International Journal of Social Robotics (2018) 10:235–249
https://doi.org/10.1007/s12369-017-0448-1

Teaching Robot Navigation Behaviors to Optimal RRT Planners

Noé Pérez-Higueras[1] · **Fernando Caballero**[2] · **Luis Merino**[1]

Accepted: 15 November 2017 / Published online: 23 November 2017

Abstract
This work presents an approach for learning navigation behaviors for robots using Optimal Rapidly-exploring Random Trees (RRT*) as the main planner. A new learning algorithm combining both Inverse Reinforcement Learning and RRT* is developed to learn the RRT*'s cost function from demonstrations. A comparison with other state-of-the-art algorithms shows how the method can recover the behavior from the demonstrations. Finally, a learned cost function for social navigation is tested in real experiments with a robot in the laboratory.

Keywords Path planning · Learning from demonstration · Social robots

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 that humans can understand the robot's intentions [15]. This is called human-aware navigation.

This problem was initially tackled by including pre-programmed costs and constraints related to human-awareness into motion planners to obtain socially acceptable paths [11,36]. However, hard-coded social behaviors might be inappropriate [5]. In many implementations (for instance [12,24]), these costs are grounded in Proxemics theory [8]. However, as shown in [19], Proxemics is focused on people interaction, and it could not be suitable for navigating among people.

All in all, it is easier to demonstrate socially acceptable behaviors than mathematically defining them. Therefore, learning these social behaviors from data seems a more principled approach. In particular, we consider in this paper the application of telepresence robots [33] (see Fig. 1). Our goal is to increase the autonomy of such robots, freeing the users from the low-level navigation tasks. In this setup, it is very natural to collect navigation data online from the users controlling the telepresence robot. Then, this data is easily reproducible off-line and thus, used as examples to learn social navigation behaviors during idle times of the robot.

Thus, we present in the paper a new approach to learn navigation behaviors from demonstrations. We make use of Inverse Reinforcement Learning (IRL) concepts [21] and sampling-based costmap planners, like Optimal-RRTs (RRT*)[10], to identify the RRT cost function that best fits the example trajectories. Many IRL approaches for robot navigation are based on discrete Markov Decision Process (MDPs) to frame the problem [9,28,37]. MDPs allow dealing with uncertain outcomes of actions. However, MDPs present some disadvantages for robot motion planning, like poor scalability with the involved large (and typically continuous) state spaces, which leads to high computational complexity. The proposed method, which we have called Rapidly-exploring random Trees Inverse Reinforcement Learning (RTIRL), makes use of the RRT* instead of an MDP. It exploits the

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).

✉ Noé Pérez-Higueras
noeperez@upo.es

Fernando Caballero
fcaballero@us.es

Luis Merino
lmercab@upo.es

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

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

Fig. 1 Image of the telepresence robot developed in the TERESA project http://teresaproject.eu

advantages of the RRT* to deal with continuous state and control spaces and to scale with the state size. It is also general enough to be applied in different scenarios.

The contribution of this paper is threefold: a new method for learning robot navigation behaviors from demonstrations that employs widely-used navigation planners like RRT*; a comparison with other approaches from the state of the art that shows how the method adequately recovers the demonstrated behavior; and the application of the method to the human-aware navigation problem.

The paper is structured as follows. After a summary of the related work, next section describes the algorithm for learning from demonstrations using RRT*. Then, Sect. 3 describes the particular problem of social navigation considered in the paper. Section 4 presents a benchmarking evaluation of the method, including a comparison with other algorithms from the state of the art. In Sect. 5, a set of experiments with a real robot using a learned cost function for social navigation is presented. Finally, Sect. 6 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 to the problem of human-aware navigation, and in particular, an interesting approach is Learning from Demonstrations (LfD) [2]. One successful technique to do that is Inverse Reinforcement Learning (IRL) [21]: 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 similar robot policy. Other LfD techniques that directly learn maps from inputs to actions, like Behavioral Cloning [35], are in general more adequate for learning low-level control policies than for planning [2]. An interesting characteristic of IRL techniques is that they try to learn the underlying behavior that the demonstrator is showing, and therefore, they allow to generalize to different scenarios or situations where to perform the learned task. Furthermore, the learned reward functions can be transferred to different planners.

The literature tackles the IRL problem from different points of view. The initial projection method proposed by Abbeel [1] was followed by other well-known approximations. A probabilistic method based on the principle of maximum entropy is presented in [39], or a maximum margin structured prediction based on subgradients is employed in [30,31]. In [27], a Bayesian perspective is taken to solve the problem.

In most of the existing models, the IRL technique makes use of MDPs as the underlying process. However, it is complex to encode general problems with MDPs due to its computational complexity which scales poorly with high dimensionality and large state spaces.

Different solutions have been considered to deal with the main MDP drawbacks. The computational cost is managed in [20] by using a Bayesian nonparametric mixture model to divide the observations and obtain a group of simpler reward functions to learn. In [18], deterministic MDPs with large, continuous state and action spaces are handled by considering only the shape of the learned reward function in the neighborhood of the experts demonstrations. Furthermore, neural networks are applied in [38] to learn non-linear policies. In [23], a socially normative behavior is learned using flexible graph-based representation to model the underlying MDP.

Other authors try to replace or represent the MDP by another process according to the aim of the task. In [30], the MDP can be replaced by an A* search algorithm. Different car driving styles are learned in [16] by modeling the car dynamics using a time-continuous spline representation instead of an MDP. Another example is [14], where the cooperative navigation behavior of humans is learned

using Hamiltonian Markov chain Monte Carlo sampling to compute the feature expectations over high-dimensional continuous distributions.

While the previous approaches show promising results, they employ representations tailored to the problem at hand. In this paper, we present an algorithm for learning robot navigation behaviors from demonstrations using RRT* as main planner [10]. RRT* planners are very common in robotics applications; they are asymptotically optimal, can cope with continuous state and action spaces and kinodynamic constraints, reason implicitly about obstacles and can be easily extended to spaces with higher dimensions. We aim at creating a new learning algorithm combining both IRL and RRT* techniques 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. A similar idea of ours is recently presented in [34] where the Maximum Margin Planning (MMP) method is extended to learn RRT* cost functions.

This work is an extension of the work initially presented in [25]. The paper has been significantly extended, including a thorough evaluation of the learning performance through a comparison with similar state-of-the-art approximations; the application of the method to the social navigation problem considering a wider set of features; and a comparison of the resultant navigation behavior with respect to non human-aware planners.

2 Learning a RRT* Cost Function

2.1 IRL Formulation with RRT* Planners

RRT* is a technique for optimal motion planning employed extensively in robotics [10]. It is flexible and easily adapted to different scenarios and problems. RRT* approaches reason about collisions with obstacles at moderate computational cost even in high dimensionality. They explore the configuration space to obtain optimal paths on cost spaces, and the kinodynamic extension allows reasoning about the robot dynamics.

The RRT* algorithm considers that a cost function is associated with each point x in the configuration space of the robot. The algorithm 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 finite set of configuration points $\zeta = \{x_1, x_2, \ldots, x_N\}$.

Without loss of generality, we can assume that the cost for each point x can be expressed as a weighted linear combination of a set of J functions $f_j(x)$:

$$c(x) = \sum_{j=1}^{J} \omega_j f_j(x) = \omega^T f(x) \tag{1}$$

The function $f(x) = [f_1(x), f_2(x), \ldots, f_J(x)]^T$ is based on J measurable characteristics that will be called features. The particular features will depend on the task to be learned, and are used to extract the information from the robot configurations that are relevant for that particular task. While not explicitly indicated, the value of the features will depend also on the scenario s (given by the position of obstacles, people, goals and any other relevant information external to the robot) where the planning takes place.

The cost of a path is then the sum of the cost for all points in it. Particularly, in the RRT*, the cost of a path is the sum of the sub-costs of moving between pairs of points in the path:

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

$$= \omega^T \underbrace{\sum_{i=1}^{N-1} \frac{f(x_i) + f(x_{i+1})}{2} \|x_{i+1} - x_i\|}_{f(\zeta)} \tag{3}$$

$$= \omega^T f(\zeta) \tag{4}$$

where $f(\zeta)$ is called the feature count vector of path ζ. Thus, for a given weights vector ω, the algorithm will return trajectories that try to minimize this cost.

Given a set of demonstration trajectories by an expert,

$$\mathcal{D} = \{\zeta_1, \zeta_2, \ldots, \zeta_D\} \tag{5}$$

and the definition of the feature function $f(x)$ in (1), 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.

The concept of similarity in this context is ambiguous. One may want to reproduce exactly the same trajectories in the same situations. However, it is necessary to learn a representation that can generalize to other cases. As in [1,16], this can be achieved by using the mentioned features as a measure of similarity. The objective is then to model the underlying trajectory distribution of the expert $p(\zeta|\omega)$ with the constraint that the expected value of the features for the path generated by the model is the same as the expected value of the features for the given demonstrated trajectories:

$$\mathbb{E}(f(\zeta)) = \frac{1}{D} \sum_{i=1}^{D} f(\zeta_i) \tag{6}$$

There are many distributions $p(\zeta|\omega)$ that achieve the previous constraint. Applying the Maximum Entropy Principle

[13,39] to the IRL problem leads to the following form for the probability density for the trajectories returned by the demonstrator:

$$p(\zeta|\omega) = \frac{1}{Z(\omega)} e^{-\omega^T f(\zeta)} \tag{7}$$

where the partition function $Z(\omega) = \int e^{-\omega^T f(\zeta)} d\zeta$ is a normalization function that does not depend on ζ. Thus, this model assumes that the expert is exponentially more likely to chose a trajectory with lower cost $\omega^T f(\zeta)$.

Under this model, the (log-)likelihood of the demonstrated trajectories is given by:

$$\mathcal{L}(\mathcal{D}|\omega) = -D \log(Z(\omega)) + \sum_{i=1}^{D} (-\omega^T f(\zeta_i)) \tag{8}$$

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

$$\frac{\partial \mathcal{L}(\mathcal{D}|\omega)}{\partial \omega} = D \underbrace{\frac{1}{Z(\omega)} \int f(\zeta) e^{-\omega^T f(\zeta)} d\zeta}_{\mathbb{E}(f(\zeta))} - \sum_{i=1}^{D} f(\zeta_i) \tag{9}$$

$$= D(\mathbb{E}(f(\zeta)) - \frac{1}{D} \sum_{i=1}^{D} f(\zeta_i)) \tag{10}$$

We arrive at the original constraint in (6) by setting this gradient to zero. This means, as indicated in [17], that the IRL problem under the Maximum Entropy distribution is equivalent to maximizing the likelihood of the demonstrations when assuming an exponential distribution.

Maximizing the likelihood (8) to obtain the set of weights cannot be solved analytically. But the gradient (10) can be used to apply gradient ascend techniques to solve this problem. As mentioned in [16], this gradient can be intuitively explained. If the value of one of the features for the trajectories returned by the planner is higher than the value in the demonstrated trajectories, the corresponding weight should be increased to penalize those trajectories.

The main problem with the computation of the previous gradient is that it requires to estimate the expected value of the features $\mathbb{E}(f(\zeta))$ for the generative distribution (7) over the continuous space of trajectories.

$$\mathbb{E}(f(\zeta)) = \frac{1}{Z(\omega)} \int f(\zeta) e^{-\omega^T f(\zeta)} d\zeta \tag{11}$$

In [13], a probabilistic generative model for trajectories is derived from data, and the expectation is computed by Monte Carlo Chain sampling methods, which are very computationally demanding. One option is to approximate the previous expectation by the features of the most likely trajectory [16].

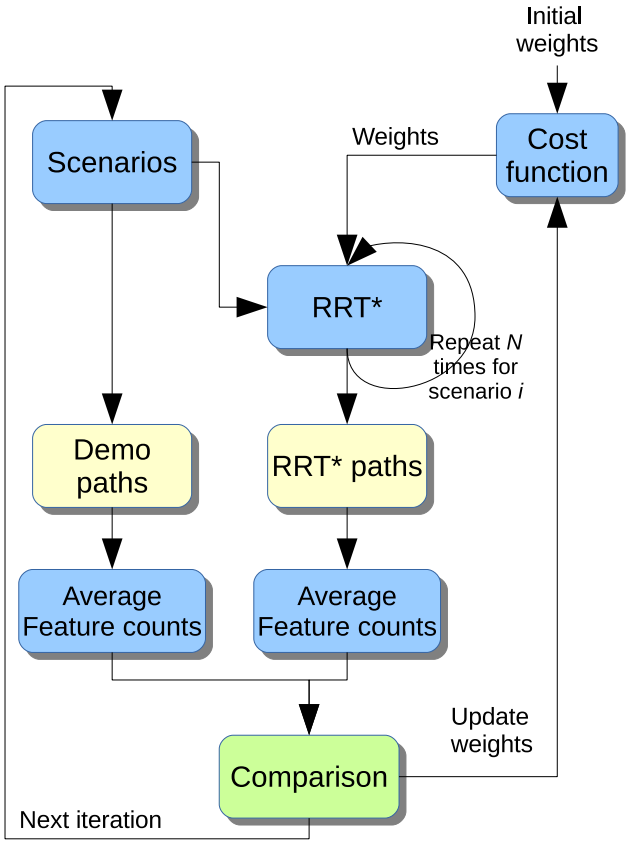

Fig. 2 General learning scheme proposed to adjust the weights of the cost function of a RRT* planner

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 the cost (the most likely) given infinite time. As the planning time is limited, the RRT* will provide trajectories with some variability on the features, and thus the feature expectation $\mathbb{E}(f(\zeta))$ is computed by running several times the planner between the start and goal configurations. This is then used to calculate the gradient and adapt the weights employed in the RRT* planner.

The experimental results will show that the method can recover the behaviors taught by the expert.

2.2 RTIRL Algorithm

The general scheme of the learning algorithm proposed is presented in Fig. 2. First, we define a scenario as a particular situation where to perform the task proposed (for instance, a particular position of the obstacles, people positions and orientations, starting and goal positions, for the task of social navigation). So, for each different scenario $s \in \mathcal{S}$ present in the demonstrations, we plan a path to the respective goal using the RRT* planner. The planner has a limited time to

Algorithm 1 RTIRL

Require: Trajectory examples $\mathcal{D} = \{\zeta_1^1, \ldots, \zeta_D\}$ from $\mathcal{S}$ 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

plan, so small variations of the resulting path can arise. So, as commented before, we deal with that by repeating the plan some times and calculating the average values over the number of repetitions. Then, the feature count of the demonstration paths and the RRT* paths are calculated and averaged over all the scenarios. The difference between these values is used to update the weights of the cost function at each iteration. Thus, we get the weights that make the cost function fit better the demonstration paths as output of the algorithm.

The approximation is described in more detail in Algorithm 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 Eq. (3).

Then we initialize the weights with a random value (Line 2). It is noteworthy that the weights are being normalized during the learning iterations and the features values for each node are also normalized for the sake of clarity of representation, but this is not a requirement of the algorithm.

The key point is the gradient given by (10), 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 estimating and normalizing the features counts (Line 7). In Lines 9 and 11, the averaged values are computed.

Based on this comparison the weights of the cost function are updated using gradient descent (line 13):

$$\omega_{i+1} \leftarrow \omega_i - \frac{\lambda}{\phi} * \nabla\mathcal{L}_i \tag{12}$$

where ϕ is a value for stabilization which is being incremented after each iteration, 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.

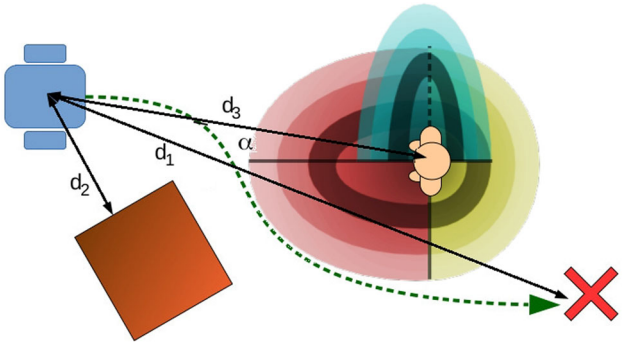

Fig. 3 Some of the features employed in the social cost function learned. $d1$, distance to the goal. d_2, distance to the closest obstacle. d_3, distance from the people to the robot. α, angle between the person front and the robot location. The three Gaussian functions deployed over the people are also represented

3 Features for Social Navigation

The general learning approach presented above can be applied to any task that can be solved with a RRT* planner and by defining the adequate features that describe the task properly. From this section on, we apply the algorithm to learn the particular task of human-aware navigation. So, a set of features specially designed for that task is presented here.

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

The selection of the adequate features involved in the social navigation task is not trivial [29]. A review and analysis of the most used features in the literature are presented in [37]. Taking into account these previous works, a set of commonly-used features have been considered here. In particular, they are the distance to the goal, the distance to the closest obstacle, and the distance from the robot to the people in the scene and the angle of the robot position with respect to the people α, as depicted in Fig. 3. We are not using dynamic features as velocities or accelerations here, which we will leave for future work.

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

$$f_1(x_k) = \|x_k, x_{goal}\| \tag{13}$$

The second feature function uses the distance to the closest obstacle for each node x_k, with the aim of motivating the robot to keep some distance from the obstacles. This value grows when the distance to the obstacle decreases. A distance transform function is employed to obtain the distance to the closest obstacle for each sample [3]. Then, a normalized value is obtained by using this distance according to Eq. (14).

$$f_2(x_k) = \frac{a_1}{\gamma * (\|x_k, x_{closest\_obs\_k}\| + a_2)} \tag{14}$$

where a_1 and a_2 are the parameters that define the function and take the values 2 and 0.2 respectively. γ is a normalization parameters that takes the value 10 in this case.

Then, other three feature functions representing a proxemics cost with respect to the people in the environment are employed. These cost functions are defined by Gaussian functions and are inspired by the model used by Kirby et al. [12]. One Gaussian function is placed in the front of the person, another one in the back, and the last one in the right side of the person. The shape of these Gaussian functions can be seen in Fig. 3. These cost functions p depend 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 costs due to all persons in the scenario are integrated according to the next expressions, where $\mathcal{P}$ represents the set of people:

$$f_3(x_k) = max\{p_{front}(d_{jk}, \alpha_{jk}), \forall j \in \mathcal{P}\} \tag{15}$$
$$f_4(x_k) = max\{p_{back}(d_{jk}, \alpha_{jk}), \forall j \in \mathcal{P}\} \tag{16}$$
$$f_5(x_k) = max\{p_{right\_side}(d_{jk}, \alpha_{jk}), \forall j \in \mathcal{P}\} \tag{17}$$

The first Gaussian function is asymmetric and placed in the front of the person with $\sigma_h = 1.20$ (m) being the variance in the direction the person is facing, and considering a smaller variance towards the sides $\sigma_s = \sigma_h/1.5$. The second Gaussian is placed in the back of the person with $\sigma_h = \sigma_s = 0.8$. Finally, a third Gaussian is placed in the right side of the person with $\sigma_h = 0.8$ and $\sigma_s = 0.8/2.5$. This function can be useful in corridor scenarios where, usually in Europe, the people usually walk on the right-hand side [15]. Another one could be placed on the left side, so the system should learn the correct weight for each side.

The values of the J (5 in this case) feature functions are normalised and the cost function for each node x_k is built adding its weighted values $c(x_k) = \sum_{i=1}^{J} \omega_i f_i(x_k)$ where $\omega_i \in [0, 1]$ and $\sum_i \omega_i = 1$.

Finally, the total cost of the N 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 Eq. (2).

4 Experimental Results

In this section, we validate the ability of the proposed RTIRL algorithm to recover the cost function and the behavior implicitly encoded by the demonstrations. To do so, a ground-truth cost function is employed to obtain a set of demonstration trajectories from a motion planner optimizing such function. Then, these trajectories are used by the proposed learning algorithm. This way, the learnt cost function can be directly compared with the ground-truth one.

Moreover, we present a comparison with two learning from demonstration algorithms of the state-of-the-art: the Maximum Margin Planning algorithm (MMP) [30], which uses an A* algorithm as a planner, and a new learning algorithm called Rapidly Exploring Learning Trees (RLT*) [34], which adapts MMP to use a RRT* as a planner. These algorithms have been chosen because they follow the same idea of using a planning algorithm instead of an MDP, and these planning algorithms are also two of the most used planners in robot navigation.

The MMP is a method for learning to plan which attempts to automate the mapping from perception features to costs (or rewards) as well as IRL tries to find the weights of a cost (or reward) function of an MDP. The idea behind MMP is to determine the set of weights that makes that the cost of the demonstrated trajectories are lower than any other path between the same start and goal configurations. For better generalization, the cost function is augmented by a margin that reduces the costs of paths far from the demonstrations. The notion of closeness to demonstrations is defined by a loss-function $\mathcal{L} : \zeta \times \zeta_D \rightarrow \mathbb{R}_+$ which defines for each path ζ how much the learner pays for failing to match the example behavior ζ_D. This loss function value is inverted $(1 - \mathcal{L}(\zeta(k), \zeta_D))$ and added to the cost function. This way, the algorithm tries to make any demonstrated path better than any other path by a margin that scales with the size of the loss.

The loss-function employed in the A*-based MMP algorithm [30] counts the number of states that the planner visits but the teacher did not. It is also recommended to relax this function so that nearby paths are also admissible. In the case of RLT* the loss-function used takes into account the nearby paths but it is not explicitly specified [34]. Therefore, in our implementation, we have followed the idea of admitting also nearby path by defining a loss-function, used in both methods (MMP and RLT*), which add a cost for each point of the evaluated path based on the Euclidean distance from that point to the closest point of the demonstrated path according to:

$$\mathcal{L}(\zeta(k), \zeta_D) = \begin{cases} 1 & \text{if } d_{min}(\zeta(k), \zeta_D) > 1 \\ d_{min}(\zeta(k), \zeta_D) & \text{otherwise} \end{cases} \tag{18}$$

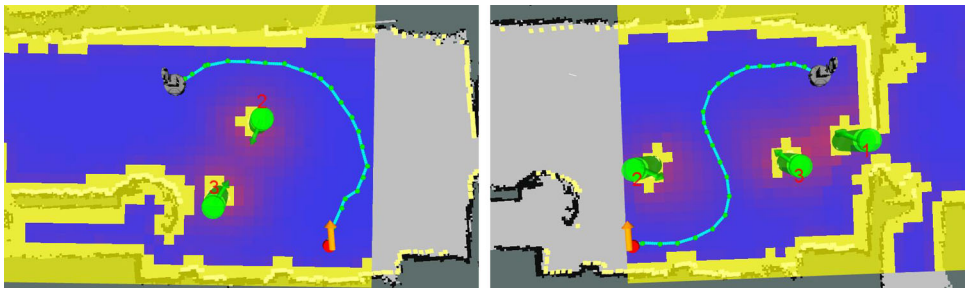

Fig. 4 Some of the scenarios employed in the validation (scenario 9 and scenario 19 respectively). The green cylinders represent the people in the scene indicating their orientations with an arrow. A low-resolution coloured costmap is also shown

where the function d_{min} calculates the Euclidean distance between the point k of the path ζ and the closest point of the demonstration path ζ_D.

4.1 Algorithm Validation

In order to validate the algorithm performance, we have used a set of demonstration trajectories obtained by a planner optimizing a given ground-truth cost function unknown to the learner. That is, the demonstration trajectories have been recorded by the planner using a cost function with the features explained in Sect. 3 and a known set of weights. This way, we can compare the weights learned with the ground-truth weights and the path costs of the demonstrations and the learned paths. The paths generated with the cost function computed with the learning algorithm are expected to recover the behavior of the demonstration trajectories. Moreover, we should obtain approximately the same weights of the cost function used to generate the demonstration trajectories, our ground-truth, in case that a unique set of weights can replicate the desired behavior.

Particularly, we have employed the ground-truth cost function to record 30 trajectories, each one in a different demonstration scenario (different initial robot position, goal position and person positions in different parts or rooms in a house map). Also, we varied the number of persons in the scenes from 1 to 3. These demonstrations have been recorded by using the RRT* planner with the ground-truth cost function proposed. All the trajectories are performed in a fixed area of 100 m^2 with the robot in the center. And the time for plan calculation employed is 10 s, which have been seen that it is enough to converge close to the optimum given the space for planning.

Testing and validation of the approximation will be based on splitting the 30 different demonstration trajectories into 20 trajectories to learn the weights of each cost function, and 10 configurations to compare the resultant paths. An example of the scenarios employed in the validation can be seen in Fig. 4 where the scenarios 9 and 19 are presented. In order to have small well-defined sets of trajectories in different enough scenarios, we have grouped the trajectories used for learning and evaluation in three trials for cross-validation according to Table 1.

Table 1 Demonstration trials

Trial 1	Learning	[1–20]
	Evaluation	[21–30]
Trial 2	Learning	[11–30]
	Evaluation	[1–10]
Trial 3	Learning	[1–10, 21–30]
	Evaluation	[11–20]

Table 2 Relative error in the weights committed by the learning algorithms in the three trials

RE_ω	RTIRL	RLT*	MMP
Trial 1	0.101	0.208	0.524
Trial 2	0.057	0.197	0.361
Trial 3	0.047	0.109	0.561
Mean:	0.068 ± 0.016	0.171 ± 0.031	0.482 ± 0.062

The mean values and the standard errors are also shown

The time the RRT* is allowed to plan a path during the execution of the learning algorithms RLT* y RTIRL is 3 s (other times are also considered in Sect. 4.2). In the case of the MMP learning, the resolution of the discretization of the state space for the A* planning is 0.05 m, which is considered a good resolution to get a smooth A* path. Finally, 100 iterations of each learning algorithm have been performed. As the ground-truth weights are known (ω_{gt}), we can calculate the relative errors committed in the final weights learned according to:

$$RE_\omega = \frac{\|\omega_{gt} - \omega\|}{\|\omega_{gt}\|} \tag{19}$$

These relative errors obtained in the weights by each algorithm for the 3 trials are shown in Table 2. As can be seen, the RTIRL approximation can recover the ground-truth weights with a small error for all the trials committing a final average error much lower than the other approximations.

Once the learning phase has finished, we can compare the 10 trajectories reserved for evaluation of the 3 trials with the

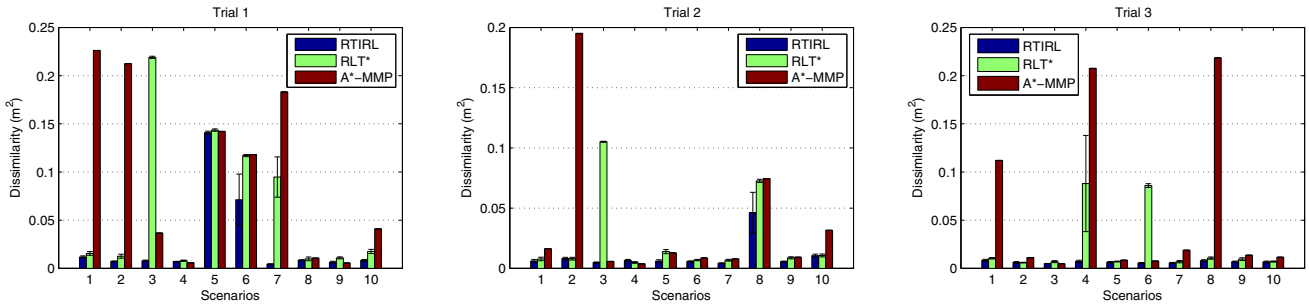

Fig. 5 Dissimilarity values for the 10 configurations for evaluation of the 3 trials

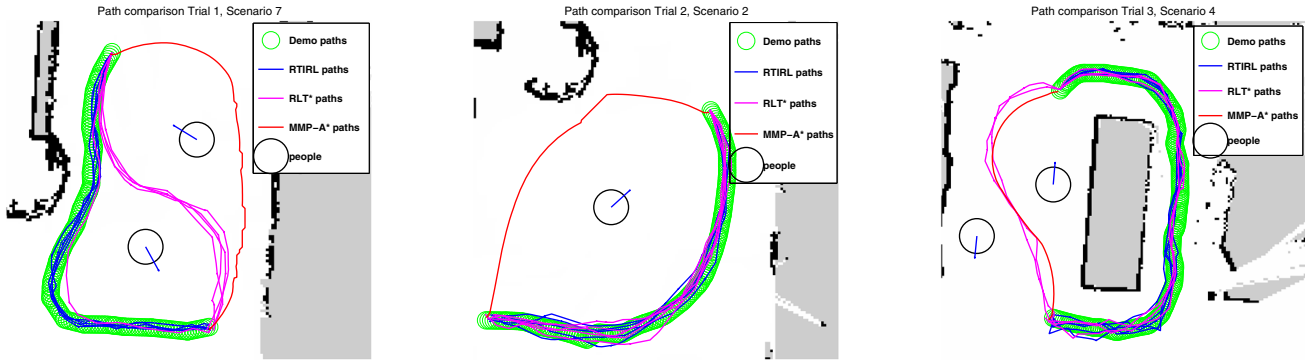

Fig. 6 Visual comparison of the paths learned in some of the scenarios for evaluation of the three trials

trajectories computed by the RRT* (and A* planner in case of MMP) with the estimated cost functions in order to visualize how close to the trajectories are in other scenarios. The planning time for the RRT* planner and the space discretization for the A* planner are the same that the ones employed in the learning phase. To overcome the randomness of the RRT* algorithm, 5 paths are planned and an average computation over these paths is performed. The comparison is performed using the following metrics:

4.1.1 Path Dissimilarity Comparison

The first metric, that we called dissimilarity, is used to compare how different are two given paths. This metric is an estimation of the area contained between two paths ζ_1 and ζ_2 by employing the Riemann's summation:

$$dissimilarity(\zeta_1, \zeta_2) = \frac{\sum_{k=1}^{N-1} \frac{d(\zeta_1(k),\zeta_2)+d(\zeta_1(k+1),\zeta_2)}{2} \|\zeta_1(k) - \zeta_1(k+1)\|}{\delta} \quad (20)$$

where the function d calculates the Euclidean distance between the point k of the path ζ_1 and its closest point of the path ζ_2. δ is the number of divisions that we performed in the path ζ_1. In this case, we have split the paths into slots of 10 *cm* in which we calculate the area. The paths are more similar when the dissimilarity approaches zero.

Figure 5 shows the dissimilarity values (along with the standard errors in the case of the 5 paths of RRT*-based approximations) for the trials 1, 2 and 3 respectively. As can be seen, for trial 1, the dissimilarity of the RTIRL method is the lowest in most of the cases. The other algorithms lead to big dissimilarity values in some scenarios while the RTIRL method keep a low dissimilarity except for scenarios 5 and 6 where none of the algorithms is able to lead to a good path. In the case of the trial 2, the same happens again. The MMP weight set is not able to reproduce correctly the behavior in scenarios 2, 8 and 10. The RLT* weight set fails in scenarios 3 and 8. The RTIRL algorithm only partially fails in scenario 8 because some of the 5 paths take another homotopy. Finally, for trial 3, the RTIRL was able to reproduce very well all the evaluation paths while the MMP and RLT* cost functions lead to paths farther from the ground-truth in scenarios 1, 4 and 8, and 4 and 6 respectively.

For a better understanding of the dissimilarity values, Fig. 6 shows a visual comparison of the paths corresponding with some of the worst cases of the scenarios for evaluation observed in the dissimilarity comparison presented in the Fig. 5. As can be seen, the weight set learned by the RTIRL approximation (blue lines) leads the paths closer to the demonstrated ones (green circles) in most of the cases.

Table 3 summarizes the dissimilarity values for the 3 trials. The results obtained by the weights learned with the RTIRL algorithm are better than the state-of-the-art methods.

Table 3 Dissimilarity means and standard errors committed by the learning algorithms in the three trials

$Dissim$	RTIRL	RLT*	MMP
Trial 1	0.027 ± 0.014	0.065 ± 0.024	0.098 ± 0.028
Trial 2	0.010 ± 0.004	0.025 ± 0.011	0.037 ± 0.022
Trial 3	0.007 ± 0.000	0.024 ± 0.011	0.061 ± 0.027
Mean	0.015 ± 0.006	0.038 ± 0.015	0.049 ± 0.025

Table 4 Mean relative errors in the feature counts of the paths with the standard errors committed by the learning algorithms in the three trials

RE_{fc}	RTIRL	RLT*	MMP
Trial 1	0.147 ± 0.054	0.300 ± 0.080	0.344 ± 0.081
Trial 2	0.132 ± 0.054	0.228 ± 0.088	0.214 ± 0.066
Trial 3	0.061 ± 0.008	0.152 ± 0.059	0.201 ± 0.059
Mean	0.113 ± 0.038	0.227 ± 0.076	0.253 ± 0.069

4.1.2 Feature Count Comparison

Another interesting comparison is the feature count of the paths $f(\zeta)$, obtained according to Eq. (3). Figure 7 shows the relative errors committed in the feature counts of the paths with respect to those of the demonstration paths. Again, the RTIRL method commits the lowest errors in most of the cases while the other methods fail strongly in some cases. These results are stated in Table 4, which averages the results of the 3 trials. As can be seen, the RTIRL method reaches significant better results than the RLT* and MMP methods which performance is quite similar according to this metric.

4.1.3 Path Cost Comparison

Finally, as the weights of the ground-truth cost function are known, we can also compare the costs of the learned paths and the demonstration paths. To do that, we can calculate the relative error in the cost of the paths according to:

$$RE_{costs}(\zeta_l^i, \zeta_d^i) = \frac{\omega_{gt}^T (f(\zeta_l^i) - f(\zeta_d^i))}{\omega_{gt}^T f(\zeta_d^i)} \tag{21}$$

where ζ_l^i is the path obtained in the scenario i with the cost function learned and ζ_d^i is the demonstration path in the scenario i. ω_{gt} is the weight vector of the ground-truth function.

The relative errors in the costs of the paths for the evaluation scenarios of the 3 trials is shown in Fig. 8. As can be seen, the results are quite similar to those obtained for the relative error in the feature count, where the error committed by the RTIRL algorithm is much lower than the error reached by the other approximations. Moreover, the average results for the trials in presented in Table 5. Again, the RTIRL clearly improves the results of the RLT* and MMP methods.

4.1.4 Statistical Significance

A Welch's T-test is employed to compare the results of the methods in the 3 trials. Table 6 shows the results of the t-tests when comparing the different metrics along with the p-values obtained. As can be observed, the better results reached by the RTIRL method can be considered significant (for a 0.05 level) whereas the differences between the RLT* and the MMP method can be neglected in the setup presented.

4.2 Learning with Different RRT* Planning Times

We also evaluate the performance of the proposed learning algorithm regarding the time that the RRT* planner is allowed to plan a path during the learning process (this time may be different from that used in execution after learning) and the number of repetitions of the planning. Table 7 shows the average relative error committed in the weights for the methods RTIRL y RLT* according to different planning times of the RRT* planner (0.5, 3 and 9 s) and different number of planning repetitions (10 repetitions, 5 repetitions and 1 repetition respectively). As can be observed, the algorithm behaves well even for short planning times. The relative error in the weights just varies around a 3% in both algorithms. These results highlights the importance of the number of

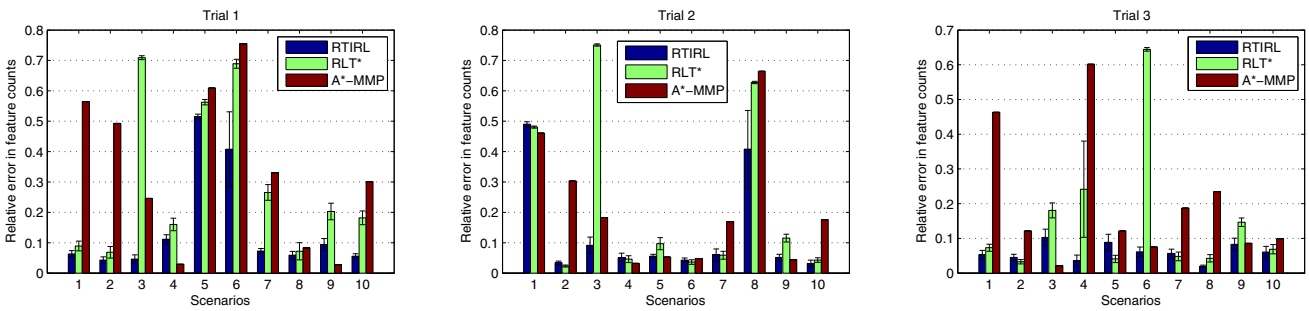

Fig. 7 Relative error in the feature counts for the 10 configurations for evaluation of the 3 trials

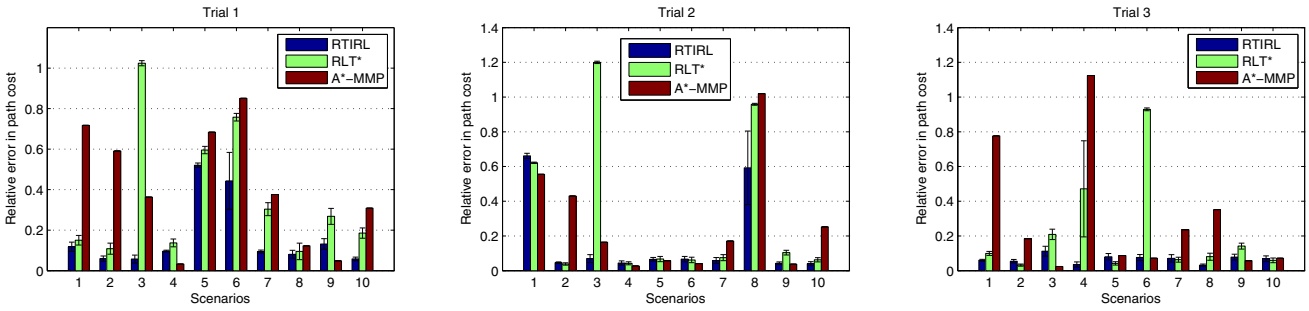

Fig. 8 Relative error in the cost of the paths for the 10 configurations for evaluation of the 3 trials

Table 5 Mean relative errors in the path costs with the standard errors committed by the learning algorithms in the three trials

RE_{costs}	RTIRL	RLT*	MMP
Trial 1	0.166 ± 0.054	0.363 ± 0.101	0.410 ± 0.092
Trial 2	0.169 ± 0.077	0.323 ± 0.139	0.276 ± 0.100
Trial 3	0.067 ± 0.007	0.213 ± 0.090	0.299 ± 0.116
Mean	0.134 ± 0.046	0.300 ± 0.110	0.328 ± 0.103

Table 6 Results of the Welch's t-test

T-tests	RTIRL-RLT*	RTIRL-MMP	RLT*-MMP
$Dissim.$	$p = 0.041$	$p = 0.003$	$p = 0.124$
RE_{fc}	$p = 0.031$	$p = 0.005$	$p = 0.663$
RE_{costs}	$p = 0.024$	$p = 0.006$	$p = 0.828$

Table 7 Average relative error in the learned weights committed by the learning algorithms according to different planning times of the RRT* planner used in the learning phase

RE_{ω}	0.5 s (10 rep)	3 s (5 rep)	9 s (1 rep)
RTIRL	0.098	0.068	0.077
RLT*	0.149	0.171	0.180

repetitions. Even if the time for planning is short, we can still have a good estimation of the average features values if we repeat the plan several times. The results also show that the RTIRL is all in all more efficient in terms error versus planning time than the RLT* algorithm.

5 Evaluation of the Social Navigation

In this section, the cost function for social navigation learned using the RTIRL algorithm is evaluated by performing real experiments with the TERESA robot in the laboratory. We perform a comparison between the social capabilities of a navigation system employing the cost function learned from a set of demonstrations (which we will denote Social Navigation System (SNS)) and a standard navigation stack where all entities in the scene are considered as simple obstacles. For the latter we employ the popular move_base framework from ROS[1] [26] (denoted Non-Social Navigation System (Non-SNS)). The default configuration parameters have been used but the related ones to the robot footprint, velocities and accelerations than have been adjusted to the TERESA robot.

The library of RRT algorithms and the rest of necessary modules for the SNS have been developed by the authors and are available in the Github of the Service Robotics Lab[2] under BSD license. A RRT* replanning each 0.5 s in a local area surrounding the robot and using the learned costs is employed for motion planning, and a simple controller derived from the Dynamic Windows Approach algorithm [7] is used to follow the RRT path.

5.1 Metrics

A set of simple metrics for a social evaluation used in the literature [22], some of them based on the Proxemics theory [8], has been considered here:

– *Time to reach the goal* (T_p) Time since the robot start the navigation until the goal is correctly reached.

$$T_p = (T_{goal} - T_{ini}) \tag{22}$$

– *Path length* (L_p) The length of the path followed by the robot from the initial point to the goal position.

$$L_p = \sum_{i=1}^{N-1} \|x_r^i - x_r^{i+1}\|_2 \tag{23}$$

– *Cumulative heading changes* (CHC) It counts the cumulative heading changes of in the robot trajectory measured

[1] http://wiki.ros.org/move_base.

[2] https://github.com/robotics-upo/upo_robot_navigation.

by angles between successive waypoints. It gives a simple way to check on smoothness of path and energy [22] so a low value is desirable.

$$CHC = \sum_{i=1}^{N-1} (h_r^i - h_r^{i+1}) \tag{24}$$

where h_r^i indicates the heading of the robot in the position i. The angles and their difference are normalized between $-\pi$ and π.

– *Average distance to closest person* (CP_{avg}) A measure of the mean distance from the robot to the closest person along the trajectory.

$$CP_{avg} = \frac{1}{N} \sum_{i=1}^{N} (\|x_r^i - x_{cp}^i\|_2) \tag{25}$$

where x_{cp}^i indicates the position of the closest person to the robot at step i.

– *Minimum and maximum distance to people* (CP_{min} and CP_{max} respectively) The values of the minimum and the maximum distances from the robot to the people along the trajectory. It can give an idea of the dimension of the robot trajectory with respect to the people in the space.

$$CP_{min} = \min\{\|x_r^i - x_{cp}^i\|_2 \mid \forall i \in N\} \tag{26}$$

$$CP_{max} = \max\{\|x_r^i - x_{cp}^i\|_2 \mid \forall i \in N\} \tag{27}$$

– *Personal space intrusions* (CP_{prox}) This metric is based on the Proxemics theory which define personal spaces around people for interaction [8]. These areas are defined as:

  – Intimate. Distance shorter than 0.45 m.
  – Personal. Distance between 0.45 and 1.2 m.
  – Social. Distance between 1.2 and 3.6 m.
  – Public. Distance longer than 3.6 m.

  Thus, the metric classifies the distance between the robot and the closest person at each time step in one of the Proxemics spaces (in our case we use only three: Intimate, Personal and Social+Public), and obtain a percentage of the time spent in each space for the whole trajectory:

$$CP_{prox}^k = \left(\frac{1}{N} \sum_{j=1}^{N} \mathcal{F}(\|x_r^j - x_{cp}^j\|_2 < \delta^k) \right) * 100 \tag{28}$$

where N is the total number of time steps in the trajectory, δ defines the distance range for classification defined by $k = \{Intimate, Personal, Social+Public\}$, and $\mathcal{F}(\cdot)$ is the indicator function. The desirable behavior should lead the robot to spend most of the time in the Social or Public distance range, but this depends on the dimensions of the space and the density of people in the area. So, for a small area shared with people, intrusions in the Personal area are acceptable whereas for a big open space the robot should stay in the Social and Public distances. In any case, the intrusions in the Intimate space should be avoided.

– *Interaction space intrusions* (IS_{prox}) This metric is inspired by the work of Okal and Arras [22] in formalizing social normative robot behavior, and it is related to groups of interacting persons. It measures the percentage of time spent by the robot in the three Proxemics spaces considered with respect to an interaction area formed by a group of people that are interacting with each other. The detection of the interaction area of the group is based on the detection of F-formations. A F-formation arises whenever two or more people sustain a spatial and orientational relationship in which the space between them is one to which they have equal, direct, and exclusive access [4,32].

$$IS_{prox}^k = \left(\frac{1}{N} \sum_{j=1}^{N} \mathcal{F}(\|x_r^j - x_f^j\|_2 < \delta^k) \right) * 100 \tag{29}$$

where x_f^j determines the center of the closest formation or group of people f to the robot at step j.

5.2 Experiments

The social navigation behavior learned using the proposed learning algorithm RTIRL is evaluated here. A cost function using the features described in Sect. 3 has been learnt from navigation demonstrations in static scenarios. The demonstrations were recorded in the Robotics Lab and some scenarios of the TERESA project (an elderly care center in Troyes, France, see Fig. 1). The planning time for the RRT* employed during the learning phase was 3 s. The learned cost function is then used by the Social Navigation System.

Two sets of experiments have been performed. One in a static scenario (the learning is performed from static scenes) and another one in a dynamic scenario, to check the behavior in scenarios with moving pedestrians.

These evaluation experiments have been recorded in the Robotics Lab of the Pablo de Olavide University. The room has an approximate dimension of 4.60 × 7.60 m and it is equipped with an Optitrack Motion Capture System. This system has been used to accurately estimate the position of the robot and persons involved in the experiments so that results cannot be biased by localization errors.

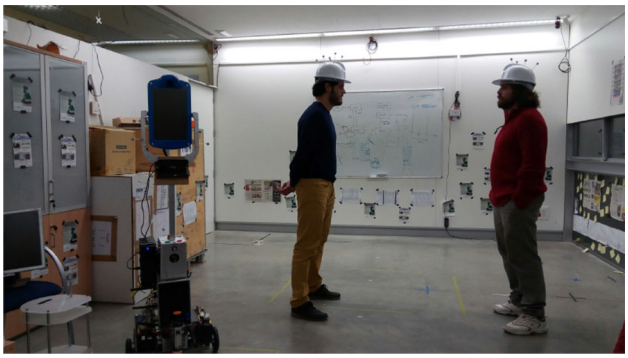

Fig. 9 Capture of the real experiments for the social navigation evaluation in the University Pablo de Olavide

5.2.1 Experiments in Static Scenarios

In the static scenario we have considered two different situations: first, a small group of two people talking to each other in the middle of the scene, as can be seen in Fig. 9; second, there are two persons in the scenario, but they are not interacting each other. Figure 10 shows two examples of the scenarios proposed, where the robot paths and the people position and orientation are shown. As can be seen, the Social Navigation System (SNS) avoid interfering with the group indicating a more acceptable behavior than the Non-Social Navigation System (Non-SNS) from the social point of view.

Tables 8 and 9 shows the metrics, with their standard deviations, obtained for four runs of each navigation system in the static scenario with a group of interacting people, and another four runs in the scenario with two non-interacting individuals respectively.

In the case of two people not forming a group, the metrics do not present many significant differences as can be observed in Table 8 (note that the IS_{prox} metric is not applicable because there is not interaction space). This is because of, in this particular setup, the social taught behaviors lead the robot to similar paths of the Non-SNS. Nevertheless, some metrics can be highlighted: the minimum distance to people is higher in the case of the SNS which indicates that the system always was able to keep some distance to the person; and the SNS was able to spend less time in the Personal space in favor of the time in the Social space which indicates a better social behavior.

Furthermore, the case of a group of people presents clearer differences. The Non-SNS performs clear intrusions in the Intimate space with respect to the closest person and also to the interaction space of the group. Again, the minimum distance to people is higher in the case of SNS. These results show that the SNS is more respectful of the people spaces and the interaction space of the group from a social point of view.

5.2.2 Experiments in Dynamic Scenarios

Another test has been recorded in a dynamic situation. The aim is to check whether the social cost function learned for static situations is still able to behave in a socially acceptable way in an environment with moving pedestrians. To do that, the RRT* re-plans at a frequency of 2 Hz. In this scenario, two people are walking freely in the scenario, sometimes forming a group and sometimes walking alone. At the same time, the robot is receiving waypoints that lead it to cross the room avoiding the people around. One long run for each navigation system was performed. Figure 11 shows the paths followed by the people in one of the runs (right) and the paths followed by the robot with the SNS and the Non-SNS (left). As can be seen, the SNS tried to deal with the people adapting its paths to the individuals in a social way unlike the Non-SNS, which tried to perform the same direct trajectory to the goal many times.

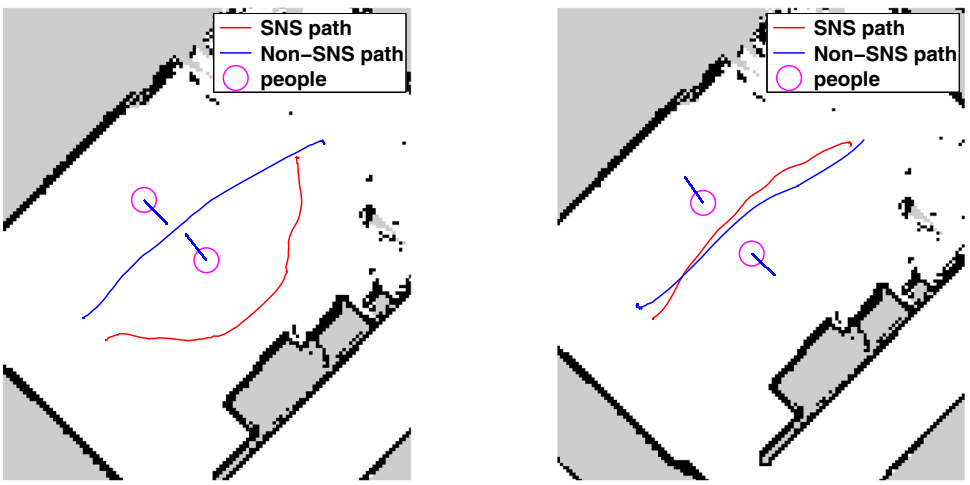

Fig. 10 Static scenario for social evaluation along with some of the paths obtained by the SNS and non-SNS. Left: Two persons, represented by magenta circles with the blue arrow indicating their orientation, are forming a group. Right: Two persons are facing the walls (not a group). The red line shows the path followed by the TERESA social navigation system (SNS). The blue line shows the path of the move_base non-social system (Non-SNS)

Table 8 Results for social navigation evaluation with a static scenario with two people

No Group	$T_p(s.)$	$L_p(m.)$	$CHC(rads.)$	$CP_{avg}(m.)$	$CP_{min}(m.)$	$CP_{max}(m.)$
SNS	22.70 ± 1.52	4.86 ± 0.40	4.54 ± 0.79	1.74 ± 0.09	1.08 ± 0.16	2.47 ± 0.13
Non-SNS	24.18 ± 1.73	4.92 ± 0.23	4.37 ± 0.40	1.90 ± 0.14	0.63 ± 0.07	2.82 ± 0.01
	$CP_{prox}(\%)$					
	Intimate			Personal		Social+
SNS	0.00 ± 0.00			12.77 ± 1.02		87.23 ± 0.87
Non-SNS	0.00 ± 0.00			17.24 ± 0.98		82.76 ± 0.98

Table 9 Results for social navigation evaluation with a static group of two people

Group	$T_p(s.)$	$L_p(m.)$	$CHC(rads.)$	$CP_{avg}(m.)$	$CP_{min}(m.)$	$CP_{max}(m.)$
SNS	23.65 ± 0.64	5.45 ± 0.27	4.18 ± 2.55	1.87 ± 0.08	0.79 ± 0.17	2.71 ± 0.11
Non-SNS	23.15 ± 1.34	5.02 ± 0.08	4.41 ± 0.08	1.88 ± 0.12	0.42 ± 0.15	2.74 ± 0.17
	$CP_{prox}(\%)$			$IS_{prox}(\%)$		
	Intimate	Personal	Social+	Intimate	Personal	Social+
SNS	0.00 ± 0.00	17.83 ± 1.52	82.17 ± 1.52	0.00 ± 0.00	0.00 ± 0.00	100.00 ± 0.00
Non-SNS	2.60 ± 3.67	14.50 ± 3.06	82.90 ± 0.61	7.64 ± 0.70	13.00 ± 0.01	79.37 ± 0.71

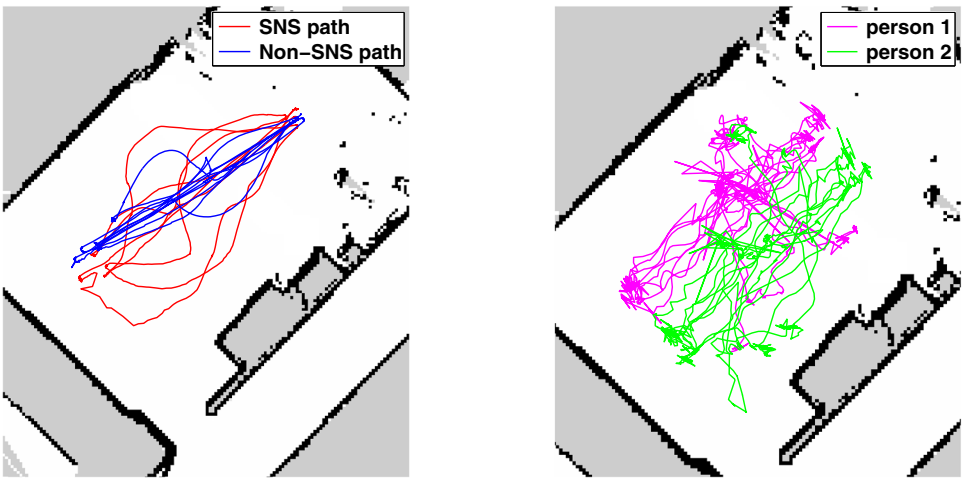

Fig. 11 Representation of the paths followed in the dynamic free run experiment. Left: paths followed by the robot using the SNS (red) and Non-SNS (blue). Right: Paths followed by the two people in the scenario

Table 10 Results for social navigation evaluation of a free run with two people moving in the scenario

	$T_p(s.)$	$L_p(m.)$	$CHC(rads.)$	$CP_{avg}(m.)$	$CP_{min}(m.)$	$CP_{max}(m.)$
SNS	495.50	70.24	89.16	2.50	**0.31**	4.65
Non-SNS	387.80	50.18	74.34	2.50	0.27	4.84
	$CP_{prox}(\%)$			$IS_{prox}(\%)$		
	Intimate	Personal	Social+	Intimate	Personal	Social+
SNS	0.60	14.31	85.08	3.31	12.27	84.42
Non-SNS	3.17	10.11	86.73	8.51	12.67	78.81

Table 10 shows the metrics obtained in the experiment. In this case, the total time T_p indicates the total duration of the run being the recording of the SNS case a bit longer than the Non-SNS recording. Under this condition, the path length metric L_p indicates the total length traveled by the robot (all the trajectories), and, as can be seen, it was longer in the

SNS case. This fits with the paths shown in Fig. 11 left, and explains the higher value in the cumulative heading changes (CHC) since the SNS tried to adapt its paths to the people in the scene. Again the most relevant and significant metrics are the intrusions in the Proxemics spaces of the closest person (CP_{prox}) and the interaction area (IS_{prox}). The intrusions in the intimate spaces are much lower in the case of SNS, what states the effort on minimizing the disturbance of people. As a general result, we can say that the behavior of the SNS is more socially acceptable than the behavior showed by the Non-SNS as expected.

6 Conclusions

This paper presented an approach for teaching a robot social navigation behaviors using demonstrations. To this end, a method based on IRL has been implemented and linked with a regular RRT* 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 typical problems associated to IRLs based on MDPs. The proposed method allows to deal with continuous state spaces and is capable of dealing with larger scenarios (in terms of persons and features) than previous discrete-MDP-based approaches for human-aware navigation [28]. It also simplifies the generalization of the behavior thanks to the inherent benefits of RRTs. The approach has been validated using demonstrations provided from a ground-truth cost function. The results show that it is able to properly approximate the demonstrated cost function, and to obtain behaviors more similar to the demonstrations than related algorithms from the state-of-art. Furthermore, Sect. 5 has shown how the taught behaviors lead to more socially appropriate motion than methods not considering the learned costs.

The proposed method has been applied to a telepresence robot. While the learning approach is offline, this setup allows to gather additional data whenever the user employs the robot in teleoperated mode, and thus to update the learned costs iteratively when new data is available, considering new scenarios.

Future work will consider including kinodynamic constraints 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. Additionally, we will also evaluate learning the relevant features from the demonstrations. For that, recent approaches of deep learning for IRL [6] will be considered. Finally, 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.

Funding this study was funded by the EC-FP7 under grant agreement no. 611153 (TERESA).

Compliance with Ethical Standards

Conflict of Interest The authors declare that they have no conflict of interest.

References

1. Abbeel P, Ng AY (2004) Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the twenty-first international conference on Machine learning, ICML '04. ACM, New York, NY, USA, p 1. https://doi.org/10.1145/1015330.1015430
2. Argali B, Chernova S, Veloso M, Browning B (2009) A survey of robot learning from demonstrations. Robot Auton Syst 57:469–483
3. Borgefors G (1986) Distance transformations in digital images. Comput. Vision Graph. Image Process. 34(3):344–371. https://doi.org/10.1016/S0734-189X(86)80047-0
4. Cristani M, Bazzani L, Paggetti G, Fossati A, Tosato D, Bue AD, Menegaz G, Murino V (2011) Andrea Fossati, Del˜Bue, A.: social interaction discovery by statistical analysis of F-formations. In: British machine vision conference (BMVC), pp 23.1–23.12. https://doi.org/10.5244/C.25.23
5. Feil-Seifer D, Mataric M (2011) People-aware navigation for goal-oriented behavior involving a human partner. In: Proceedings of the IEEE international conference on development and learning (ICDL)
6. Finn C, Levine S, Abbeel P (2016) Guided cost learning: Deep inverse optimal control via policy optimization. In: Proceedings of the 33rd international conference on machine learning, vol 48
7. Fox D, Burgard W, Thrun S (1997) The dynamic window approach to collision avoidance. IEEE Robot Autom 4(1):23
8. Hall ET (1990) The hidden dimension. Anchor
9. Henry P, Vollmer C, Ferris B, Fox D (2010) Learning to navigate through crowded environments. In: Proceedings of the IEEE international conference on robotics and automation (ICRA), pp 981–986
10. Karaman S, Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. Int J Robot Res 30(7):846–894
11. Kirby R, Forlizzi J, Simmons R (2010) Affective social robots. Robot Auton Syst 58:322–332
12. Kirby R, Simmons RG, Forlizzi J (2009) Companion: a constraint-optimizing method for person-acceptable navigation. In: RO-MAN, pp 607–612. IEEE
13. Kretzschmar H, Kuderer M, Burgard W (2014) Learning to predict trajectories of cooperatively navigating agents. In: 2014 IEEE international conference on robotics and automation (ICRA) , pp 4015–4020. IEEE
14. Kretzschmar H, Spies M, Sprunk C, Burgard W (2016) Socially compliant mobile robot navigation via inverse reinforcement learning. Int J Robot Res. https://doi.org/10.1177/0278364915619772
15. Kruse T, Pandey AK, Alami R, Kirsch A (2013) Human-aware robot navigation: a survey. Robot Auton Syst 61(12):1726–1743. https://doi.org/10.1016/j.robot.2013.05.007
16. Kuderer M, Gulati S, Burgard W (2015) Learning driving styles for autonomous vehicles from demonstration. In: Proceedings of the IEEE international conference on robotics & automation (ICRA), Seattle, USA, vol 134
17. Kuderer M, Kretzschmar H, Sprunk C, Burgard W (2012) Feature-based prediction of trajectories for socially compliant navigation. In: Proceedings of robotics: science and systems (RSS). Syd-

ney, Australia. http://www.informatik.uni-freiburg.de/~kudererm/publications/kuderer12rss.pdf
18. Levine S, Koltun V (2012) Continuous inverse optimal control with locally optimal examples. In: ICML '12: Proceedings of the 29th international conference on machine learning
19. Luber M, Spinello L, Silva J, Arras KO (2012) Socially-aware robot navigation: A learning approach. In: 2012 IEEE/RSJ international conference on intelligent robots and systems, pp 902–907. https://doi.org/10.1109/IROS.2012.6385716
20. Michini B, Cutler M, How JP (2013) Scalable reward learning from demonstration. In: IEEE international conference on robotics and automation (ICRA). IEEE. http://acl.mit.edu/papers/michini-icra-2013.pdf
21. Ng AY, Russell SJ (2000) Algorithms for inverse reinforcement learning. In: Proceedings of the seventeenth international conference on machine learning, ICML '00. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 663–670. http://dl.acm.org/citation.cfm?id=645529.657801
22. Okal B, Arras KO (2016)Formalizing normative robot behavior. In: Social robotics: 8th international conference, ICSR 2016, Kansas City, MO, USA, November 1-3, 2016 Proceedings. Springer International Publishing, pp 62–71. https://doi.org/10.1007/978-3-319-47437-3_7
23. Okal B, Arras KO (2016) Learning socially normative robot navigation behaviors with bayesian inverse reinforcement learning. In: Proceedings of the IEEE international conference on robotics and automation, ICRA , Stockholm, Sweden, pp 2889–2895. https://doi.org/10.1109/ICRA.2016.7487452
24. Pacchierotti E, Christensen H, Jensfelt P (2006) Evaluation of passing distance for social robots. In: IEEE workshop on robot and human interactive communication (ROMAN). Hartfordshire, UK
25. Pérez-Higueras N, Caballero F, Merino L (2016) Learning robot navigation behaviors by demonstration using a RRT* planner. In: International conference on social robotics. Springer International Publishing, pp 1–10
26. Quigley M, Conley K, Gerkey BP, Faust J, Foote T, Leibs J, Wheeler R, Ng AY (2009) Ros: an open-source robot operating system. In: ICRA workshop on open source software
27. Ramachandran D, Amir E (2007) Bayesian inverse reinforcement learning. In: Proceedings of the 20th international joint conference on artical intelligence vol 51, pp 2586–2591. http://www.aaai.org/Papers/IJCAI/2007/IJCAI07-416.pdf
28. Ramón-Vigo R, Pérez-Higueras N, Caballero F, Merino L (2014) Transferring human navigation behaviors into a robot local planner. In: Proceedings of the IEEE international symposium on robot and human interactive communication, RO-MAN. https://doi.org/10.1109/ROMAN.2014.6926347
29. Ramon-Vigo R, Perez-Higueras N, Caballero F, Merino L (2015) Analyzing the relevance of features for a social navigation task. In: L.P. Reis, A.P. Moreira, P.U. Lima, L. Montano, V. Munoz-Martinez (eds.) Robot 2015: Second Iberian Robotics Conference, *Advances in Intelligent Systems and Computing*, vol. 418, pp. 235–246. Springer International Publishing. https://doi.org/10.1007/978-3-319-27149-1_19
30. Ratliff ND, Bagnell JA, Zinkevich Ma (2006) Maximum margin planning. In: International conference on machine learning—ICML '06(23), pp 729–736. https://doi.org/10.1145/1143844.1143936
31. Ratliff ND, Silver D, Bagnell JA (2009) Learning to search: functional gradient techniques for imitation learning. Auton Robots 27(1):25–53. https://doi.org/10.1007/s10514-009-9121-3
32. Setti F, Russell C, Bassetti C, Cristani M (2015) F-formation detection: individuating free-standing conversational groups in images. PLoS ONE 10(5):1–32. https://doi.org/10.1371/journal.pone.0123783
33. 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 (2015) Teresa: a socially intelligent semi-autonomous telepresence system. In: Workshop on machine learning for social robotics at ICRA-2015 in Seattle
34. Shiarlis K, Messias J, Whiteson S (2017) Rapidly exploring learning trees. In: Proceedings of the IEEE international conference on robotics and automation (ICRA). IEEE, Singapore, Singapore
35. Shiarlis K, Messias J, Whiteson S (2017) Acquiring social interaction behaviours for telepresence robots via deep learning from demonstration. In: Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS). Vancouver, Canada
36. Sisbot EA, Marin-Urias LF, Alami R, Siméon T (2007) A human aware mobile robot motion planner. IEEE Trans Robot 23(5):874–883
37. Vasquez D, Okal B, Arras KO (2014) Inverse reinforcement learning algorithms and features for robot navigation in crowds: an experimental comparison. In: Proceedings IEEE/RSJ international conference on Intelligent Robots and Systems (IROS), pp 1341–1346. https://doi.org/10.1109/IROS.2014.6942731
38. Xia C, Kamel AE (2016) Neural inverse reinforcement learning in autonomous navigation. Robot Auton Syst 84:1–14. https://doi.org/10.1016/j.robot.2016.06.003
39. Ziebart B, Maas A, Bagnell J, Dey A (2008) Maximum entropy inverse reinforcement learning. In: Proceedings of the national conference on artificial intelligence (AAAI)

Noé Pérez-Higueras received his Computing Engineer Degree in 2009 and a Master Degree in Industrial Computing in 2010 both from the University of Almería, Spain. After 3 years collaborating with the Department of Languages and Computing of the University of Almería, he started as Ph.D. Candidate in Robotics in the Pablo de Olavide University, Seville, Spain. His research interests are robot navigation and planning and social robotics.

Fernando Caballero is Associate Professor at the Department of System Engineering and Automation, University of Seville, Seville, Spain. He received his B.S. in Telecommunication Engineering in 2003 and his Ph.D. in Robotics in 2007 both from the University of Seville, Spain. His research focuses in robot localization, mapping and navigation in real setings. He has participated in numerous National and European funded research projects involving aerial and ground robots.

Luis Merino is Associate Professor at the School of Engineering, Pablo de Olavide University, Seville, Spain. He received the Telecommunications Engineer Degree in 2000 from the University of Seville. In 2007 he received a Ph.D. degree on Robotics from the University of Seville. His thesis has been awarded with the ABB Award to the Best Doctoral Dissertation on Robotics 2007, given by the Spanish Committee of Automation (CEA, Robotics Group). His research interests include robot localization and navigation, multi-robot systems, sensor fusion and planning under uncertainties applied to robotics, and he leads or has led several EU projects on social navigation.