Iterative Program Synthesis for Adaptable Social Navigation

Jarrett Holtz[1] and Simon Andrews[2] and Arjun Guha[3] and Joydeep Biswas[1]

***Abstract*— Robot social navigation is influenced by human preferences and environment-specific scenarios such as elevators and doors, thus necessitating end-user adaptability. State-of-the-art approaches to social navigation fall into two categories: model-based social constraints and learning-based approaches. While effective, these approaches have fundamental limitations – model-based approaches require constraint and parameter tuning to adapt to preferences and new scenarios, while learning-based approaches require reward functions, significant training data, and are hard to adapt to new social scenarios or new domains with limited demonstrations.**

In this work, we propose Iterative Dimension Informed Program Synthesis (IDIPS) to address these limitations by learning and adapting social navigation in the form of human-readable symbolic programs. IDIPS works by combining program synthesis, parameter optimization, predicate repair, and iterative human demonstration to learn and adapt model-free action selection policies from orders of magnitude less data than learning-based approaches. We introduce a novel predicate repair technique that can accommodate previously unseen social scenarios or preferences by growing existing policies.

We present experimental results showing that IDIPS: 1) synthesizes effective policies that model user preference, 2) can adapt existing policies to changing preferences, 3) can extend policies to handle novel social scenarios such as locked doors, and 4) generates policies that can be transferred from simulation to real-world robots with minimal effort.

I. Introduction

Social navigation is a fundamental robot behavior that is integrally tied to human preferences, and that needs to be robust to potentially unknown; environment-specific decision making. State-of-the-art approaches to social navigation fall into two major categories, model-based and model-free approaches. Model-based approaches include those that employ engineered models of social constraints, such as the social force model [1] or human-robot proxemics [2]. However, no single model can capture all possible social constraints, and adapting models requires tedious parameter tuning [3]. Model-free approaches include those that leverage Neural Networks (NNs) for Learning from Demonstration [4] or Reinforcement Learning [5]. However, these approaches suffer from limitations common to NNs: they are data-intensive [6], difficult to understand [7], and challenging to adapt to new domains without starting over [8].

Recent work on Layered Dimension Informed Program Synthesis (LDIPS) [9] has shown that program synthesis, when extended with dimensional analysis, addresses many of these concerns by learning robot behaviors as human-readable programs. However, while LDIPS can learn new behaviors, it does not address policy adaptation. In this work, we build upon LDIPS to present Iterative Dimension Informed Program Synthesis (IDIPS) to synthesize and adapt social navigation behaviors from human demonstration. A highlight video of our approach can be found at `https://youtu.be/JoT8nZ_Rsto`.

Given a set of demonstrations and an optional starting behavior, IDIPS produces a new minimally altered behavior consistent with the demonstrations. Iterative application of IDIPS allows for continuous refinement based on further demonstration. IDIPS employs three modules to accomplish this: a synthesis module for learning new behaviors, a parameter optimization module for adjusting real-numbered parameter values, and a predicate repair module for adjusting logical components. For the synthesis module, we extend LDIPS with MaxSMT constraints to handle potentially conflicting human demonstrations by allowing for partial satisfaction [10]. For the parameter repair module, we employ SMT-Based Robot Transition Repair (SRTR) [11], an approach for transition repair based on human corrections. Finally, for the predicate repair module, we introduce a novel technique for adapting the conditional logic with the minimal syntactic changes to accommodate new demonstrations. An open-source implementation of our approach can be found at `https://github.com/ut-amrl/pips`.

We evaluate IDIPS in simulation and on real robots to demonstrate the following: 1) IDIPS can synthesize effective policies for social navigation that model the preferences of distinct demonstration sets, 2) synthesized policies can be automatically adapted towards the preferences of alternative demonstrations with a small number of corrections, 3) IDIPS can adapt synthesized policies to unseen scenarios, such as a locked door, with a small number of corrections, and 4) synthesized programs can be transferred to a real-world mobile service robot and adapted.

II. Background and Related Work

We frame social navigation as a discounted-reward Markov Decision Process (MDP) $M = \langle S, A, T, R, \gamma \rangle$ consisting of the state space S that includes robot, human, and environment states; actions A represented either as discrete motion primitives [12] or continuous local planning actions [3]; the world transition function

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

for the probabilistic transition to states s' when taking action a at previous state s; the reward function $R : S \times A \times S \mapsto \mathbb{R}$; and discount factor γ. The solution to this MDP is represented as a policy $\pi : S \times A \mapsto A$ that decides what actions

[1]University of Texas at Austin. {jaholtz, joydeepb}@utexas.edu
[2]University of Massachusetts, Amherst. sbandrews@umass.edu
[3]Northeastern University. a.guha@northeastern.edu

to take based on the previous state-action pair. The optimal social navigation policy π^* maximizes the expectation over the cumulative discounted rewards

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

The social navigation problem has been extensively studied, and there are two broad classes of algorithms – *model-based* approaches that encode known models to represent either R or π directly, and *model-free* approaches that do not assume that R is known, and also do not enforce prior model structure on the learned π.

Model-based approaches can be further classified based on which of π and R are based on pre-defined models. Some of the earliest approaches to autonomous navigation in social settings were with model-based policies, including standing in line [13] and person-following [14]. Unfortunately such approaches require explicit enumeration of all possible social scenarios, which is infeasible in real world settings. To generalize behaviors to novel scenarios, social factors have been used to model R, such as proxemics [2] for social navigation [15], or the social force model [1], [16], and learn π by optimizing for the discounted rewards. Such an optimization may be computationally intractable over continuous state and action spaces – to overcome this limitation, multi-policy decision making (MPDM) for social navigation [12] decomposes M into hierarchical policies, where at the highest level, π selects lower-level controllers (policies) as actions. MPDM is thus capable of real-time optimization of the discounted objective function by rollouts of the hierarchical policy over a receding horizon. However, MPDM still requires model-based specifications of R, which makes it hard to adapt to novel social circumstances such as taking turns through busy doors, which may not be captured by the models for R.

Model-free approaches rely on function approximators to represent π and optionally R, most recently using deep neural networks (DNNs). If R is known, deep reinforcement learning (DRL) can be used to learn a DNN-based representation of π [5], [17]. If R is not known, inverse reinforcement learning (IRL) for social navigation [18], [19] first infers R, and then given access to T in simulation or via real-world evaluation, learns π. If R is neither known, nor easy to infer π from, learning for demonstration (LfD) approaches such as Generative Adversarial Imitation Learning [4] are used to infer π directly from state-action sequences from user demonstrations. Despite their success at inferring π DNN-based approaches suffer from known limitations - they require significant data, are brittle to domain changes, and once trained are hard to adapt to new settings.

We propose a novel solution to social navigation that overcomes limitations of both model-based, and DNN-based model-free social navigation: synthesis of π as symbolic programs in a model-free LfD setting. IDIPS synthesizes symbolic policies from orders of magnitude fewer demonstrations than DNN-based approaches, and since IDIPS does not assume any prior for R or π, it is capable of adapting to completely new scenarios. IDIPS builds on recent work on SMT-based solutions to social navigation [20] and robot program repair [11], and extends them using dimension-informed program synthesis [9].

III. Symbolic Policies for Adaptable Social Navigation

We propose learning π directly from demonstrations in the form of symbolic action selection policies (ASPs) written in the language described in prior work [9]. A symbolic ASP represents π as a human-readable program where the actions are discrete subpolicies such as follow, halt, or pass. This formulation is similar to the MPDM formulation [12], where the ASP serves the role of the reward-based subpolicy selection in MPDM. For the remainder of this paper, we will refer to these sub-policies as *actions*.

The structure of symbolic ASPs is essential to learning and adapting these policies. Consider the structure of the policy shown in Fig. 1. A full *policy* π consists of logical branches that each return an *action* a, each branch consists of a *predicate* b that represents decision logic, and is in turn composed of *expressions* e that compute features from elements of S and real-valued *parameters* x_p that determine the decision boundaries.

```
if (a_i==GoAlone && |p_r − H_p[0]| > 2.0): return GoAlone

elif (a_i==GoAlone && (p_r − H_p[1]).x > 1.0 &&
      v_r.x − H_v[0].x > 0.0 && |p_r − H_p[0]| ≤ 2.0): return Pass

elif (a_i==GoAlone && (p_r − H_p[1]).x ≤ 1.0
      && |p_r − H_p[0]| ≤ 2.0 ||
      v_r.x − H_v[0] ≤ 0.0 && |p_r − H_p[0]| ≤ 1.0): return Follow
```

Fig. 1: An example symbolic action selection policy (ASP). Predicates in each branch are outlined in orange. Expressions are highlighted in purple, parameters in green, and actions in red.

The structure of the ASPs and the operator rules are described by the language, but the actions, elements of S, and the library of operators used to calculate e are particular to the application domain to which IDIPS is applied. To learn these policies from user demonstrations we require a way to demonstrate the action transitions and world states.

A. End-User Policy Demonstrations

We propose two methods for leveraging user guidance to generate demonstrations of the form $\langle a_t, s_t, a_{t+1}\rangle$. In simulation, we simulate the state transition function T while executing a policy π that continuously executes a fixed action a to generate a series of demonstrations $\{\langle a, s_t, a\}$. At any time the user may interrupt the simulation, optionally rewind to a time $t - n$, and provide a demonstration directing the robot to transition to a fixed π' that executes a'. This process adds a single transition demonstration $\langle a, s_{t-n}, a'\rangle$, and continues until the user is satisfied.

In the real world, the robot observes states s_t and continously executes a policy π with the real world providing T. At any time, the user may interrupt the robot and joystick it through a series of states $\{s_t \dots s_{t+n}\}$. The user then

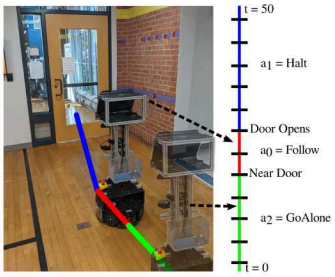

Fig. 2: An example labeled demonstration sequence and corresponding timeline showing the robot waiting for the door. The overlayed timeline is color-coded to show the robot action for that time.

labels this sequence with actions that should have been executed by identifying the states s_t where the robot should transition from an action a_t to an action a_{t+1} resulting in a demonstration sequence $\{\langle a_{t-1}, s_t, a_t\rangle : t \in (t_1, t_n)]\}$. Fig. 2 shows a visual representation of a demonstration sequence for a robot moving through a doorway.

IV. Iterative Dimension Informed Program Synthesis

Given a sequence of demonstrations, the goal is to learn a new policy or adapt an existing one to better match the demonstrations. At a high-level, this process proceeds as follows: When there is no initial ASP, we need to *synthesize* a new symbolic ASP from scratch. Given an existing policy and a new set of demonstrations, we identify the components in need of adaptation through *fault localization*. For each faulty predicate identified by fault localization, we make minimal modifications to the ASP that improve performance. Since each predicate is composed of expressions compared to real-valued parameters, we can optimize performance via *parameter optimization*. If, after parameter optimization, the policy is still faulty, then those faults must lie with the predicate itself. To maximize performance on the new demonstrations while maintaining performance on prior scenarios, *predicate repair* can be employed to add new conditional logic. An end-user can then deploy the ASP, and as adjustments become necessary, repeat this process to adapt the ASP as shown in Fig. 3.

Our approach is Iterative Dimension Informed Program Synthesis (IDIPS), shown in Fig. 4a. IDIPS synthesizes policies by combining three modules: a policy synthesizer for π (line 2), an optimizer for real-valued parameters (line 6), and a predicate repair module that extend predicate logic (line 8). IDIPS consumes an optional ASP π and a demonstration sequence $\mathbf{D}$ and produces a new ASP π' that exceeds a minimum success rate s_m on $\mathbf{D}$, if such a π' exists. As our policy synthesis module line 2, we employ Layered Dimension Informed Program Synthesis (LDIPS) [9]. Given demonstrations, LDIPS produces a symbolic policy matching the structure described in § III that is constrained by a dimension-informed type system to prevent synthesis of physically meaningless expressions. When no initial policy is provided, then IDIPS is equivalent to running LDIPS alone for the given $\mathbf{D}$. When a complete π is provided, IDIPS attempts adaptation to satisfy $\mathbf{D}$.

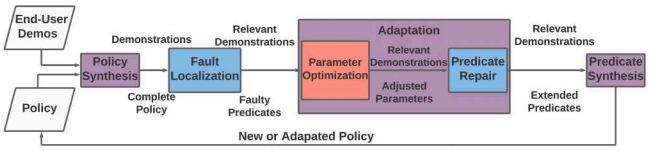

Fig. 3: System diagram of IDIPS. Unmodified components are shown in red, updated components are shown in purple, and novel components are shown in blue.

A. Fault Localization

IDIPS needs only adapt predicates with suboptimal performance with respect to $\mathbf{D}$. We identify these predicates using a *fault localization* procedure made possible by the structure of our symbolic ASPs. As shown in Fig. 4a, `FindPredicates` (line 3) simulates running π on every demonstration and identifies the predicates b in π or implied by the demonstrations, and the corresponding positive and negative examples $(\mathbf{E_p}, \mathbf{E_n})$ for which the b should return `true` and `false`, respectively. For a predicate b that determines whether π should transition from executing previous action a_1 to a new action a_2, positive and negative examples are drawn from demonstrations $\mathbf{D}$ such that

$$\mathbf{E_p} = \{\langle s_t, a_t, a_{t+1}\rangle \in \mathbf{D} : a_t = a_1, a_{t+1} = a_2\}, \quad (3)$$

$$\mathbf{E_n} = \{\langle s_t, a_t, a_{t+1}\rangle \in \mathbf{D} : a_t = a_1, a_{t+1} \neq a_2\}. \quad (4)$$

Then, for each $\langle b, \mathbf{E_p}, \mathbf{E_n}\rangle$ the helper function `Score` calculates the percentage of examples with which b is consistent, and any predicate with a score less than s_m is considered faulty (lines 6 and 8). Given a faulty b, adaptation proceeds with the optimization module first, and then the repair module if the optimized policy does not meet s_m.

B. Parameter Optimization

The optimization module adjusts real-valued parameters to match the demonstrated behavior. We employ SMT-based Robot Transition Repair (SRTR) as our optimization module [11]. SRTR is a white-box approach that leverages MaxSMT to correct parameters based on user-supplied corrections. A high-level version of the SRTR algorithm is shown in Fig. 4b. In this algorithm, the helper function `ExtractParams` (line 2) identifies all parameters in a predicate via program analysis. Given $\vec{x_p}$ (line 3) employs a canonical partial evaluator [21] to form a simplified representation of $\mathbf{E_p}$, $\mathbf{E_n}$, b, and $\vec{x_p}$ as a MaxSMT formula, that can be optimized with an off the shelf MaxSMT solver [10]. By leveraging SRTR's MaxSMT approach, we can satisfy the maximum number of new demonstrations while minimizing changes to the parameters, yielding updated predicates b' that can be incorporated into π' and are amenable to further repair.

C. Predicate Repair

Parameter optimization and synthesis are not always sufficient for adapting policies – parameter optimization cannot add new conditional branches in an ASP, and from-scratch synthesis risks degrading performance on *previously working* demonstrations. To overcome these challenges, IDIPS uses predicate repair to extend conditionals while minimizing syntactic change. Predicate repair consumes a predicate b

```
1  IDIPS(s_m, D, π):
2    π' = Synthesize(s_m, D, π)
3    problems = FindPredicates(D, π')
4    for (b, E_p, E_n) in problems:
5      if (Score(b, E_p, E_n) < s_m):
6        b' = Optimize(E_p, E_n, b)
7      if (Score(b, E_p, E_n) < s_m):
8        b' = Repair(s_m, E_p, E_n, b)
9    return π'
```

(a) IDIPS

```
1  Optimize(E_p, E_n, b):
2    x⃗_p = ExtractParams(b)
     φ=∃x⃗_p
3      | (∀w ∈ E_p . PartialEval(b, w))∧
       (∀w ∈ E_n . ¬PartialEval(b, w))
4    return Optimize(φ) #MaxSMT
```

(b) Parameter Optimization

```
1  Repair(E_p, E_n, b):
2    falseNeg = Score(b, E_p, ∅) < 1.0
3    falsePos = Score(b, E_n, ∅) < 1.0
4    if (falsePos and falseNeg):
5      b' = (b ∧ p) ∨ (p ∧ p')
6    elif (falseNeg): b' = b ∨ p
7    elif (falsePos): b' = b ∧ p
8    return PredSynth(E_p, E_n, b')
```

(c) Predicate Repair

Fig. 4: Algorithms for Iterative Dimension Informed Program Synthesis (IDIPS) and accompanying submodules.

and sets of positive and negative examples $\mathbf{E_p}$ and $\mathbf{E_n}$ for which b has been determined to be faulty, and outputs an extended predicate b' with minimal syntactic extensions and improved performance on $\mathbf{E_p}, \mathbf{E_n}$.

The predicate repair algorithm (Fig. 4c) proceeds in three steps: fault classification, predicate extension, and predicate synthesis. 1) Fault classification identifies whether b returns false positives, false negatives, or both, given the example sets $\mathbf{E_p}$ and $\mathbf{E_n}$ (lines 2–3). 2) Predicate extension (lines 5–7) builds on b with three kinds of predicate placeholders b', that either weaken, strengthen, or both. If b contains both false positives and false negatives, b is extended using both a conjunction and a disjunction to produce b' (line 5). If the only failures are false negatives, b is made less strict using a disjunction to yield b' (line 6). If the only failures are false positives, b is made stricter using a conjunction, to produce b' (line 7). 3) Predicate synthesis completes the parts of b' that were extended – p and p' (line 8) by synthesizing new structure, expressions, and parameters to replace them.

D. Predicate Synthesis

Given an extended predicate b' predicate synthesis completes placeholder p such that b' has maximal performance on $\mathbf{E_p}$ and $\mathbf{E_n}$. Predicate synthesis employs a fragment of LDIPs [9] extended to handle conflicting demonstrations using MaxSMT. This fragment of LDIPS is responsible for enumerating and completing all candidate predicates. Internally predicate synthesis requires expression synthesis to complete candidate predicates with physically meaningful expressions. Expression synthesis employs feature enumeration to iteratively synthesize all possible expressions. In turn, expression synthesis employs parameter synthesis to find real-valued threshold parameters that satisfy all demonstrations, if any such parameters exist for a candidate program.

We extend this fragment of LDIPS with MaxSMT constraints by using parameter optimization as in § IV-B. Instead of searching for parameter values that satisfy all demonstrations we solve for values that satisfy the maximum subset of the demonstrations. Then, for each of expression synthesis and predicate synthesis, we consider all possible candidates and return the best performing solution. When predicate synthesis is complete, the best performing adapted policy π' consisting of the best performing b' is returned. This process can continue iteratively, with the user deploying π' and providing new demonstrations to IDIPS as necessary.

V. Evaluation

We present results from several experiments in the social navigation domain that evaluate IDIPS's ability to 1) effectively synthesize policies that reflect user preferences, 2) adapt existing policies to different preferences, 3) repair existing policies for novel scenarios, and 4) transfer programs to real-world robots and repair for performance.

A. Implementation

We consider a mobile service robot deployed in the hallways of a busy office building moving alongside humans. For simulating humans, we employ pedsim_ros [1], a crowd simulator built on libpedsim. We use a virtual hallway with three possible starting and ending robot locations for our simulated experiments, as shown in Fig. 5.

We define the world state $w = \langle S_r, S_h, S_e \rangle$ to include the robot state S_r consisting of global and local goal locations, human states (S_h) consisting of the poses and velocities of the three closest humans, and the state of navigation relevant elements of the environment (S_e), such as the map and door state. Our ASPs consist of four actions, stopping in place (Halt), navigating to the goal (GoAlone), following a human (Follow), and passing a human (Pass).

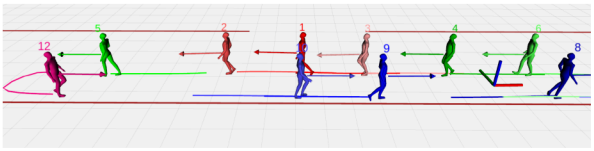

Fig. 5: Example Simulation Environment.

B. Metrics

As a model-free approach to social navigation, LDIPS does not need any reward function to be specified – it directly synthesizes ASPs from user demonstrations. However, we use three quantitative metrics to measure the difference in performance between different ASPs. To evaluate the impact on humans during navigation, we evaluate two social metrics, force and blame, as described in work on MPDM [12]. Force is a metric of distance between the robot and the closest human, and represents the influence humans and the robot exert on each other as they travel, while blame is a metric of position and velocity that approximates the robot's level of responsibility for this influence. To evaluate efficiency, we report the time in seconds to reach the goal.

C. Performance of Synthesized Social Navigation

To evaluate the performance of IDIPS synthesized ASPs, we synthesize policies from two demonstration sets (Nice, Greedy) with opposite preferences, a *Nice-I* policy that prioritizes socially passive behavior, and a *Greedy-I* behavior that prioritizes time to goal. The demonstration sets were

[1] https://github.com/srl-freiburg/pedsim_ros

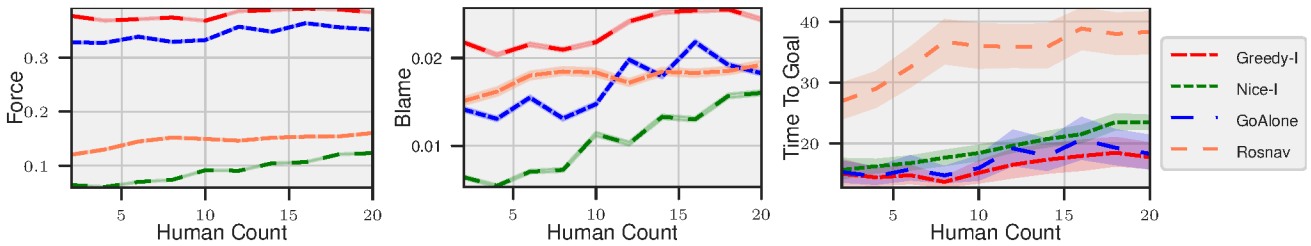

Fig. 6: Performance for synthesized policies and baselines. The shaded regions regions around lines represent the 90% confidence interval.

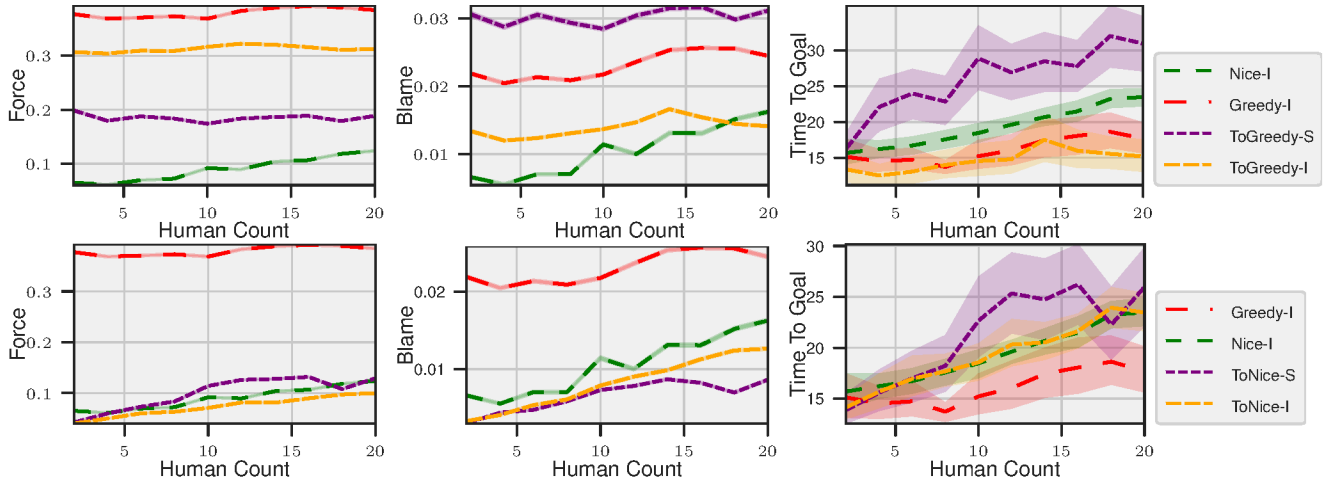

Fig. 7: Performance for adapted policies. The shaded regions regions around lines represent the 90% confidence interval.

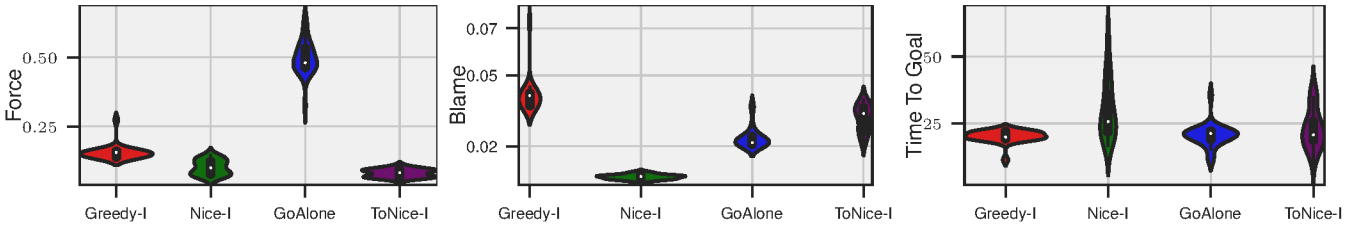

Fig. 8: Performance from real robot experiments.

human-generated in simulation as described in § III-A, based on user interpretations of "Nice" and "Greedy". We compare synthesized policies to a GoAlone policy representing the default behavior of our robot using only the GoAlone action and a solution leveraging the ROS navigation stack extended with social awareness that we refer to as Rosnav [2].

Demo Set	Satisfied (%)		Timesteps	Used for Synthesis
	Nice-I	Greedy-I		
Nice	92	76	5642	640
Greedy	81	94	2618	350

Fig. 9: Percentage of demonstration sets satisfied per policy.

We vary the number of humans and robot goal during 25 trials each for a total of 1000 trials for each policy, and present the results in Fig. 6. Greedy-I consistently reaches the target in the shortest time but also exhibits the highest force and blame. In comparison, Nice-I is slower but exhibits the lowest force and blame. While Rosnav achieves comparable performance to Nice-I in terms of Force and Blame, it comes with a significant increase in travel time.

[2]http://wiki.ros.org/social_navigation_layers

To evaluate how closely the synthesized behaviors model their respective demonstration sets, we show the number of demonstrations from each demo set satisfied by each synthesized policy in Fig. 9. As expected, this table shows that synthesized behaviors most closely model the demonstrations from which they were synthesized.

D. Adapting Social Navigation Preferences

To simulate a scenario where an end-user wants to adapt an existing policy for their preferences or domain, we adapt Nice-I with demonstrations from Greedy-I, and vice versa using two methods: parameter optimization (SRTR) [11] (*ToGreedy-S, ToNice-S*), and IDIPS (*ToGreedy-I, ToNice-I*). We repeat our experimental procedure for these policies and report the results in Fig. 7. For blame and force, the performance shifts as expected for both SRTR and IDIPS – the ToNice-* ASPs sacrifice time efficiency for social niceness, while the ToGreedy-* ASPs prioritize time efficiency over social niceness. However, the adaptation is significantly more effective using IDIPS than using only SRTR, validating our hypothesis that program repair is key to improving adaptation beyond parameter optimization – ToGreedy-S is unable to improve navigation time as significantly as ToGreedy-I.

Similarly, ToNice-S is unable to navigate as efficiently as ToNice-I, and thus has higher time to goal.

E. Real World Evaluation

To evaluate transferring IDIPS policies from simulation to the real world, we employ a Cobot [22] mobile service robot. Per COVID-19 safety guidelines, only a single human participant was used for this case study. We consider two scenarios in this environment and four policies transferred from simulation (GoAlone, Nice-I, Greedy-I, ToNice-I). For each policy, we vary the travel direction and speed of the human and the starting location of the robot for a total of 25 trials per policy. For the results shown in Fig. 8, all of the policies are able to accomplish the goal while roughly maintaining their relationship in terms of our metrics. The Nice-I behavior is still the most passive policy, while the Greedy-I behavior is more aggressive, showing that repair for domain transfer with IDIPS is effective.

F. Adapting To Unseen Scenarios

To evaluate IDIPS's adaption to novel scenarios, we performed two sets of experiments by adding a closed door in both simulation and a real hallway Fig. 2. We evaluate the Nice-I and Greedy-I policies, and those same policies adapted using demonstrations of how to wait for the door before proceeding (*NiceDoor-I, GreedyDoor-I*). We adapt the policies separately for simulated and real-world examples and record 200 trials with varying human counts in simulation and ten trials with a single human in the real-world. Fig. 10 shows the percentage of successful runs for each policy. Neither original policy waits for the door, leading to almost complete failure. In contrast, the repaired policies are able to handle the new scenarios with minimal failures in simulation and complete success in the real-world. This adaptation would require extensive modification for Rosnav and is only possible when relearning from scratch in DNN-based approaches.

Policy	**Success Rates** (%)	
	Simulation	**Real World**
Nice-I	2	0
Greedy-I	7	0
NiceDoor-I	100	100
GreedyDoor-I	84	100

Fig. 10: Success rates for different ASPs on closed door scenarios.

VI. Conclusion

In this work, we presented an approach for learning and adapting symbolic social navigation policies that builds on dimension informed program synthesis. We model social navigation as an action selection problem and learn and adapt behaviors with small numbers of human-generated demonstrations. Our experimental evaluation demonstrated that this technique can learn effective social navigation policies that model the preferences of the demonstrations and that these symbolic policies can be efficiently adapted for changing user preference and novel scenarios. Further, we demonstrated in a case study that our technique can adapt policies on real-world mobile service robots.

VII. Acknowledgments

This work is conducted in collaboration between the AMRL at UT Austin and Professor Arjun Guha at Northeastern University, and is supported in part by NSF (CAREER-2046955, IIS-1954778, SHF-2006404, CCF-2102291 and CCF-2006404), ARO (W911NF-19-2-0333), DARPA (HR001120C0031), Amazon, JP Morgan, and Northrop Grumman Mission Systems.

References

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

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

[3] 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.

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

[5] 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.

[6] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, and P. Corke, "The limits and potentials of deep learning for robotics," *The International Journal of Robotics Research*, pp. 405–420, 2018.

[7] N. Topin and M. Veloso, "Generation of policy-level explanations for reinforcement learning," in *AAAI*, 2019.

[8] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience," in *ICRA*, 2019, pp. 8973–8979.

[9] J. Holtz, A. Guha, and J. Biswas, " Robot Action Selection Learning via Layered Dimension Informed Program Synthesis ," in *CORL*, 2020.

[10] L. M. de Moura and N. Bjørner, "Z3: An efficient smt solver," in *TACAS*, 2008.

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

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

[13] Y. Nakauchi and R. Simmons, "A social robot that stands in line," *Autonomous Robots*, pp. 313–324, 2002.

[14] R. Gockley, J. Forlizzi, and R. Simmons, "Natural person-following behavior for social robots," in *HRI*, 2007, pp. 17–24.

[15] 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.

[16] 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.

[17] 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.

[18] D. Vasquez, B. Okal, and K. O. Arras, "Inverse reinforcement learning algorithms and features for robot navigation in crowds: An experimental comparison," in *IROS*, 2014, pp. 1341–1346.

[19] B. Okal and K. O. Arras, "Learning socially normative robot navigation behaviors with bayesian inverse reinforcement learning," in *ICRA*, 2016, pp. 2889–2895.

[20] T. Campos, A. Pacheck, G. Hoffman, and H. Kress-Gazit, "Smt-based control and feedback for social navigation," in *ICRA*, 2019, pp. 5005–5011.

[21] N. D. Jones, C. K. Gomard, and P. Sestoft, "Partial evaluation and automatic program generation," in *Prentice Hall international series in computer science*, 1993.

[22] M. Veloso, J. Biswas, B. Coltin, and S. Rosenthal, "Cobots: Robust symbiotic autonomous mobile service robots," in *IJCAI*, 2015, p. 4423–4429.