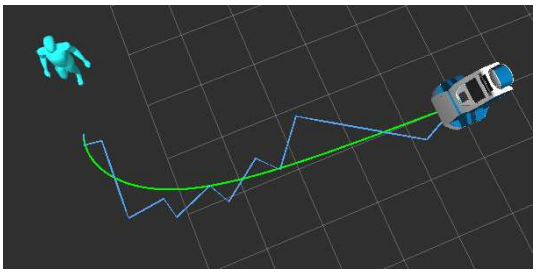
Robots Learning How and Where to Approach People

Omar A. Islas Ramírez[a] Harmish Khambhaita[b] Raja Chatila[a] Mohamed Chetouani[a]
Rachid Alami[b]

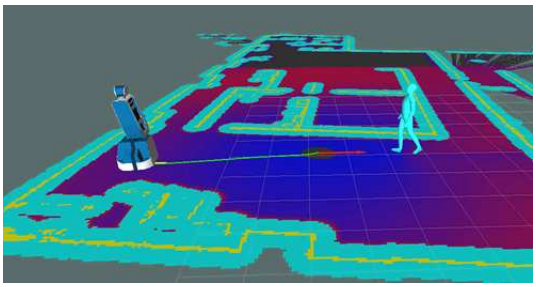
Abstract

Robot navigation in human environments has been in the eyes of researchers for the last few years. Robots operating under these circumstances have to take human awareness into consideration for safety and acceptance reasons. Nonetheless, navigation have been often treated as going towards a goal point or avoiding people, without considering the robot engaging a person or a group of people in order to interact with them. This paper presents two navigation approaches based on the use of inverse reinforcement learning (IRL) from exemplar situations. This allow us to implement two path planners that take into account social norms for navigation towards isolated people. For the first planner, we learn an appropriate way to approach a person in an open area without static obstacles, this information is used to generate robot's path plan. As for the second planner, we learn the weights of a linear combination of continuous functions that we use to generate a costmap for the approach-behavior. This costmap is then combined with others, e.g. a costmap with higher cost around obstacles, and finally a path is generated with Dijkstra's algorithm.

Keywords. Human Aware Navigation, Inverse Reinforcement Learning, Approaching People



(a)



(b)

Figure 1: **a)** Proposed path to approach the person. Violet line: MDP resolution in a deterministic or the most probable transition case. Green line: fitted curved treated with least squares and Bézier lines. **b)** Layered Costmap Navigation with IRL learned layer

1 Introduction

We approach people everyday and interact with them, and it is an intuitive situation when one gathers with their friends or family. In this intuitive behavior, we know that certain motions or situations are not socially acceptable and we try to avoid them. What do we do exactly? This is a simple question, but when we refer to a robot, we have to model and formalize its behavior, and implement it from path planner to entire navigation process.

In this paper, two navigation strategies to approach a human were implemented using low level information about human's position and orientation. The first one is a path planner that takes into account only a relative human polar frame as in Figure 1(a) and the second one is a costmap layer [9] based on the same variables that can take into account obstacles shown in Figure 1(b). The main difference compared to other works in human aware navigation [16, 4, 17] is that instead of a human operator giving a goal, it is our algorithm that provides the goal to reach and the appropriate path.

This work is partially performed within the SPENCER project (Social situation-aware perception and action for cognitive robots) which aims to deploy a fully autonomous mobile robot to assist passengers at the Amsterdam Schiphol Airport. In the approach developed in this paper, the robot learns a policy from exemplary trajectories of humans approaching people to interact with them. Such trajectories define social norms as a reference for robot behavior.

The rest of the paper is organized as follows. Section 2 refers to related works. Given that our scenario is based on learning from demonstrations using IRL techniques, we define our model in Section 3. This model is applied to demonstrations given by an expert, in our case these demonstrations are paths generated by a robot controlled by a person. These demonstrations are the input of the IRL algorithm. Learned policy from IRL output is used to generate a path-plans in Section 4. Lastly, Section 5 provides experimental results before a conclusion.

[a]ISIR-CNRS, Université Pierre et Marie Curie, Paris, France {islas,chatila,chetouani}@isir.upmc.fr

[b]LAAS-CNRS: Laboratory for Analysis and Architecture of Systems, Toulouse, France {harmish, rachid.alami}@laas.fr

2 Related Works

In robot navigation, path planners usually minimize time or distance. However, this is often not the case for social path planning, because we need to respect the private and social spaces of a person or group of people. This topic is handled by Human Aware Navigation [6]. Some authors [5, 16] have taken into account proxemics as costs and constraints in the path planner to obtain acceptable paths with hard-coded proxemics values derived from sociology. However, these values are not necessarily true in all situations, as they could depend on the velocities of the people, as commented in [10].

Other works that deal with the subject of approaching humans [14, 3], focus on tackling the problem of a task planner, considering *pedestrians' intentions* such as people interested in approaching the robot. As for the navigation part, they look for the robot's path intersecting a person while he/she moves. Shomin's work [15] considers approaching people in order to interact in collaborative tasks, nonetheless they used hard-coded waypoints in order to navigate. In our work, we focus in the way the robot shall move in order to reach an engagement given previously generated demonstrations. A main difference with these related works is that we find the final position given the demonstrations instead of hardcoding it.

Inverse Reinforcement Learning method enables a robot to learn a policy using discrete and finite MDP (Markov Decision Process) in which the states are derived from the robot's relative position and orientation with respect to the human. Lately, IRL has been shown to teach machines to act as humans do. For example, in the Human Aware Robotics domain, recent works address robot navigation in crowds [2, 17] and other social spaces [4]. These examples tackle navigation from point A to point B while avoiding people, not for approaching them. The closest work to ours is [13] where they develop a method based on IRL for enabling a robot to move among people and in their vicinity (4mx4m) in a human-like manner. We specifically address the problem of *how to approach people* to interact with them, therefore our algorithm is able to provide a proper goal to be reached by the robot and a social path to reach this goal. This requires a specific model representing the space around the humans and appropriate trajectories for homing on them.

3 Modeling Steps

In this section, we first recall the inverse reinforcement learning problem based on the MDP. We then introduce the components of the MDP which composes our modeling.

3.1 MDP and IRL

A finite Markov Decision Process is classically defined by the following five elements:

- A finite set of states S.
- A finite set of actions A.
- A transition probability function $P(s_t, a_{t-1}, s_{t-1})$, which is the probability to reach state s_t by achieving action a_{t-1} in state s_{t-1}. The transition matrix $T(S, A, S)$ is composed of all such probabilities $P(s_t|a_{t-1}, s_{t-1})$ and its size is $S \times A \times S$.
- A reward function $R(s_t, a_{t-1}) \in \mathbb{R}$ that depends on the state-action pair.
- A discount factor $\gamma \in [0, 1)$ which reflects the importance of short-term vs. long-term rewards.

Solving the MDP consists of finding an optimal policy, which provides an action for every state that should be selected in order to maximize the total utility.

Reinforcement Learning (RL) is a part of machine learning in which the learner is not told which actions to take, as in most forms of machine learning, instead it must discover which actions yield the most reward by trying them out. Inverse Reinforcement Learning (IRL) on the other hand, deals with the problem of finding the reward from either an existent policy or from a demonstrated sequence (as in our case).

We assume that the expert from which we want to learn can be modeled by an MDP. Our problem is defined by the tuple $\langle S, A, T, R, D, \gamma \rangle$, which is an MDP plus the added D variable which represents demonstrations given by an expert.

Nowadays, we have a collection of IRL algorithms [11, 8, 18, 12, 1].

Since we want to find a reward function based on the state-action pairs, we can represent a state-action pair as a vector of features $\Phi(s, a) = [f_1(s, a), f_2(s, a), \ldots, f_n(s, a)]$, where f_i is the ith function of the state-action pair. Thus, we can represent our reward function as a linear combination of these features $R(s, a) = w^T \Phi(s, a)$. Where w is the vector of weights.

In general, learning the reward function is accomplished as follows. At the very first time a random reward is created, for this case, a random weighted vector w. At each step i of demonstration k the reward obtained will be denoted $R(s_i^k, a_i^k)$. Depending on the IRL algorithm, an optimal policy $\pi^*(s)$ is found by maximizing the probability of the reward given the demonstrations as a posterior probability of the likelihood of the demonstrations given the reward and a prior function of the reward $P(R|D) \propto P(D|R)P(R)$ or by maximizing the expected sum of rewards given the demonstrations $E[\sum_{t=0}^{N} \gamma^t R(s, a)]$.

3.2 State

For the sake of clarity, we introduce the state representation considering one person only. The robot state will be the human-centered polar representation of the robot with respect to the person. This representation is depicted in Figure 2.

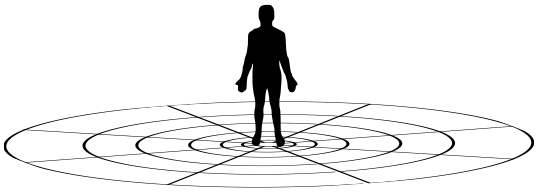


Figure 2: Human centered State

Two components are needed, distance d and angle θ from the reference point. The distance component d is discretized along a quadratic based function (Figure 3). This function allows to easily change the state space to create various tests and to have more precision in the region near the person.

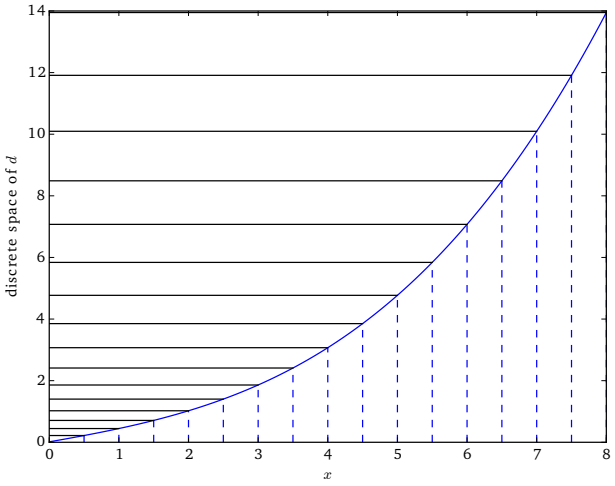


Figure 3: Discretization of distance state given the quadratic based function. Values in y axis are the ones used for Φ_d.

For the state angle component θ, we divided the region into m sections. Thus, the range between each state is a region $2\pi/m$ of the environment.

Both parameters (distance and angle) define the state. The state representation is, then, in $\mathbb{R}^{n\times m}$, and we have a total number of states of $S = n \cdot m$. For MDP purposes, the conversion of this 2-dimensional matrix $\mathbb{R}^{n\times m}$ needs to be transformed in a vector which is going to represent the state. For this work, the matrix was simply reduced into one dimension $f : \mathbb{R}^{n\times m} \to \mathbb{R}^S$ by concatenating the rows.

3.3 Actions and Transitions

Given the state representation, we define a set of 5 actions described below.

1. (θ_c, d_c) : staying in the same place.
2. $(\theta_c + 1, d_c)$: moving forward in θ_c.
3. $(\theta_c - 1, d_c)$: moving backward in θ_c.
4. $(\theta_c, d_c + 1)$: moving forward in d_c.
5. $(\theta_c, d_c - 1)$: moving backward in d_c.

Where θ_c represents the current angular state and d_c the current distance from the person. An example of state transition probability is shown in Figure 4, where we represent our polar states as an unfolded map.

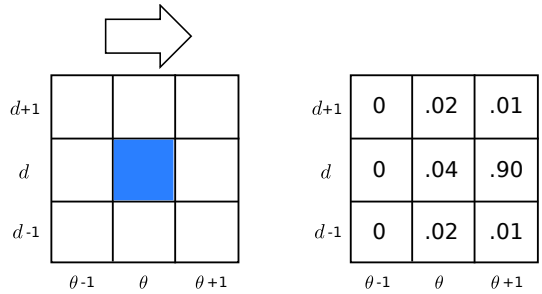


Figure 4: Illustration of action "go in direction $\theta + 1$" and its transition probabilities to adjacent places.

The transition matrix is the agglomeration of the 5 actions for all the states. The probabilities reflect the actual reachability of the robot. Figure 4 is representing only the adjacent states, we have to imagine that the figure expands in all the environment, with transition probability of zero for all the other values not shown in the figure, and thus having a sparse matrix of size $S \times A \times S$.

3.4 Feature Representation

Two methods are developed to tackle the approaching behavior. *Naive Global Planner*, in which a path planner is created directly based on the response of the IRL algorithm and *Layered Costmap Navigation*, in which a state of the art path planner used based on [9]. In the first one, the number of features is equal to the number of states multiplied by the number of possible actions. In the second one Radial Basis Functions (RBF) are used to represent the state. Each one of these approaches is going to work differently for the implementation.

Naive Global Planner

In order to build the state-action vector, first we create a base feature vector based on our number of states S, as follows $\Phi(s) = [\phi_1(s), \phi_2(s), \ldots, \phi_S(s)]$. In which $\phi_i(s)$ is a Kronecker delta function where $\phi_i(s) = [i = s]$ using Iverson bracket notation. In order represent $\Phi(s, a)$, the technique used in [7] is applied, creating a feature vector with size of the features $\Phi(s)$, multiplied by the number of actions. Let's say the action a is equal to 2, given the possible 5 actions, then $\Phi(s, a) = [\mathbf{0}, \Phi(s), \mathbf{0}, \mathbf{0}, \mathbf{0}]$. Where $\mathbf{0}$ is a zero vector with the size of $\Phi(s)$.

Layered Costmap Navigation

The main difference with the previous case is the use of continuous state features. Our intention is to build a costmap for the approach scenario in which the robot navigates.

Since the states that are taken into account correspond to the polar human representation, we set n number of random points in the environment within a range for each axis

of $r_d = [0, 14]$ and $r_\theta = [-\pi, \pi)$, where r represents range and is given in *meters* and *radians* respectively. An example of these random points can be seen as the crosses in Figure 8 and they represent the mean in the 2D gaussian used for the RBF. As for the value of the standard deviation, all RBF bins have the same value which is a quarter of the range for each axis. Thus, the vector state representation is $\Phi(s) = [\phi_1(s_{\text{coord}}), \phi_2(s_{\text{coord}}), \ldots, \phi_n(s_{\text{coord}})]$, where $\phi_i(s_{\text{coord}})$ is the ith RBF and s_{coord} is the cartesian center of the state s. Then we set $\Phi(s, a) = \Phi(s)$ given than it is intended to use this information in a costmap, which is only represented by the states and not the actions, differently from *Naive Global Planner*.

4 Adapting IRL Results

Input for an IRL is the demonstrations given by an expert, in our case, the demonstrations are the paths the expert chose to go to a person. These paths are sampled in state-action pairs which are converted to features described in Subsection 3.4. The output and the post-process applied this output is described in next subsections for each planner.

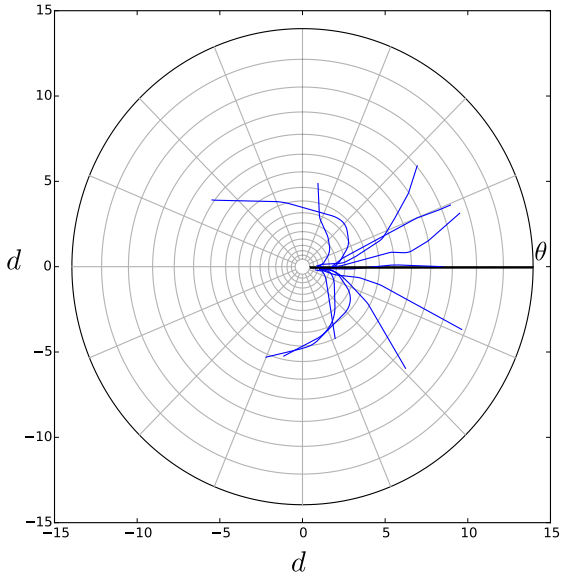


Figure 5: Demonstration of the robot approaching the target person. Values of $N = 16$ and $M = 16$ for the state space. The darker black line represents the front of the person.

4.1 Naive Global Planner

The result of this IRL provides the rewards to the MDP, and by applying the optimal navigation policy in this MDP, the robot moves along the sequence of states which forms the optimal trajectory to approach a person. Each state (i.e. the cell in the representation described in the previous section) is represented by its center. As a result the trajectory is a discontinuous line as shown in green in Figure 6. We hence need to smooth this trajectory taking into account the robot orientation and human orientation. Smoothing process is described next and the result is also shown in Figure 6. These trajectories are the global plan, nonetheless they do not take into account other constraints such as obstacle avoidance.

Data Fitting:

The trajectory points are first transformed in the global frame containing the grid map. Then the points are re-transformed with a parametric function t such that for the first (x, y) coordinate $t = 1$, for the second $t = 2$ and so on. After that the points (t, x) and (t, y) are separated as two sets of data.Next, each set of data is processed with a least squares function approximation shown as the green dotted line in Figure 6.

Bézier:

A smooth curve can be generated from the two fitted functions. However, the orientation of the robot is not taken into consideration. Bézier curves can smooth the trajectory to respect robot orientation. We still have the parametric function, but since Bézier uses Bernstein Polynomial, it is inherently parametric. We use our previously presented functions with a set of few points as control points for Bézier. Another control point is added projecting the orientation of the robot, thus the path starts in the direction of the robot's orientation. This procedure is shown in Figure 6.

4.2 Layered Costmap Navigation

After the learning process the w vector is set. One important aspect is that $\Phi(s, a) = \Phi(s)$ and s is represented by spatial features. Thus, a costmap can be generated in the environment. The cost of some area around the person is calculated given a normalization of $R(s) = w^T \Phi(s)$ for all the coordinates in the map. Thus, s must be translated to the polar coordinates of the human frame. Then, based on [9], the cost is passed to the upper layer for every field if the value is higher than the one already set in the upper layer. Then Dijkstra's algorithm implementation is used to calculate the best path, being the end position of the planner the position in which the maximum value of reward is found in the costmap and the orientation opposite to the human.

Figure 8 shows a costmap result the weighted values of $R(s)$, result of the application of IRL with the demonstrations given in Figure 5, this is feasible due to the representation of features as continuous functions. Even when we have discrete states, the values of the coordinate system is in $\mathbb{R}$ for distance and angle.

5 Experimental Results

5.1 Experimental Setup

We employed ROS (Robot Operating System) to simulate the human movements, allowing us to control both robot

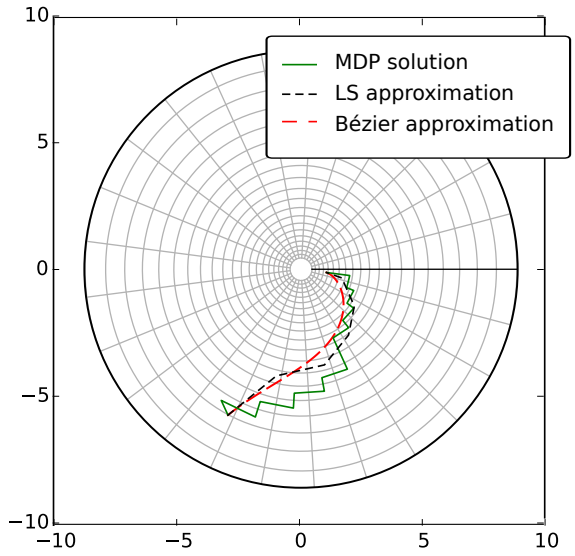


Figure 6: IRL post-processing. The green line represent the result of the MDP. The black line represents the least square approximation as a parametric function in x and y. The red line is a Bézier curve created from the set of points of this parametric function and the initial orientation of the robot.

and human behaviors (positions and velocities). We employed it to generate trajectories of robot while approaching humans. The robot is manually controlled during different approaching scenarios. A set of eleven demonstrations was performed with this experimental platform for the learning process. The path taken by the robot in different positions with different orientations can be seen in Figure 5. This represents the path followed by the robot in the human reference frame. Considering people's comfort, the robot approached the people in order to finish its behavior in the near-peripheral vision of the person. Nonetheless, if the exemplary demonstrations were performed by a human in a human environment, this behavior could differ from ours and thus the learning output.

5.2 Metrics

We propose two metrics for the evaluation of the *Naive Global planner* model.

We use a test-set of paths generated with our experimental platform but not used as inputs for training the IRL algorithm. This test-set consists of 30 recorded paths.

The first metric, called the Trajectory Difference Metric (TDM), is a modified version on the Mean Square Error (MSE). TDM evaluates every point of one trajectory to the closest point of another trajectory, where evaluating the closest point is the difference regarding MSE. This metric compares the parametric function generated by our algorithm with the trajectories of the test-set. In order to do so, all trajectories from the test-set and those provided by the algorithm are equally sampled. If P is the set of all points in the test-set trajectory and G_i is one point of the generated trajectory, $\overline{G_iP}$ represents all the distances between the point G_i and the set of points P.

The TDM is then calculated as the average value of the minimal values of these distances:

$$\mathrm{TDM} = \frac{1}{n}\sum_{i}^{n} \min \overline{G_iP} \tag{1}$$

The second metric is trajectory length, expressed as the ratio of the absolute value of the difference between the generated trajectory length and the test-set trajectory length to the test-set trajectory length:

$$\mathrm{l_{error}} = \frac{|l_m - l_{irl}|}{l_m} \tag{2}$$

For (2) we can have the case when the IRL path is longer than the test-set path or the other way around. This is simply due that human behavior is not necessarily always the same.

5.3 Results

The results shown here are from the *Naive Global planner*, given that we can measure and compare with a test set directly.

The IRL result gives an optimal policy based on the examples given. Figure 7 represents the state values, red color being the higher rewards and the blue the lower rewards, the figure corresponds to the 25x25 scenario, nonetheless the values for the 16x16 scenario have a similar pattern. The numbers on the axes represent the discretized values of distance between 0.5 and $14m$ and of orientation between 0 and 2π (hence the top and the bottom of the figure represents orientation 0 or in front of the person). The concentration of the maximum values lies around zero degrees at the second discrete position d.

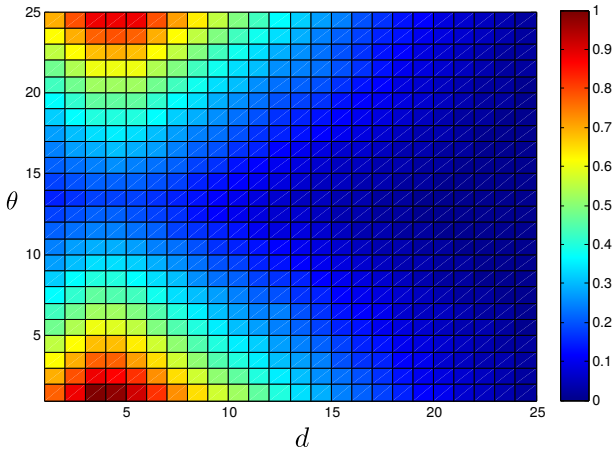


Figure 7: *Naive Global Planner* learned state environment, red value is the maximum V Value for each state. The image displayed correspond to the 25x25 grid.

As shown in the figure, the learning process has produced an optimal region near and facing the person to guide robot

navigation. The discrete representation in Figure 7 is a matrix of rewards used to generate the optimal curves to approach a human.

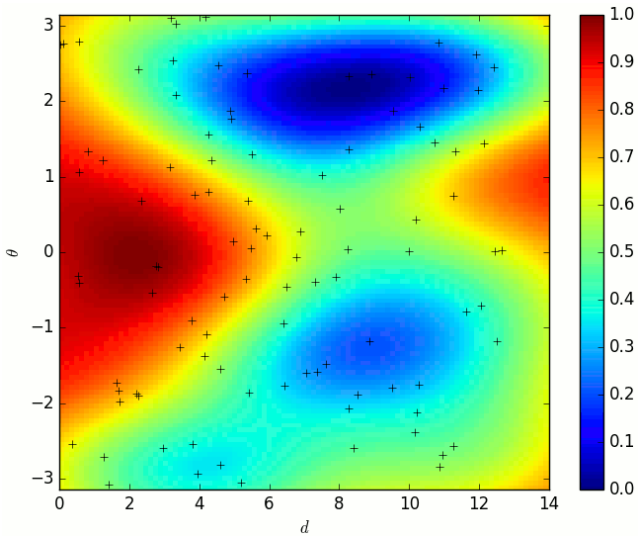


Figure 8: *Layered Costmap Navigation*: Costmap generated with $w^T\Phi(s)$ in an unfolded polar map. The blue + signs represent the center of all the RBF used in this task.

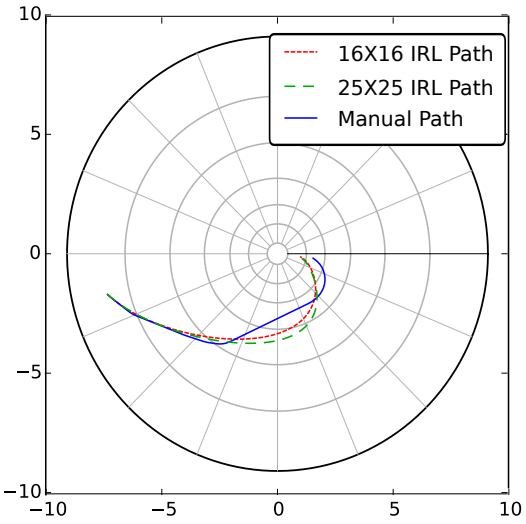


Figure 9: IRL comparison at the same starting point with a manual path sample.

Table 1 shows the average of the two metrics (1) and (2) with their respective standard deviations through the 30 test-set sample trajectories. We applied the IRL algorithm to the polar space divided in 16x16 and 25x25 discrete values respectively, and we can see that the 25x25 divisions performs slightly better that 16x16 divisions, for both the TDM and the l_{error} metrics.

Given the disparity in human motions, we can consider that the average mean error around half a meter between the test-set and the IRL trajectories is acceptable.

Figure 10 represents the error described in (1) for all the samples in the 25x25 case. The x axis represents the initial

Table 1: TDM and l_{error} evaluation of generated paths

	16x16 vs real	25x25 vs Real
TDM (meters)	0.5423 ± 0.3089	0.5322 ± 0.2995
l_{error} (%)	5.5612 ± 4.5321	5.4040 ± 4.2936

angular position of the robot given the orientation of the person. The starting angular position could go from $-\pi$ to π. We performed this analysis in order to see the behavior in different orientations of the robot. We can see that the error is low in our samples in the region near zero, nonetheless in regions near $\pm\pi$, we can also find small errors.

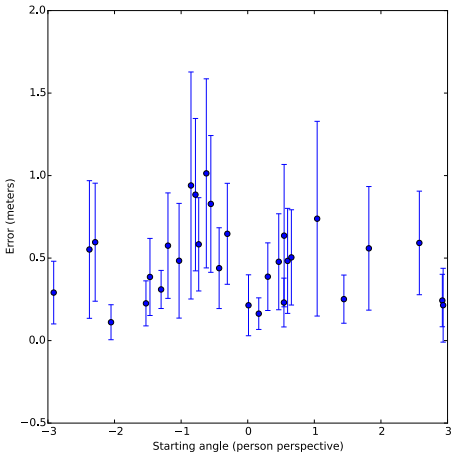


Figure 10: Evaluation of errors in (1) for all samples for 25x25 case. Angles are in rd

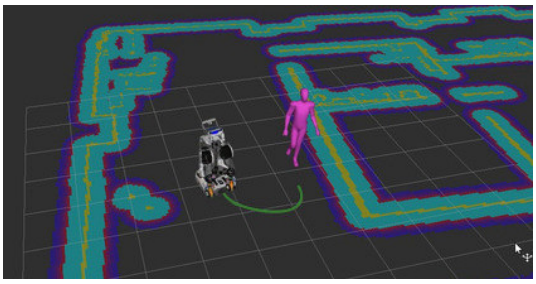
When we compare with actual human motions, we need to take into account that the human behavior is not completely smooth (e.g., the blue line of Figure 9). However, the paths generated by our method appear quite acceptable.

As discussion we could substitute Bézier with B-Splines. The main difference between both of them is that Bézier will start the path with the same orientation as the robot, while B-Splines will not start at that same orientation and this difference of orientations can be corrected with a local planner which can be convenient for high frequency updates.

With the *Layered Costmap Navigation*, the robot goes to a pertinent position (seen by the human eye) and it takes into account the obstacles, nonetheless, in order to have a good navigation we will probably need to add another layer as in [5] to give a higher cost to the center of the person. This method though, takes more computational time than the *Naive Global Planner*, because we need to compute all the cost inside the costmap area.

As an early stage test, we implemented the algorithm in a close space with PR2 robot. The detection and tracking of the person is achieved by an OptiTrack System. Figure 11(a) shows a person wearing a helmet that we use for tracking. The visual representation of the robot, hu-

(a)



(b)

Figure 11: Early stage real scenario **a)** Person wearing a helmet that is detectable by OptiTrack to get his position and orientation. **b)** Visualization of the computed path based on the Naive Global Planner (green line).

man and the proposed path generated by the *Naive Global Planner* is shown in Figure 11(b).

Videos of the results can be seen in: `http://chronos.isir.upmc.fr/~islas/approach/`

6 Conclusions

In this work we developed two path planning algorithms to approach a person. First of them uses an IRL algorithm to directly learn the social approach-paths which require smoothing. In this algorithm we recalculate the paths as the person moves. Also, an important feature is that in our global planner also selects the goal, being the final position to go and the solution of the MDP itself. Thus, both methods provide the goal that must be reached.

For the first case, we are also able to reach almost the same performance with our two discretized state cases, 16x16 and 25x25, while the first one needs much less computational time for finding a solution.

Concerning the second planner, we added a layer based of the IRL result of RBFs function to the state of the art Layered Costmap Navigation. We could still add another layer such as in [16], to avoid going near to the person.

We are aware of some drawbacks regarding both approaches. In which this framework only works for approaching one person and not taking into account the speed of the robot, since we let the local planner to take care of that. Nonetheless adding one feature (speed) and using a wrapper between global and local planner could be an interesting future work.

This work is a first step towards IRL based Human Aware Navigation for approaching. In the future, as stated in the introduction, we aim to create a general framework for approaching people by exemplary data taking into account more people and more parameters. We also plan to have user studies to measure the level of comfort and how natural the behavior of the robot is while approaching people. We also aim to implement rewards functions that can be used in navigation planner such as RRT* instead of a costmap, this can improve the speed of calculations and lead to better answers. Lastly, we would like to verify convergence with the number of exemplary demonstrations needed by different IRL algorithms.

Acknowledgements

This research has been supported by the European Commission under contract number FP7-ICT-600877 (SPENCER) and by Laboratory of Excellence SMART (ANR-11-LABX-65) supported by French State funds managed by the ANR within the Investissements d'Avenir programme (ANR-11-IDEX-0004-02).

References

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[2] P. Henry, C. Vollmer, B. Ferris, and D. Fox. Learning to navigate through crowded environments. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 981–986, May 2010.

[3] Yusuke Kato, Takayuki Kanda, and Hiroshi Ishiguro. May I Help You?: Design of Human-like Polite Approaching Behavior. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, HRI '15, pages 35–42, New York, NY, USA, 2015. ACM.

[4] Beomjoon Kim and Joelle Pineau. Socially adaptive path planning in human environments using inverse reinforcement learning. *International Journal of Social Robotics*, pages 1–16, 2015.

[5] Thibault Kruse, Alexandra Kirsch, Harmish Khambhaita, and Rachid Alami. Evaluating directional cost models in navigation. In *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction*, pages 350–357. ACM, 2014.

[6] Thibault Kruse, Amit Kumar Pandey, Rachid Alami, and Alexandra Kirsch. Human-aware robot navigation: A survey. *Robotics and Autonomous Systems*, 61(12):1726–1743, 2013.

[7] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.

[8] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear Inverse Reinforcement Learning with Gaussian Processes. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 19–27. Curran Associates, Inc., 2011.

[9] D.V. Lu, D. Hershberger, and W.D. Smart. Layered costmaps for context-sensitive navigation. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014)*, pages 709–715, September 2014.

[10] M. Luber, L. Spinello, J. Silva, and K.O. Arras. Socially-aware robot navigation: A learning approach. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 902–907, October 2012.

[11] Bernard Michini and Jonathan P. How. Improving the efficiency of Bayesian inverse reinforcement learning. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3651–3656. IEEE, 2012.

[12] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. *Urbana*, 51:61801, 2007.

[13] Rafael Ramon-Vigo, Noe Perez-Higueras, Fernando Caballero, and Luis Merino. Transferring human navigation behaviors into a robot local planner. In *Robot and Human Interactive Communication, 2014 RO-MAN: The 23rd IEEE International Symposium on*, pages 774–779. IEEE, 2014.

[14] S. Satake, T. Kanda, D.F. Glas, M. Imai, H. Ishiguro, and N. Hagita. How to approach humans?-strategies for social robots to initiate interaction. In *2009 4th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 109–116, March 2009.

[15] M. Shomin, B. Vaidya, R. Hollis, and J. Forlizzi. Human-approaching trajectories for a person-sized balancing robot. In *2014 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*, pages 20–25, September 2014.

[16] E.A. Sisbot, L.F. Marin-Urias, R. Alami, and T. Simeon. A Human Aware Mobile Robot Motion Planner. *IEEE Transactions on Robotics*, 23(5):874–883, October 2007.

[17] Dizan Vasquez, Billy Okal, and Kai O. Arras. Inverse Reinforcement Learning algorithms and features for robot navigation in crowds: An experimental comparison. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 1341–1346. IEEE, 2014.

[18] Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum Entropy Inverse Reinforcement Learning. In *AAAI*, pages 1433–1438, 2008.