GAIL	83.3	18.9
PIPS	88.9	89.2
DQN	54.1	8.8
PPO	55.1	47.2

Fig. 7: Success rates in the training environment and after transferring to a novel environment.

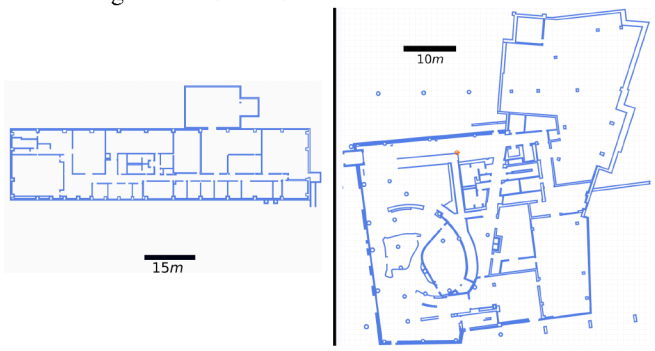

Fig. 8: Environment used for training and evaluation on the left, and new environment for transfer evaluation on the right.

configuration than the map used for policy training. This new map consists of a different set of static obstacles and a new set of possible start and goal locations for both the robot and the humans. For comparison, the environment used for the training and the evaluation environment used for this experiment are shown in Fig. 8.

We evaluate each policy on the same 2000 random trials drawn from the new environment and report the performance in terms of force, blame, and time to goal in Fig. 6 and in terms of the success rate in Fig. 7. The first key result is that the engineered policies (Goalone, Ref) do not degrade in success rate during the transfer between environments, while all but the PIPS-learned policy degrade significantly. Similarly, the time to goal metric reflects this, with more successful behaviors achieving lower average times to goal and behaviors more likely to fail achieving much slower average times to goal. The new environment is less difficult for the engineered policies, as the distribution of humans is sparser in the larger, more open, map. This suggests that despite our efforts to design an observation space and reward that are environment-agnostic, the non-symbolic learning algorithms are still making decisions that are highly environment-dependent. In terms of the social metrics, the results for the learned policies reflect what we stated about the engineered policies. The new environment is more manageable with lower force and blame achieved by all behaviors thanks to the more sparse human distributions. These results demonstrate what we would expect, that the model-based and symbolic approaches perform better in environment transfers than black-box model-free approaches.

VI. Discussion and Future Work

In this paper, we presented SocialGym, a configurable simulator and benchmarking tool for socially navigating robots. SocialGym provides an interface for easily integrating learning algorithms with a 2D simulator designed to abstract away perception and localization and focus on the difficult task of learning robust robot behaviors for navigating amongst humans in complex environments. Further, we present empirical results from evaluating a suite of baseline algorithms in our provided social action selection benchmark that demonstrate the utility of SocialGym. While SocialGym provides an extensible interface, it is difficult to imagine that any one solution will perfectly fit the needs of myriad researchers, and we provide only a single initial benchmark evaluation case initially. It is our hope that SocialGym will reduce the barrier to entry for evaluating learning algorithms for social robot navigation by allowing other researchers to build on an extensible foundation for social navigation learning and evaluation.

VII. Acknowledgments

This work is supported in part by NSF (CAREER2046955, SHF-2006404), ARO (W911NF-19-20333, W911NF-21-20217), and Northrop Grumman Mission Systems.

References

[1] D. Mehta, G. Ferrer, and E. Olson, "Autonomous navigation in dynamic social environments using multi-policy decision making," in *IROS*, 2016, pp. 1190–1197.

[2] J. Holtz, S. Andrews, A. Guha, and J. Biswas, " Iterative Program Synthesis for Adaptable Social Navigation ," in *IROS*, 2022.

[3] D. Helbing and P. Molnár, "Social force model for pedestrian dynamics," *Phys. Rev. E*, vol. 51, pp. 4282–4286, May 1995.

[4] G. Ferrer, A. Garrell, and A. Sanfeliu, "Robot companion: A social-force based approach with human awareness-navigation in crowded environments," in *IROS*, 2013, pp. 1688–1694.

[5] J. Mumm and B. Mutlu, "Human-robot proxemics: Physical and psychological distancing in human-robot interaction," in *HRI*, 2011, pp. 331–338.

[6] K. Charalampous, I. Kostavelis, and A. Gasteratos, "Robot navigation in large-scale social maps: An action recognition approach," *Expert Systems with Applications*, vol. 66, pp. 261 – 273, 2016.

[7] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *IROS*, 2017, pp. 1343–1350.

[8] T. V. D. Heiden, C. Weiss, N. S. Nagaraja, and H. V. Hoof, "Social navigation with human empowerment driven reinforcement learning," *ICANN*, vol. abs/2003.08158, 2020.

[9] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *ICRA*, 2019, pp. 6015–6022.

[10] P. Ciou, Y. Hsiao, Z. Wu, S. Tseng, and L. Fu, "Composite reinforcement learning for social robot navigation," in *IROS*, 2018, pp. 2553–2558.

[11] L. Tai, J. Zhang, M. Liu, and W. Burgard, "Socially compliant navigation through raw depth inputs with generative adversarial imitation learning," in *ICRA*, 2018.

[12] H. Karnan, A. Nair, X. Xiao, G. Warnell, S. Pirk, A. Toshev, J. Hart, J. Biswas, and P. Stone, "Socially compliant navigation dataset (scand): A large-scale dataset of demonstrations for social navigation," *IEEE Robotics and Automation Letters*, pp. 1–8, 2022.

[13] J. Holtz, A. Guha, and J. Biswas, " Interactive Robot Transition Repair With SMT ," in *IJCAI*, 2018, pp. 4905–4911.

[23] K. M. O. and A. B.R., "Srin: A new dataset for social robot indoor navigation," *Glob J Eng Sci.*, 2020.

[14] ——, " Robot Action Selection Learning via Layered Dimension Informed Program Synthesis ," in *CORL*, 2020.

[15] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, Sep 2004, pp. 2149–2154.

[16] J. K. Haas, "A history of the unity game engine," 2014.

[17] Epic Games, "Unreal engine." [Online]. Available: https://www.unrealengine.com

[18] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," *CoRR*, vol. abs/1705.05065, 2017. [Online]. Available: http://arxiv.org/abs/1705.05065

[19] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[20] N. Tsoi, M. Hussein, J. Espinoza, X. Ruiz, and M. Vázquez, "Sean: Social environment for autonomous navigation," in *Proceedings of the 8th International Conference on Human-Agent Interaction*, ser. HAI '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 281–283. [Online]. Available: https://doi.org/10.1145/3406499.3418760

[21] A. Biswas, A. Wang, G. Silvera, A. Steinfeld, and H. Admoni, "Socnavbench: A grounded simulation testing framework for evaluating social navigation," *THRI*, 2021.

[22] L. Manso, P. Trujillo, L. Calderita, D. R. Faria, and P. Bachiller, "Socnav1: A dataset to benchmark and learn social navigation conventions," *ArXiv*, vol. abs/1909.02993, 2020.

[24] M. Chevalier-Boisvert, L. Willems, and S. Pal, "Minimalistic gridworld environment for openai gym," https://github.com/maximecb/gym-minigrid, 2018.

[25] D. V. Lu, D. B. Allan, and W. D. Smart, "Tuning cost functions for social navigation," in *Social Robotics*, G. Herrmann, M. J. Pearson, A. Lenz, P. Bremner, A. Spiers, and U. Leonards, Eds. Cham: Springer International Publishing, 2013, pp. 442–451.

[26] Okal, Billy and Linder, Timm, "PedsimROS." [Online]. Available: https://github.com/srl-freiburg/pedsim_ros

[27] B. P. Gerkey, "Planning and control in unstructured terrain," 2008.

[28] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable baselines3," https://github.com/DLR-RM/stable-baselines3, 2019.

[29] S. Wang, S. Toyer, A. Gleave, and S. Emmons, "The `imitation` library for imitation learning and inverse reinforcement learning," https://github.com/HumanCompatibleAI/imitation, 2020.