Learning robot navigation behaviors by demonstration using a RRT* planner*

Noé Pérez-Higueras[1], Fernando Caballero[2] and Luis Merino[1]

[1] School of Engineering, Universidad Pablo de Olavide, Crta. Utrera km 1, Seville, Spain

[2] Dpt. of System Engineering and Automation, University of Seville, Camino de los Descubrimientos, s/n, Seville, Spain

Abstract. This paper presents an approach for learning robot navigation behaviors from demonstration using Optimal Rapidly-exploring Random Trees (RRT*) as main planner. A new learning algorithm combining both Inverse Reinforcement Learning (IRL) and RRT* is developed in order to learn the RRT*'s cost function from demonstrations. This cost function can be used later in a regular RRT* for robot planning including the learned behaviors in different scenarios. Simulations show how the method is able to recover the behavior from the demonstrations.

1 Introduction

Today, more and more mobile robots are coexisting with us in our daily lives. As a result, the creation of motion plans for robots that share space with humans in dynamic environments is a subject of intense investigation in robotics. Robots must respect human social conventions, guarantee the comfort of surrounding persons, and maintain legibility, so humans can understand the robot's intentions [10]. This is called human-aware navigation.

This problem was initially tackled by including costs and constraints related to human-awareness into motion planners to obtain socially acceptable paths [18, 7]. In these cases, these costs are pre-programmed. However, hard-coded social behaviors might be inappropriate [3]. In many cases (for instance [8, 15]), these costs are grounded in Proxemics theory [5]. However, as shown in [13], Proxemics is focused on people interaction, and it could not be suitable for navigating among people.

Therefore, learning these social behaviors from data seems a more principled approach. Also, it is easier to demonstrate socially acceptable behaviors than mathematically defining them. In particular, we consider in this paper the application of telepresence robots [17]. Our goal is to increase the autonomy of such robots, freeing the users from the low level navigation tasks. In this setup,

* This work is partially supported by the EC-FP7 under grant agreement no. 611153 (TERESA) and the project PAIS-MultiRobot funded by the Junta de Andalucía (TIC-7390)

it is very natural to obtain navigation data from the user, considering them as examples and using them to learn social navigation behaviors.

Thus, we aim to develop an approach to learn navigation behaviors from user data. The paper makes use of Inverse Reinforcement Learning (IRL) concepts and sampling-based planners (in particular, RRT* [6]) to identify the RRT cost function that better fit the example trajectories. Differently to classic IRL approaches based on Markov Decision Process (MDPs), the presented method is computationally faster and scales very well with the state size, being able to deal with continuous state and control spaces, and it is general enough to be applied in different scenarios.

The paper is structured as follows. After a summary of related work, next section describes the algorithm for learning from demonstrations using RRT*. Then, Section 3 describes the particular problem of social navigation considered in the paper. Later on, Section 4 validates the approximation in simulation. Finally, Section 5 summarizes the paper contribution and outlooks future work.

1.1 Related work

In the last years, several contributions have been presented regarding the application of learning the task of human-aware navigation. Supervised learning is used in [19] to learn appropriate human motion prediction models that take into account human-robot interaction when navigating in crowded scenarios. In [4], the parameters of a model based on social forces are learnt from feedback provided by users.

An additional approach is learning from demonstrations [2]: an expert indicates the robot how it should navigate among humans. This approach is particularly relevant for the case of telepresence robots. One way to learn from demonstrations is through Inverse Reinforcement Learning (IRL) [1]. The observations of an expert demonstrating the task are used to recover the reward (or cost) function the demonstrator was attempting to maximize (minimize). Then, the reward can be used to obtain a corresponding robot policy.

Different aspects to tackle the IRL problem have been proposed. A probabilistic method based on the principle of maximum entropy is presented in [20]. The computational cost problem is managed in [14] by using a Bayesian nonparametric mixture model to divide the observations and obtain a group of simpler reward functions. From another point of view, the authors in [12] represent the reward by using Gaussian processes instead of a linear combination of features.

In the above mentioned models, the IRL technique makes use of discrete Markov Decision Processes (MDPs) as the underlying process. However, it is complex to encode general problems with MDPs due to its computational complexity. Many authors turn to state discretization which can be tricky in many cases.

Optimal Rapidly-exploring Random Trees (RRT*) [6] are extensively employed in robot planning. They are flexible and easily adapted to different scenarios and problems. They implicitly reason about collisions with obstacles at moderate computational cost even in high dimensionality. They can explore the

state space to obtain optimal paths on cost spaces and the kinodynamic extension allows reasoning about the robot dynamics.

In the paper, we present an algorithm for learning robot navigation behaviors from demonstrations using RRT* as main planner. We aim at creating a new learning algorithm combining both IRL and RRT* techniques in order to extract the proper weights of the cost function from demonstration trajectories. This cost function can be used later in a regular RRT* to allow the robot reproducing the desired behavior at different scenarios.

2 Learning a RRT* cost function

RRT* [6] is a technique for optimal motion planning. It considers that a cost function is associated to each point x in the configuration space. The RRT* seeks to obtain the trajectory ζ^* that minimizes the total cost along the path $c(\zeta)$. It does so by randomly sampling the configuration space and creating a tree towards the goal. The paths are then represented by a set of discrete configuration points $\zeta = \{x_1, x_2, \cdots, x_N\}$.

Without loss of generality, we can assume that the cost function for each point can be expressed as a linear combination of a set of sub-cost functions, that will be called features $c(x) = \sum_j \omega_j f_j(x) = \omega^T f(x)$. The cost of a path is then the sum of the cost for all points in the path. Particularly, in the RRT*, the cost is the sum of the sub-costs of moving between pairs of points in the path:

$$c(\zeta) = \sum_{i=1}^{N-1} c(x_i, x_{i+1}) = \sum_{i=1}^{N-1} \frac{c(x_i) + c(x_{i+1})}{2} \|x_{i+1} - x_i\| \tag{1}$$

$$= \omega^T \sum_{i=1}^{N-1} \frac{f(x_i) + f(x_{i+1})}{2} \|x_{i+1} - x_i\| = \omega^T f(\zeta) \tag{2}$$

Thus, for given weights ω, the algorithm will return trajectories that try to minimize this cost.

Given a set of demonstration trajectories $\mathcal{D} = \{\zeta_1, \zeta_2, \cdots, \zeta_D\}$, the problem of learning from demonstrations, in this setup, means to determine the weights ω that lead our planner to behave similarly to these demonstrations. According to [1, 11], this similarity is achieved when the expected value of the features for the trajectories generated by the planner is the same as the expected value of the features for the given demonstrated trajectories:

$$\mathbb{E}(f(\zeta)) = \frac{1}{D} \sum_{i=1}^{D} f(\zeta_i) \tag{3}$$

One approach to solve this problem is to model the underlying trajectory distribution of the expert and consider the demonstrations as samples from this distribution. As noted in [9], applying the Maximum Entropy Principle [20] to

the IRL problem leads to the following form for the probability density for the trajectories returned by the demonstrator:

$$p(\zeta|\omega) = \frac{1}{Z(\omega)} e^{-\omega^T f(\zeta)} \tag{4}$$

where $Z(\omega)$ is a normalization function that does not depend on ζ. One way to determine ω is maximizing the (log-)likelihood of the demonstrated trajectories under the previous model:

$$\mathcal{L}(\mathcal{D}|\omega) = -D \log(Z(\omega)) + \sum_{i=1}^{D} (-\omega^T f(\zeta_i)) \tag{5}$$

The gradient of the previous log-likelihood with respect to ω is given by:

$$\nabla \mathcal{L} = \frac{\partial \mathcal{L}(\mathcal{D}|\omega)}{\partial \omega} = \mathbb{E}(f(\zeta)) - \frac{1}{D} \sum_{i=1}^{D} f(\zeta_i) \tag{6}$$

Setting this gradient to zero one arrives to (3). As mentioned in [11], this gradient can be intuitively explained. If the value of one of the features for the trajectories returned by the planner are higher from the value in the demonstrated trajectories, the corresponding weight should be increased to increase the cost of those trajectories.

The main problem with the computation of the previous gradient is that it requires to compute the expected value of the features $\mathbb{E}(f(\zeta))$ for the generative distribution (4). In [9], a probabilistic generative model for trajectories is derived from data, and the expectation is computed by Monte Carlo Chain sampling methods, which is very computationally demanding.

In our case, we will approximate the expert by the RRT* planner on board the robot. Being an asymptotically optimal planner, for some given weights, the RRT* will provide the trajectory that minimizes this cost given infinite time. As the planning time is limited, the RRT* will provide trajectories with some variability on the features, and thus the expected feature count is computed by running several times the planner between the start and goal configurations. This is then used to compute the gradient and adapt the weights used in the RRT* planner.

As mentioned, this is an approximation to the expert model. A similar idea is used in [11]. The experimental results will show that the method is able to recover the taught behaviors by the expert.

The method proposed is described in detail in Algorithm 1.1. The example trajectories from which we want to learn are used as input. The output of the method are the weights for the cost function of the RRT* algorithm in (1).

First, in Line 1 we obtain the average feature count $\overline{f}_{\mathcal{D}} = \frac{1}{D} \sum_{i=1}^{D} f(\zeta_i)$ from the example trajectories in $\mathcal{D}$ scenarios. The feature counts are obtained as the addition of the feature values of pairs of nodes of the trajectory evaluated similarly to Equation (1).

Algorithm 1.1: RRT*-IRL

Require: Trajectory examples $\mathcal{D} = \{\zeta_1^1, \ldots, \zeta_D^S\}$ in $\mathcal{S}$ scenarios
Ensure: Function features weights $\omega = [\omega_1, \ldots, \omega_J]^T$
1: $\overline{f}_{\mathcal{D}} \leftarrow calculeAvgFeatureCounts(\mathcal{D})$
2: $\omega \leftarrow randomInit()$
3: **repeat**
4: **for each** $s \in \mathcal{S}$ **do**
5: **for** $rrt\_repetitions$ **do**
6: $\zeta_i \leftarrow getRRTstarPath(s, \omega)$
7: $f(\zeta_i) \leftarrow calculeFeatureCounts(\zeta_i)$
8: **end for**
9: $\overline{f}^s_{RRT^*} \leftarrow (\sum_{i=1}^{rrt\_repetitions} f(\zeta_i))/rrt\_repetitions$
10: **end for**
11: $\overline{f}_{RRT*} \leftarrow (\sum_{i=1}^{S} \overline{f}^i_{RRT^*})/S$
12: $\nabla\mathcal{L} \leftarrow (\overline{f}_{RRT*} - \overline{f}_{\mathcal{D}})$
13: $\omega \leftarrow UpdateWeights(\nabla\mathcal{L})$
14: **until** convergence
15: **return** ω

Then we initialize the weights with an unsigned integer random value (Line 2). It is noteworthy that the weights are not being normalized during the learning iterations, so that changes in the value of one weight do not provoke the variation of the values of the rest of weights. They are normalized after the learning has finished. On the other hand, the features values for each node are normalized but this is not a requirement of the algorithm.

The key point is the gradient given by (6), which requires a comparison of the features counts obtained from the example trajectories and the expected value from the RRT* planner. The latter is obtained by running $rrt\_repetitions$ times the planner for the current weight values for each scenario considered (Line 6) and obtaining and normalizing the features counts (Line 7). In Lines 9 and 11 the averaged values are obtained.

Based on this comparison the weights of the cost function are updated using exponentiated gradient descent (line 13), as in [20]:

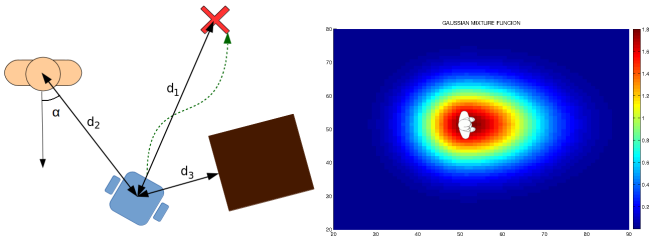
$$\omega_i \leftarrow \omega_i * e^{(\lambda/\phi)*\nabla\mathcal{L}_i} \tag{7}$$

where ϕ is the number of the current iteration of the algorithm, λ is an adjusting factor of the equation and $\nabla\mathcal{L}_i = \frac{\partial\mathcal{L}(\mathcal{D}|\omega)}{\partial\omega_i}$ is the i-th component of the gradient.

Finally, the learning process finishes when the variations of the weight values keep under a certain convergence value ϵ.

3 Cost function for social navigation

The social navigation task considered here involves the robot navigation in different house environments like rooms and corridors where some persons stand in different positions so that the robot has to avoid them to reach the goal.



(a) Features (b) Gaussian function

Fig. 1: (a) Features employed in the social cost function learned. $d1$, distance to the goal. d_2, distance from the people to the robot. α, angle between the person front and the robot location. d_3, distance to the closest obstacle. (b) Gaussian mixture function deployed over each person. The lateral bar shows the costs based on the color displayed.

A small set of well-known features have been considered here. They are the distance to the goal, the distance from the robot to the people in the scene, the angle of the robot position with respect to the people α, and the distance to the closest obstacle, as depicted in Figure 1a. Notice that this paper focuses on the use of the features for social robot navigation, but we will not get into details about the nature and importance of the features themselves.

Thus, three feature functions are combined to obtain the cost function employed in the RRT* planner. The function are computed for each sample x_k of the configuration space. The first one is just the Euclidean distance from the robot position to the goal:

$$f_1(x_k) = \|x_k, x_{goal}\| \tag{8}$$

The second feature function represents a proxemics cost with respect to the persons in the environment, and follows the model used by Kirby et *al.* [8]. This cost function is defined by a mixture of Gaussian functions, and its shape can be seen in Fig. 1. This cost function p depends on the distance (d_{jk}) and relative angle (α_{jk}) of the robot position x_k with respect to each person j in the scenario. The cost due to all persons in the scenario is integrated according to the next expression, where P is the total number of persons:

$$f_2(x_k) = \prod_{j=1}^{P} (p(d_{jk}, \alpha_{jk}) + 1) - 1 \tag{9}$$

Figure 1b shows this cost function for one person, which is implemented as a mixture of two Gaussian functions: the first function is asymmetric and placed in the front of the person with $\sigma_h = 1.20m$ the variance in the direction the person is facing, and a smaller variance in the sides $\sigma_s = \sigma_h/1.5$. The second Gaussian is placed in the back of the person with $\sigma_h = \sigma_s = 0.8$.

The third feature function uses the distance to the closest obstacle for each node x_k, $d_{jobs}(x_k)$ with the aim of motivating the robot to keep some distance from the obstacles. This cost is based on the costmap used by the navigation system of ROS [16], in which each obstacle has a defined inflation area around. This way, the cost is zero if the robot is far enough from any obstacle ($\delta = 2$ meters in our case):

$$f_3(x_k) = \begin{cases} 0, & \text{if } d_{obs}(x_k) > \delta, \\ (254-1)e^{(-\beta(d_{obs}(x_k)-r))}, & \text{otherwise} \end{cases} \tag{10}$$

where r is the inscribed radius of the robot and $\beta = 3$ in our implementation.

The values of the n (3 in this case) feature functions are normalised and the cost function for each node x_k is built adding its weighted values $c(x_k) = \sum_{i=1}^{n} \omega_i f_i(x_k)$ where $\omega_i \in [0, 1]$ and $\sum_i \omega_i = 1$.

Finally, the total cost along the Q nodes of the path ζ is obtained based on the *motion-cost* function employed by the RRT* algorithm to calculate the cost of moving from one node to the next one according to Equation (1).

4 Experimental results

A set of experiments have been performed to evaluate whether the algorithm is able to recover the characteristics of the taught trajectories. All the presented experiments were performed by using a library of RRT algorithms developed by the authors for research purposes. The library is available in the Github of the Service Robotics Lab[3] under BSD license. The hardware employed was an *i*7 processor 3770 with 12 GB DDR3 memory, where the planner was allowed to plan a path for 2 seconds.

In these experiments, we use the RRT* with a set of known weights in the cost function to generate the example trajectories in a set of scenarios as ground truth. Then, we use these trajectories to learn the cost function with the proposed algorithm in the same configurations. Particularly, we employed 25 different configurations (different initial robot position, goal position and different number of persons and positions) in different parts or rooms in a house map. Moreover, for each configuration, 25 RRT* example trajectories were recorded. To validate the approximation, the trajectories from 15 of these configurations were used to learn the weights of the cost function, and the 10 remaining configurations were employed to compare the resulting paths. Figure 2 shows 3 of the configurations used in the validation.

Figure 3 shows the evolution of the normalised weights values, feature counts and gradients along the iterations of the learning algorithm. As can be seen, the weights converge to values close to the ground-truth ones in few iterations committing a final error around the 16%. The difference in the feature counts expectations, which is the optimization objective in (3), quickly approaches zero, and so the gradients and the weights are stabilized. The relative error committed

[3] https://github.com/robotics-upo

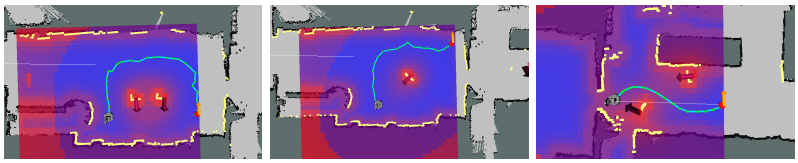
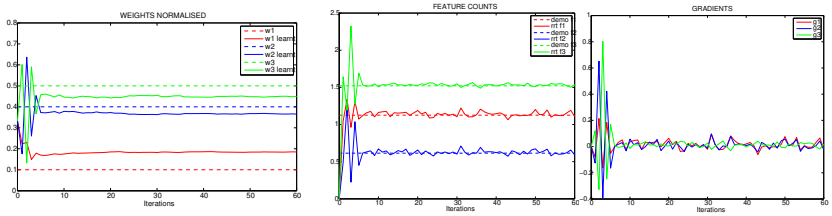


Fig. 2: Some of the scenarios employed in the cross-validation process. A coloured costmap based on the RRT* function cost is also shown.



(a) Weights (b) Feature counts (c) Gradients

Fig. 3: (a) Evolution of the weights during the learning iterations. (b) Evolution of feature counts. (c) Gradients.

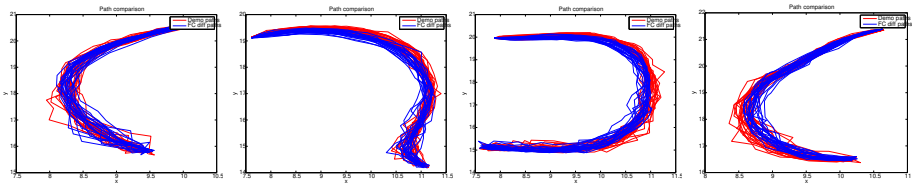
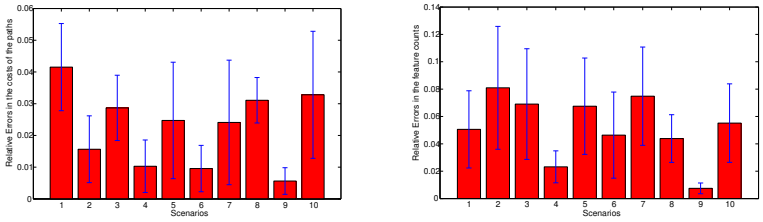


Fig. 4: Visual comparison of the demonstration paths (red lines) and the RRT paths obtained by using the weights learnt (blue lines). Trajectories 1, 5, 7 and 10 are presented from left to right.

in the weights learnt respect to the ground-truth weights is calculated as $RE_\omega = \|\bar{\omega}_\mathcal{D} - \bar{\omega}_{RRT^*}\|/\|\bar{\omega}_\mathcal{D}\| = 0.1620$.

Once the learning has finished, we can compare the demonstration paths and the RRT* paths using the weights learnt in the remaining configurations for cross-validation. A qualitative comparison of the paths can be seen in the Figure 4 for four of these trajectories (the rest are omitted for the sake of brevity). It can be seen that the behavior is very well reproduced in all the cases.

We can also compare the costs of the demonstration paths and the learnt RRT* paths. Figure 5 shows the averaged relative errors in the costs and in the feature counts. The error in feature counts is under the 8% in all the cases. On the other hand, the error in the costs is even lower being all the cases under the 4%. Moreover, it can be also seen in Fig. 4 that some of the trajectories with larger cost error (1 and 10) reproduce very well the demonstrations.



(a) Path cost (b) Feature counts

Fig. 5: (a) Relative errors in the costs of the demonstration paths and the RRT∗ paths using the learnt weights. (b) Relative errors in the feature counts.

5 Conclusions and future work

This paper presented an approach for teaching a robot behaviors based on demonstrations. To this end, a method based on IRL basis has been implemented and linked with a regular RRT* in order to learn the weights of its cost function, so the planner behaves similarly to the demonstrated behaviors. The method is simple to implement and allows to overcome the classic problems associated to IRLs based on MDPs. The proposed method is significantly less computational demanding than MDPs and simplify the generalization of the behavior thanks to the intrinsic benefits of RRTs.

The approach has been tested in simulation where the resulted learned cost function was able to properly imitate the desired behavior in the most of the cases. The feature counts always converged to values very close to the demonstrated values in the experiments, and the computed weights also allowed to reproduce the desired behavior reliably.

Future work will consider including the kinodynamic of the robot in the RRT* planner and the use of a richer set of features also employing the velocities of the robot and persons. Furthermore, a further mathematical analysis of the distributions of path costs followed by the RRT* planner as well as other optimization techniques to solve the problem will be considered.

References

1. Abbeel, P., Ng, A.Y.: Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the twenty-first international conference on Machine learning. pp. 1–. ICML '04, ACM, New York, NY, USA (2004), http://doi.acm.org/10.1145/1015330.1015430
2. Argali, B., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstrations. Robotics and Autonomous Systems 57, 469–483 (2009)
3. Feil-Seifer, D., Mataric, M.: People-aware navigation for goal-oriented behavior involving a human partner. In: Proceedings of the IEEE International Conference on Development and Learning (ICDL) (2011)

4. Ferrer, G., Garrell, A., Sanfeliu, A.: Robot companion: A social-force based approach with human awareness-navigation in crowded environments. In: Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on. pp. 1688–1694 (Nov 2013)
5. Hall, E.T.: The Hidden Dimension. Anchor (Oct 1990)
6. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. The International Journal of Robotics Research 30(7), 846–894 (2011), http://ijr.sagepub.com/content/30/7/846.abstract
7. Kirby, R., J. Forlizzi, J., Simmons, R.: Affective social robots. Robotics and Autonomous Systems 58, 322–332 (2010)
8. Kirby, R., Simmons, R.G., Forlizzi, J.: Companion: A constraint-optimizing method for person-acceptable navigation. In: RO-MAN. pp. 607–612. IEEE (2009)
9. Kretzschmar, H., Kuderer, M., Burgard, W.: Learning to predict trajectories of cooperatively navigating agents. In: Robotics and Automation (ICRA), 2014 IEEE International Conference on. pp. 4015–4020. IEEE (2014)
10. Kruse, T., Pandey, A.K., Alami, R., Kirsch, A.: Human-aware robot navigation: A survey. Robot. Auton. Syst. 61(12), 1726–1743 (Dec 2013), http://dx.doi.org/10.1016/j.robot.2013.05.007
11. Kuderer, M., Gulati, S., Burgard, W.: Learning driving styles for autonomous vehicles from demonstration. In: Proceedings of the IEEE International Conference on Robotics & Automation (ICRA), Seattle, USA. vol. 134 (2015)
12. Levine, S., Popovic, Z., Koltun, V.: Nonlinear inverse reinforcement learning with gaussian processes. In: Neural Information Processing Systems Conference (2011)
13. Luber, M., Spinello, L., Silva, J., Arras, K.: Socially-aware robot navigation: A learning approach. In: IROS. pp. 797–803. IEEE (2012)
14. Michini, B., Cutler, M., How, J.P.: Scalable reward learning from demonstration. In: IEEE International Conference on Robotics and Automation (ICRA). IEEE (2013), http://acl.mit.edu/papers/michini-icra-2013.pdf
15. Pacchierotti, E., Christensen, H., Jensfelt, P.: Evaluation of passing distance for social robots. In: IEEE Workshop on Robot and Human Interactive Communication (ROMAN). Hartfordshire, UK (Sep 2006)
16. Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software (2009)
17. Shiarlis, K., Messias, J., van Someren, M., Whiteson, S., Kim, J., Vroon, J., Englebienne, G., Truong, K., Evers, V., Perez-Higueras, N., Perez-Hurtado, I., Ramon-Vigo, R., Caballero, F., Merino, L., Shen, J., Petridis, S., Pantic, M., Hedman, L., Scherlund, M., Koster, R., Michel, H.: Teresa: A socially intelligent semi-autonomous telepresence system. In: Workshop on Machine Learning for Social Robotics at ICRA-2015 in Seattle (2015)
18. Sisbot, E.A., Marin-Urias, L.F., Alami, R., Siméon, T.: A Human Aware Mobile Robot Motion Planner. IEEE Transactions on Robotics 23(5), 874–883 (2007)
19. Trautman, P., Krause, A.: Unfreezing the robot: Navigation in dense, interacting crowds. In: IROS. pp. 797–803. IEEE (2010)
20. Ziebart, B., Maas, A., Bagnell, J., Dey, A.: Maximum entropy inverse reinforcement learning. In: Proc. of the National Conference on Artificial Intelligence (AAAI) (2008)