Int J of Soc Robotics (2016) 8:51–66
DOI 10.1007/s12369-015-0310-2

Socially Adaptive Path Planning in Human Environments Using Inverse Reinforcement Learning

Beomjoon Kim[1] · **Joelle Pineau**[1]

Accepted: 6 June 2015 / Published online: 20 June 2015

Abstract A key skill for mobile robots is the ability to navigate efficiently through their environment. In the case of social or assistive robots, this involves navigating through human crowds. Typical performance criteria, such as reaching the goal using the shortest path, are not appropriate in such environments, where it is more important for the robot to move in a socially adaptive manner such as respecting comfort zones of the pedestrians. We propose a framework for socially adaptive path planning in dynamic environments, by generating human-like path trajectory. Our framework consists of three modules: a feature extraction module, inverse reinforcement learning (IRL) module, and a path planning module. The feature extraction module extracts features necessary to characterize the state information, such as density and velocity of surrounding obstacles, from a RGB-depth sensor. The inverse reinforcement learning module uses a set of demonstration trajectories generated by an expert to learn the expert's behaviour when faced with different state features, and represent it as a cost function that respects social variables. Finally, the planning module integrates a three-layer architecture, where a global path is optimized according to a classical shortest-path objective using a global map known *a priori*, a local path is planned over a shorter distance using the features extracted from a RGB-D sensor and the cost function inferred from IRL module, and a low-level system handles avoidance of immediate obstacles. We evaluate our approach by deploying it on a real robotic wheelchair platform in various scenarios, and comparing the robot trajectories to human trajectories.

✉ Beomjoon Kim
beomjoon.kim0@gmail.com

Joelle Pineau
jpineau@cs.mcgill.ca

[1] School of Computer Science, McGill University, 3480 University, Montreal, Canada

Keywords Navigation · Obstacle avoidance · RGB-D optical flow · Learning from demonstration · Inverse reinforcement learning

1 Introduction

The ability to navigate in a crowded and dynamic environment is crucial for robots employed in indoor environments such as shopping malls, airports or schools. When navigating in such environments, it is important for a robot to not only avoid obstacles and move towards its goal, but also to do so in a socially adaptive manner. Such behaviour is essential in assistive robots, as they interact with humans on a daily basis. The goal of this research is to propose a local path planner that generates such socially adaptive navigation behaviour, integrated with a feature extraction method and a planning architecture that combines global, local, and low level behaviours.

A major limitation of the some of standard methods on local path planners such as [13,34] is that they treat the pedestrians merely as obstacles to be avoided, and do not take account of social variables for navigation behaviours. As such, the main technical challenge for developing a socially adaptive path planner is the fact that the notion of what is socially acceptable is not easily defined as an optimization criteria, thus we cannot apply standard search techniques using conventional cost functions (e.g. shortest path) to find a good path.

In light of such challenge, we propose a local path planner that navigates through pedestrians by learning a socially adaptive cost function from human navigation demonstrations. Our intuition is that while manually defining a socially adaptive cost function is very challenging, inferring such a cost function from a human, who is instinctively aware of socially adaptive navigation behaviours, is an easier task. To learn the cost function, we employ the inverse reinforcement learning (IRL) technique [24], which has proven effective in inferring cost functions from human demonstrations to accomplish tasks such as car parking [1], helicopter piloting [2], and driving [42]. We also introduce the use of a regularization method in IRL to eliminate unnecessary features.

Our proposed framework consists of three modules: the feature extraction module, the IRL module, and the planning module. The feature extraction module extracts the local state features such as densities and velocities of pedestrians observed via RGB-D sensor during a navigation. The IRL module uses demonstrations from a human operator to learn the socially adaptive navigation behaviours as a function of these observed state features. The behaviour is represented as weights (i.e. parameters) for the cost function, which are combined with the features extracted from the feature extraction module to compute a socially adaptive cost function that defines costs at local grid cells. In the planning module, a global path is planned using a global map given *a priori* and a local path is planned using the local grid cells associated with the costs. It then chooses which of these two plans to use, depending on the situation.

We evaluate our approach by considering different navigation scenarios arising from the deployment of a smart robotic wheelchair. We consider evaluation metrics pertinent to socially adaptive navigation such as distance to nearest pedestrian, overall path length, etc. We compare values of these metrics under three control modes: our IRL approach using RGB-D sensing, conventional control mode with a human-controlled joystick, and more traditional path planning via the dynamic window approach (DWA) [13] using laser rangefinder sensing. Our results show that the IRL approach is able to generate trajectories that are closer to the human driver, compared to the DWA planner, even when the training data was acquired under different conditions than those observed during the evaluation. This is a particularly attractive property of our approach, and is achieved because the cost function is based on dynamic features of the environment (e.g. speed, direction of surrounding pedestrians), rather than on static features of the map.

2 Problem Statement

We state the problem addressed in our paper as follows. *Given* a global path plan from the current position of the robot to the global goal, *determine* the socially adaptive local path plan from the current position of the robot to the sub-goal.

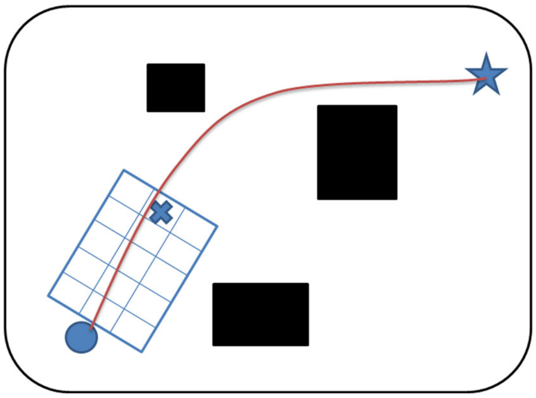

Fig. 1 An example of a typical planning scenario. The *circle* represents the robot, and the *grid cells* represents the local grid cells defined by the viewing distance and field of view of the RGB-D sensor. *Black boxes* represent the obstacles, and the *star* represents the global goal. The *red line* represents the global path. X denotes the sub-goal. Given the global path, our local path plans a socially adaptive path when the local grid cells are filled with people and their flows. (Color figure online)

A sub-goal is defined as the intersection of a global path plan and a local grid cell at the maximum viewing distance and field of view of the RGB-D sensor. This is described in detailed in Fig. 1.

Addressing this problem involves challenges such as developing a feature extraction method, learning a socially adaptive cost function, and integrating the local path planner with a three-layer planning architecture similar to that of [14].

As it is difficult to objectively define a socially adaptive path plan, we consider a path plan to be socially adaptive if it resembles the one that is planned by a human under similar navigation conditions. Our work is motivated by the goal of deploying our intelligent wheelchair in a large indoor shopping centre, where autonomous navigation capabilities can be used to navigate between stores and other points of interest without requiring substantial human intervention, even in crowded situations.

2.1 Related Work

There is a number of work on navigation in human environment [12,15,18,19,31,33,37,41,43]. These are largely divided into four sub-groups: those that involve training a motion predictor [12,43], those that focus on path-planning while assuming motions of dynamic obstacles are given [31,37], those that focus on human-aware navigation using social variables [18,19,29,32,33] with hand-designed cost function, those that uses pedestrian models [29,32] and lastly, one work on learning the cost function [15] for global path planning.

In [12], a path planner based on Polynomial Neural Network for human motion prediction is proposed. They trained the motion predictor using human motion data, which is used with the Robot Navigation-Hierarchical Partially Observable Markov Decision Process (RN-HPOMDP) to plan a path. Another work that uses training data to predict the motions of pedestrians is [43], where maximum entropy inverse optimal control is used to train the motion predictor. Clearly, the drawback of such motion learning approaches is that if people move in a significantly different manner than the training data, then the motion predictor cannot generalize to such situations. More formally, this is due to the fact that the training data and test data are non-identically and independently distributed (non-iid), whereas the iid is standard assumption in supervised learning approaches. Unfortunately, for the human motion prediction, different environments will induce non-iid data.

In [37], a path planner based on Rapidly-exploring Random Trees (RRT) algorithm is proposed. The key insight here is to rapidly plan a new path so as to avoid unforeseen pedestrians, by executing only a small segment of initially-planned trajectory while planning a new trajectory. While in simulation, where noise-free pedestrian velocity is given, it worked well, the authors did not attempt to extend their work to real-life scenarios where velocity estimation is required. Similarly in [31], a modification of DWA [13] that accommodates for dynamic obstacles is proposed. However, there is an additional difficulty of estimating angular velocities of dynamic obstacles, which alone is a difficult computer vision problem.

In [18], the authors rightly points out that a path planner should not simply guarantee a collision-free path, but should also generate behaviour adhering to social rules for human comforts. In support of this, they present a modification of human-aware navigation planner [33], that considers humans as static obstacles, for dynamic environments. Similarly in [29,32], authors propose models that induces socially aware navigation behaviors using pedestrian model and social force model, respectively. For a comprehensive survey on human-aware navigation, refer to [19]. An important distinction between these works and ours is that our framework learns the socially-adaptive cost function, while these works carefully hand-design it.

In [29,32], they specifically focus on social components such as pedestrian body pose and face orientation for navigation. They manually design pedestrian models that incorporate such components, and consider them during navigation. In contrast, our work attempts to learn social models in the form of a cost function, and incorporate them into path planning.

The closest works are [15]. In [15], human-like path planning in crowded situations is learned from an expert. However, their work is also limited to a simulated setting and they focused on a global path planner that plans based on the modelling of density and velocity of people using Gaussian processes. In contrast, our work focuses on a local path planner integrated with a feature extraction module that extracts pedestrian movement, as well as the full planning architecture that provides interactions with a global planner.

2.2 Challenges

There are a few challenges in developing a socially adaptive local path planner. The first challenge is in local feature extraction, where the objective is to extract densities and velocities of crowds around the robot. Previously, there has been a number of attempts at tracking and predicting motions of dynamic objects using filtering methods [23], hidden Markov models [5,30] or polynomial neural network [12]. In our work, rather than tracking individuals in the crowd, we propose to extract summary features using a RGB-D sensor. We represent the future positions of crowds using the velocity features, which consists of pedestrian speed and direction, and the current position of crowds using the density feature. Our feature extraction module is fully online and does not require any training. Note that in some of the earlier work on navigation in dynamic environments, feature extraction was not dealt with as these works limited their experiments to simulations where the features are assumed to be given [15,37]. Incorporating accurate feature extraction proved to be one of the major challenges of this work. The details of feature extraction and its representation are discussed in Sect. 3.

The second challenge lies in learning the socially adaptive cost function for local path planner with IRL technique. We show how to design a real-world navigation problem as a Markov decision process, how to represent navigation features, and how to define the feature function and cost function for different sections of local area. These are discussed in detail in Sect. 4.

The third challenge lies in developing the planning architecture. We had to address issues such as integration of local path planner with a global path planner, determining when to use the local or global planner, when to re-plan and when to emergency stop. To address these issues, we develop a three-layer planning architecture similar to that of [14] in which all layers run in parallel to resolve the issues mentioned. The architecture design is discussed in detail in Sect. 5.

Last but not least, there is a challenge in the evaluation of our framework. To assess our approach, it is essential that we compare executed trajectories of human driver and the robot under the same or similar conditions, otherwise the trajectories will differ. The problem is that in an uncontrolled environment, it is almost impossible to create the same navigation situation because pedestrians often move in an unpredictable manner. To address this issue and reduce variance in our real-world evaluations, we first consider scenario-based experiments, in which the robot and the human driver both execute trajectories under similar initial

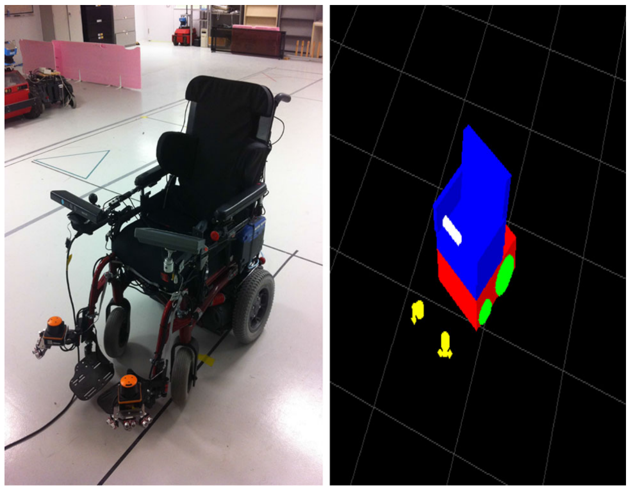

Fig. 2 The Smartwheeler robotic wheelchair (*left*), and its virtual representation (*right*)

conditions. We consider a variety of initial conditions and scenarios. We then present results of deploying our socially adaptive path planning approach in natural navigation conditions, collected by running the robot in the busy hallways of a building during normal business hours.

2.3 Robot Platform

We developed the hardware and software infrastructure to assess our framework in real-world assistive navigation scenarios. The hardware platform used is a robotic wheelchair [25], shown in Fig. 2(left), which is a modified power wheelchair mounted with an RGB-D (Kinect) sensor at the front, three Hokoyu laser range finders (2 at the front, 1 in the back), and a laptop. For the work described here, we use only the RGB-D sensor to observe the surroundings in an attempt to investigate our ability to generate effective navigation with low-cost sensing. As shown in Fig. 2, the Kinect RGB-D sensor is placed on one of the handles on the wheelchair, marked with white on the virtual representation. The sensor provides both RGB and depth images at a resolution of 640×480. It has a horizontal field of view of $57°$ and approximately 5m in depth. Conveniently, the Kinect combines RGB and depth information, and represents them as a point cloud in 3D coordinates (x, y, z), in which horizontal, vertical, and depth locations are respectively represented.

Our framework was implemented as packages in the popular robot software platform, ROS [27]. When we plan a local path, we represent it as a set of points to be traversed, and send them to a navigation package in ROS, which then calculates the velocity to be executed by the robot to reach each waypoint. All our results and experiments are performed using the wheelchair robot and ROS. It is worth noting that our framework is particularly useful in wheelchair robots because human control trajectories for this platform are easy to gather (using the conventional joystick) and abundant (assuming regular use of the platform by a mobility-impaired individual).

2.4 Outline of Our Approach

Our approach goes through three stages, consisting of two off-line stages and the on-line execution stage. These stages are summarized in Fig. 3. In the data gathering stage, a human shows demonstrations by navigating in an environment. Throughout the demonstrations, we collect the state features occurring in the trajectories of the demonstrator, as well as the demonstrator's action choices as a function of these state features. The collection of these pairs of state and action forms a demonstration dataset. During the learning stage, we pass this demonstration dataset to the IRL module to learn the navigation behaviour of the demonstrator, which is succinctly represented as weights for the cost function. Finally in the on-line execution stage, the weights are passed to the planning module. The feature extraction module extracts features at the local area using the RGB-D sensor. The features are represented as a set of feature vectors, where each feature vector describes the feature values in different parts of the local area. The event-driven planning module waits for these feature vectors from the feature extraction module. Once the feature vectors arrive, the planning module combines the learned weights and feature vectors to compute the costs. Using these costs, the planning module outputs a path plan such that the cost is minimized, and the robot executes this plan in the environment. As the robot executes the plan, the feature extraction module continues to receive new sensor data, and new feature vectors are again passed to the planning module that is waiting for them. The cycle continues until the robot reaches the goal, or is manually stopped by the operator.

As in traditional grid-cell based local path planners [20], our approach associates a numerical cost to each grid cell. However, unlike these planners that use occupancy grids to calculate costs, our planner uses environmental features that all contribute to the cost of a cell. During runtime, the feature extraction module takes RGB-D inputs and extracts environmental features at each of the cells, thus producing a set of feature vectors where each feature vector contains the feature values of a cell.

We would like to note that the demonstration data gathered off-line can be collected in a different environment than the environment in the execution stage, unlike . Due to the nature of the IRL framework, the cost function is expected to generalize well to different environments with similar pedestrian patterns. This is explained in depth in Sect. 4.2.5.

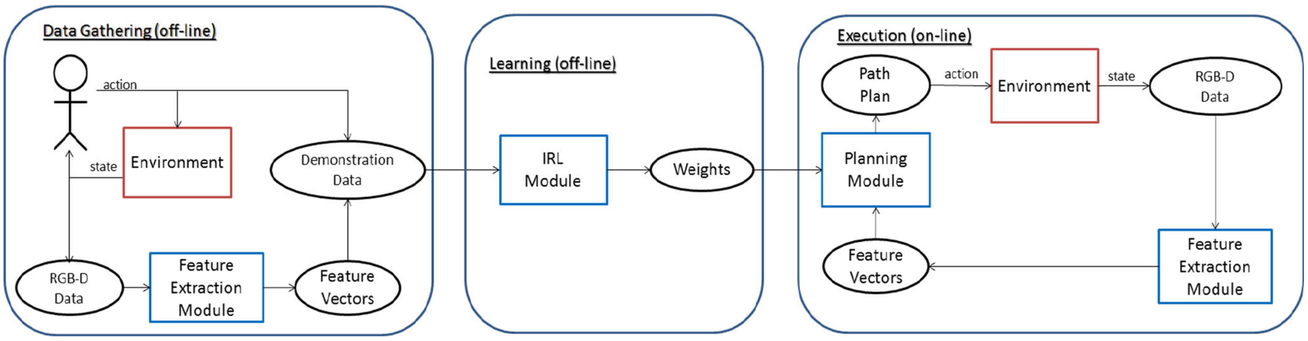

Fig. 3 Different stages of our approach and module interaction during the execution stage. *Square blue boxes* indicate the three modules. The human demonstrator is denoted with the stick figure. *Arrows* indicate the information flow. (Color figure online)

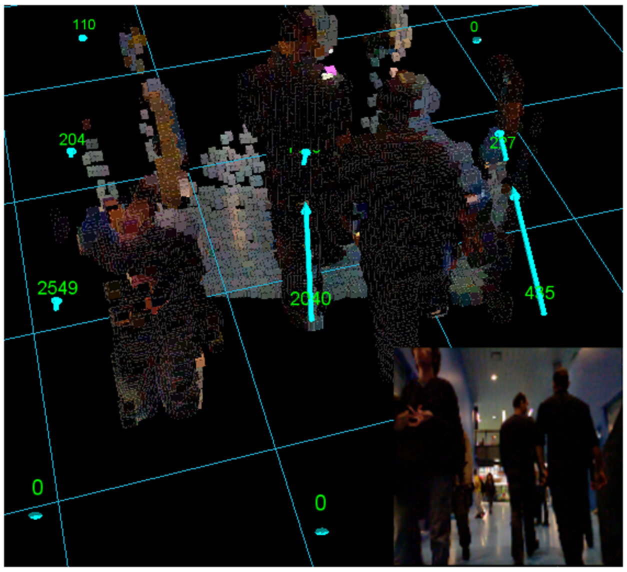

Fig. 4 Example of feature extraction and its corresponding RGB image (bottom-right corner)

3 Feature Extraction

The RGB-D sensor uses a 3D point cloud to represent the observed scene. We use this information to extract the features. In the current work, we consider four distinct features to characterize dynamic aspects of the scene: crowd density, speed, velocity (i.e. speed and direction of a crowd), and distance to the goal. The density feature estimates where the crowds currently are by counting the number of points in each cell. The speed and direction features estimate where the objects will be in the future, by estimating the velocity of each point in a point cloud. The velocity of each cell is calculated by averaging the velocities of points in the cell. The distance to the goal is calculated based on the distance between each cell and the global goal. An example of feature extraction in a crowded environment is shown in Fig. 4.

While estimating the density and distance features is trivial, calculating velocities using 3D point clouds is more complicated. In the field of computer vision, this problem is known as *optical flow* [16], in which the problem is limited to calculating the movement of pixels among RGB image sequences. We now introduce an algorithm that extends an RGB optical flow algorithm to the RGB-D setting.

3.1 RGB-D Optical Flow

In essence, the problem of determining the flow using 3D point clouds is finding the correspondence between each point at frame t to a point at frame $t+1$. Once this problem is solved, the velocity can be simply inferred by subtracting the coordinates of corresponding points. While it is difficult to determine the correspondence using only the point cloud coordinate information, the problem of finding the correspondence, or an optical flow, is a well studied area in RGB images. Thus we first estimate RGB optical flow, and then use the flow information to infer the RGB-D optical flow.

3.2 Farneback Optical Flow

Various RGB optical flow algorithms have been examined in the last three decades [6,16,35]. In robot navigation, however, the computational cost for flow estimation is crucial as it determines how quickly a robot can respond to changes in the environment. Hence we employ an RGB optical flow algorithm known as Farneback optical flow, which entails a fast flow estimation algorithm [11].

The Farneback optical flow algorithm is a tensor based algorithm in which an estimated orientation tensor is used to infer a velocity [17]. Imagine several image frames stacked together. What we obtain is a 3D image volume in which we have two spatial dimensions, and a third describing the temporal dimension. Intuitively, a movement (i.e. translation of a pixel) in the sequences of images would give a vector with a particular orientation in this 3D image volume, describing the velocity direction. Such orientation can be represented using the orientation tensor [17], which is a 3×3 symmetric positive semidefinite matrix $\mathbf{T}$ for the case of 3D image volume. The quadratic form of the orientation tensor, $\hat{\mathbf{u}}^T \mathbf{T} \hat{\mathbf{u}}$, is a measure of how much the image varies in the direction given by the vector $\hat{\mathbf{u}}$.

In [10], a method for estimating the orientation tensor $\mathbf{T}$ is proposed. The idea is to project the image signal into a second degree polynomial, whose parameters are estimated using least-squares estimation. Using these parameters, the orientation tensor matrix can be approximated. The details of orientation tensor estimation are in [10]. Clearly, once the orientation tensor is estimated, then we can treat the vector $\hat{\mathbf{u}}$ as a velocity vector. Hence we estimate the velocity by solving the following equation, where we denote $\hat{\mathbf{v}}$ for our velocity vector:

$$\hat{\mathbf{v}}^T \mathbf{T} \hat{\mathbf{v}} = 0 \tag{1}$$

$$\mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix}, \hat{\mathbf{v}} = \frac{\mathbf{v}}{||\mathbf{v}||}. \tag{2}$$

The intuition in Eq. (1) is that the image intensity in the direction of $\hat{\mathbf{v}}$ should remain the same, as it is the same pixel moving according to the direction given by $\hat{\mathbf{v}}$. In practice, we cannot require $\hat{\mathbf{v}}^T \mathbf{T} \hat{\mathbf{v}}$ to be zero, due to the aperture problem that only allows us to detect the velocity that is normal to the surface, and subsequently causes $\mathbf{T}$ to have two zero eigenvalues. However, it is sufficient to minimize the expression [11].

Instead of estimating the velocity of each pixel, we assume the velocity field occurs across a region in an image according to an affine model [11]:

$$v_x(x, y) = ax + by + c \tag{3}$$

$$v_y(x, y) = dx + ey + f \tag{4}$$

where x and y are image coordinates. Now Eq. (2) can be represented as:

$$\hat{\mathbf{v}} = \mathbf{S}\mathbf{p} \tag{5}$$

$$\mathbf{S} = \begin{pmatrix} x & y & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{6}$$

$$\mathbf{p} = \begin{pmatrix} a & b & c & d & e & f & 1 \end{pmatrix}^T \tag{7}$$

To estimate $\mathbf{p}$, a cost function is minimized:

$$d(\hat{\mathbf{v}}, \hat{\mathbf{T}}) = \hat{\mathbf{v}}^T \mathbf{T} \hat{\mathbf{v}} \tag{8}$$

$$= d_{tot} = \sum_i d(\hat{\mathbf{v}}_\mathbf{i}, \hat{\mathbf{T}}_\mathbf{i}) \tag{9}$$

where the summation is over all points of the region. Combining Eq. (9) with Eq. (5), we have

$$d_{tot}(\mathbf{p}) = \sum_i d(\hat{\mathbf{v}}_\mathbf{i}, \hat{\mathbf{T}}_\mathbf{i}) = \sum_i \mathbf{p}^T \mathbf{S}_i^T \mathbf{T}_i \mathbf{S}_i \mathbf{p} \tag{10}$$

$$= \mathbf{p}^T (\sum_i \mathbf{Q}_i) \mathbf{p} = \mathbf{p}^T \mathbf{Q}_{tot} \mathbf{p} \tag{11}$$

where $\mathbf{Q}_i = \mathbf{S}_i^T \mathbf{T}_i \mathbf{S}_i$. We require that Eq. (11) be minimized with the constraint that the last element of $\mathbf{p}$ is 1. We partition the matrices and vectors to achieve a closed form solution:

$$\mathbf{p} = \begin{pmatrix} \bar{\mathbf{p}} \\ 1 \end{pmatrix}, \mathbf{Q}_{tot} = \begin{pmatrix} \bar{\mathbf{Q}} & \mathbf{q} \\ \mathbf{q}^T & \alpha \end{pmatrix} \tag{12}$$

$$d_{tot}(\mathbf{p}) = \bar{\mathbf{p}}^T \bar{\mathbf{Q}} \bar{\mathbf{p}} + \bar{\mathbf{p}}^T \mathbf{q} + \mathbf{q}^T \bar{\mathbf{p}} + \alpha. \tag{13}$$

Eq. (13) is minimized when

$$\bar{\mathbf{p}} = -\bar{\mathbf{Q}}^{-1} \mathbf{q}, \tag{14}$$

which gives us the velocity vector $\mathbf{v}$ if substituted to Eq. (5).

3.3 RGB-D Optical Flow Algorithm

As mentioned above, we solve the correspondence problem by RGB optical flow algorithm using sequences of RGB images, then map the pixels in these images to the point clouds. Conveniently, ROS provides an API in which the index of a particular pixel in the pixel vector for an image frame corresponds to the point with the same index in the point-cloud vector. This is represented as *MapPixelToPtCloud* function. Since the velocity information is extracted on a per-point basis, we average the velocities of points in each cell to calculate the velocity of that cell. The procedure is given in Algorithm 1.

Note that the algorithm is provided for the case of a static sensor. When the sensor is moving with the robot, such as is the case with our robotic wheelchair, it is necessary to adjust for the camera's velocity by subtracting the camera's (estimated) velocity vector from the extracted velocities.

3.4 Feature Vector

We represent our features in binary feature vectors. This representation is common in IRL [3,15,42] for its simplicity, but also produced robust results as shown in these papers.

In each grid cell, there is an associated binary feature vector that describes pedestrian velocities, which consists of pedestrian speed and direction, density, and distance to the goal in that cell. Speed, density, and distance features are represented using high, medium and low bins depending on some thresholds. For instance, if the number of points in a grid cell falls into a range that is specified as "high", then the binary value for high density bin is set to one. Same for speed and distance features.

Algorithm 1 RGB-D optical flow algorithm

```
Input: RGBImg_{t1}, RGBImg_{t2}, PtCloud_{t1}, PtCloud_{t2}

//Get correspondence information at each pixel in //RGBImg_{t1}
2Dflow = FranebackOpticalFlow(RGBImg_{t1} , RGBImg_{t2})

//Loop through all the pixels in the RGBImg_{t1}
for $i_{t1} = 1$ to 640 do
  for $j_{t1} = 1$ to 480 do
    //Get the index of corresponding pixel in frame $t2$
    $i_{t2} = i_{t1} + 2Dflow[i_{t1}, j_{t1}].x$
    $j_{t2} = j_{t1} + 2Dflow[i_{t1}, j_{t1}].y$

    //Map the pixels to the 3D points
    $[x_1, y_1, z_1]$ = MapPixelToPtCloud(RGBImg_{t1}[$i_{t1}, j_{t1}$])
    $[x_2, y_2, z_2]$ = MapPixelToPtCloud(RGBImg_{t2}[$i_{t2}, j_{t2}$])

    //Infer RGB-D optical flow at the point $(x_1, y_1, z_1)$
    $[vx, vy, vz] = [x_2, y_2, z_2] - [x_1, y_1, z_1]$
    save $vx$, $vy$, and $vz$ in vectors Vx,Vy,Vz
  end for
end for

Return Vx,Vy,Vz
```

Pedestrian direction feature describes whether there are points (i.e. pedestrians) moving into or out of a cell. We also represent these situations with three bins, each indicating high, medium, or low risk of a cell. A high risk cell is a cell with points flowing in from direct neighbouring cells, a medium risk cell is a cell with points flowing in from second-degree neighbouring cells (i.e. neighbour of neighbours), and a low risk cell is a cell with points in the cell moving out of it. If a cell has points moving out and points moving in from its direct neighbours, then it is considered to be a high risk cell. Note that these are absolute orientations indicating the future positions of the points.

For a given cell, we consider a 12 dimensional feature vector (four environmental features, density, speed, direction and distance to the goal, represented in high, medium and low binary values = $3 * 4$). The motivation behind employing the binary feature representation is to mitigate the noise in feature measurements. This is described in detail in Sect. 7.

4 Inverse Reinforcement Learning

To learn a socially adaptive cost function, we consider Inverse Reinforcement Learning (IRL), a sub-field of Reinforcement learning (RL). The traditional problem of RL is to infer an optimal plan from a sequence of trajectories [36]. The IRL problem, on the other hand, is to infer the cost function from demonstrations of an expert [24], assuming that the expert is unconsciously minimizing this (unknown) cost function for planning its action sequence. The inferred cost function is then used by an AI agent, such as a robot, to plan in new situations in such a way as to achieve performance similar to that of the expert.

The framework is particularly appealing for domains where it is difficult to specify a cost function analytically, and easier to provide demonstrations from an expert. In our case, it is difficult to specify a socially adaptive cost function, so we resort to learning it from demonstration data. Variations on the IRL concept have been examined in the last decade [3,24,28]. We focus on the maximum-a-posteriori Bayesian inverse reinforcement learning (MAP-BIRL), which was shown to perform well in navigation-type scenarios [7].

We now formally define Markov decision processes and IRL, and explain how these are used in our navigation framework.

4.1 Markov Decision Processes

A Markov Decision Process (MDP) [4] is defined as a tuple $< S, A, T, C, \gamma >$, where S represents the finite set of states, A represents the finite set of actions, $T(s, a, s')$ represents the state-to-state transition function, $C(s, a)$ represents the cost received when action a is taken in state s, and $\gamma \in [0, 1)$ represents the discount factor. The objective is usually to find an optimal plan $\pi^* : S \rightarrow A$ that minimizes the expected sum of future costs, or the value function:

$$Q^\pi(s_0, a_0) = E\left[C(s_0, a_0) + \sum_{t=1}^{\infty} \gamma^t C(s_t, \pi(s_t))|\pi\right] \quad (15)$$

The above function defines a expected sum of future costs executing a plan π after executing a_0 at the current state, s_0. We denote the value function obtained by the optimal plan π^* as Q^*. The typical approach to solving for Q^* is to use dynamic programming [26].

4.2 MDP Formulation for Navigation

4.2.1 State and Action Sets

The state set, S, is defined as the local grid cells in front of the robot. This gives the cardinality of S to be equal to the number of local grid cells. Thus, each cell defines a state, and a cell is reachable if the cell is one cell away from the current cell. The action set, A, for our navigation system includes a discrete action for moving into each cell that is adjacent to the current cell, according to the state space defined above. So for example if we assume a 3×5 local grid cells, there are 3 possible motion actions for the robot (directly in front, diagonally to the left, diagonally to the right). This action space is somewhat specific to our wheelchair robot that uses differential drive. However, a larger action set can easily be accommodated for other types of robots, such as holonomic robots. The transition function, $T(s, a, s')$, between each state is deterministic.

4.2.2 Cost Function

We define the feature function that maps a state and an action to a twelve-dimensional binary feature vector:

$$\phi : S \times A \rightarrow \{0, 1\}^{12} \tag{16}$$

This feature function $\phi(s, a)$ tells us what is the feature of the state that the robot will transition into by using action a in state s. The cost function for an action a in state s cell is calculated using a linear combination of associated features in the next state:

$$C(s, a) = \sum_{i=1}^{d} w_i \phi_i(s, a) \tag{17}$$
$$= w \cdot \phi(s, a), \tag{18}$$

where d represents the dimension of the feature vector, w_i represents the weight on the ith feature, and $\phi_i(s, a)$ represents the value of the ith feature at state s. Upon the transition, the robot will suffer the cost $C(s, a)$. Intuitively, the weight w_i determines if a particular feature value is preferred over another feature value. For instance, if the high density feature has higher weight than low speed feature, then the robot suffers a lower cost when moving into a cell that has low speed feature than moving into a cell that has high density feature. Manually setting these weights to come up with a socially adaptive path planner is clearly not trivial. Hence the benefit of the IRL algorithm, which allows us to learn these weights from human demonstrations.

4.2.3 MAP-BIRL for Robot Navigation

Generally, in IRL the expert's m^{th} demonstration is provided to the agent in the form of H-step state and action sequences, $\mathcal{X}^m = \{(s_1^m, a_1^m), ..., (s_H^m, a_H^m)\}$. The expected cumulative cost of executing action a in state s and following a policy π afterwards is given by the function $Q^{\pi}(s, a)$, as in a standard MDP. The goal in the MAP-BIRL framework is to determine the cost function by computing the weight vector w that maximizes the probability of the given demonstration data $\mathcal{X}$. We model the probability of a particular action $\hat{a}$ using the soft-max function:

$$P(\hat{a}|s, C) = \frac{exp(Q^*(s, \hat{a}))}{\sum_{a \in A} exp(Q^*(s, a))} \tag{19}$$

We assume that each action is independent of another, and model the probability of the given M trajectories as follows:

$$P(\mathcal{X}|C) = \prod_{m=1}^{M} \prod_{h=1}^{H} \frac{exp(Q^*(s_h^m, a_h^m))}{\sum_{a \in A} exp(Q^*(s_h^m, a))} \tag{20}$$

In this traditional IRL setting, the features associated with each state are be fixed. Under this assumption, we can solve for M different MDPs using dynamic programming, get M different Q^*, and define our likelihood as:

$$P(\mathcal{X}|C) = \prod_{m=1}^{M} \prod_{h=1}^{H} \frac{exp(Q_m^*(s_h^m, a_h^m))}{\sum_{a \in A} exp(Q_m^*(s_h^m, a))} \tag{21}$$

where Q_m^* denotes the optimal value function for m^{th} MDP. This is similar to the assumption made in [15].

To find the weight vector w, *maximum-a-posteriori (MAP)* inference is used on the log-likelihood of this model.

$$L(w) = \sum_{m=1}^{M} \sum_{h=1}^{H} log \left[\frac{exp(Q_m^*(s_h^m, a_h^m))}{\sum_{a \in A} exp(Q_m^*(s_h^m, a))} \right] \tag{22}$$

The optimization target is then:

$$w^* = \arg\max_{w} \{L(w)\} \tag{23}$$

Any optimization package can be used to solve this as the target is convex [8].

4.2.4 Regularization

When trying to imitate the behaviour of the expert, it may not be necessary to utilize all the features of the environment. For instance, in our experiments, we observed that the speed feature surprisingly had almost no effect in navigation behaviour, and only density, flux direction, and distance features mattered (it is possible that in the case of navigation in a crowd, people walk at very similar speeds, so the information is not useful). In such cases, using all features may lead to over-fitting the cost function, especially when we do not have a lot of training data. Moreover, this reduces the number of features needed to be estimated. To mitigate this, we employ a regularization technique within our IRL method. This model selection technique prevents over-fitting by eliminating those features that are unnecessary given the dataset using a penalty term. The popular choice for the penalty term for model selection is the L_1 norm [39]. The optimization target then becomes:

$$w^* = \arg\max_{w} \{L(w) + \lambda ||w||_1\} \tag{24}$$

Here, λ is the regularization parameter; higher λ means fewer features are considered, and vice versa. Any optimization package for L_1 regularization can be used to estimate

Table 1 Inferred weights for the cost function

	Den	Speed	Dir	DistGoal
High	1.0	0.0	1.7	0.3
Medium	−0.5	0.0	1.2	−0.1
Low	−1.2	0.0	−2.3	−0.4

the weights under this new objective, with a slight modification to incorporate to solve for M number of MDPs. We used the modification of Newton-Raphson method as shown in [9], where at each iteration we solve the MDPs to get $Q^*_m, m = 1 \ldots M$ with the cost function estimated for that iteration.

4.2.5 *Understanding the Cost Function*

In our feature design, we associate a binary feature vector to each grid cell which succinctly represents the flow and density information for that cell. For instance, a cell could have a binary feature vector that indicates high density and speed moving towards the robot. During human demonstrations, we compute the feature vector from the observable quantities (Kinect readings, human controller choice). The collection of such feature vectors defines our demonstration data.

Now consider the Eqs. (18) and (21). The likelihood in Eq. (21) is the likelihood of the demonstration data defined using the value function based on the cost function in Eq. (18). As such, maximizing the likelihood via Eq. (23) will find the weights of cost function such that it will make the likelihood of feature vectors seen in the demonstration data higher in the optimal plan obtained via Q^*. In other words, our feature design allows the IRL module to learn to prefer particular features, rather than learning every possible mapping from Kinect sensor reading to an action. As a consequence, our framework only needs to learn the preferences from demonstrations, instead of all possible appropriate actions under all possible navigation scenarios and environments. We indeed show this capability in the experiment section, by testing our framework under scenarios that were not explicitly included in the demonstration data.

As a consequence of learning from demonstrations, the weights determined by the IRL module specify the priority among features in determining a path. As noted in Sect. 4.2.3, we calculate the cost of a cell s using Eq. (18). Table 1 shows an example of typical weights as found during our experiments.

Since we use binary feature vectors, we can associate an intuitive meaning to each weight. Consider a case where the robot has two reachable cells, one cell having high density, low risk direction, and high distance to the goal, and another cell having low density, high risk direction, and high distance to the goal. Using (1), weights from the table 1, and the fact that our features are binary, the first cell would have a cost of $1 - 2.3 + 0.3 = -1.0$, and the second cell would have a cost of $-1.2 + 1.7 + 0.3 = 0.8$. These costs suggest that from human demonstrations, we learned that it is safer to move into a cell that has high density at the moment but the obstacle in that cell will move away, than into a cell that might be unoccupied at the moment but an obstacle is moving into that cell, given that these cells have the same distance to the goal.

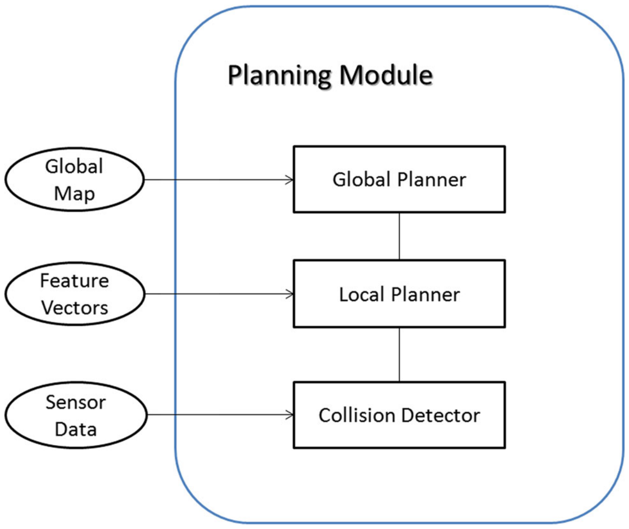

Fig. 5 Planner architecture

5 Planning

5.1 Planning Architecture

Since the RGB-D sensor has a limited range of view, two cost maps are considered simultaneously to achieve a full planning architecture: a global one and a local one. The global cost map, given *a priori*, is in an absolute reference frame where the goal is specified. The local cost map is in a local reference frame where features and costs are constantly updated based on the features as defined above. We define a path planning architecture that handles these two maps simultaneously, as shown in Fig. 5.

The architecture consists in fact of three layers: the global and local path planners, and a low-level collision detector. This is similar to the idea of a three-layer architecture suggested in [14], where you have several layers running in parallel, each responsible for different tasks simultaneously. Like the architecture suggested in [14], we have a fast-running collision detector that checks for obstacles in front of the robot several times a second based on simple hand-coded rules. The higher layers are responsible for the actual automatic path planning.

The specific role of each layer is as follows. Once the global goal is specified by the wheelchair robot user, the global path planner plans a path from the initial position to the goal using the map known *a priori*. Then, the local path planner, Algorithm 2, is executed to plan a path from the current position to the sub-goal (Recall Fig. 1). The local path planned is represented as a set of points, which we call *waypoints*, that needs to be followed in order for the robot to get to the sub-goal. The next destination for the robot therefore is the closest waypoint from the current robot position. The robot either reaches for this waypoint or executes the global path if the situation permits. While all of this is happening, the low level collision detector runs in parallel and can stop the robot if an obstacle is too close to the robot (e.g. $< 1m$). This prevents collisions caused by unforeseen risks, such as sudden appearance of obstacles or planning failure.

5.2 Local Path Planner

The local path planner, Algorithm 2, is an event-driven algorithm that returns the next destination to be reached. It proceeds as follows. It first waits for the set of feature vectors associated with each cell to be passed in by the feature extraction module. Once the feature vectors are received, it calculates the cost at each cell using the feature vector, weight vector, and Eq. (18). The algorithm then first checks if a safe zone is reached. The safe zone is defined as a small (e.g. 0.25 m) radius around the current waypoint, and the test returns true if the robot reaches this circle. If the safe zone test returns true, it then checks if an obstacle is detected. If the safe zone test returns false, the local planner continues to work towards the current waypoint. Note that the safe zone test automatically returns true if the current waypoint corresponds to the global goal. The second test (for obstacle detection) is true if one of the cells within the local action radius (e.g. 4 m) has high density. This is a check to see if we need to use our learned IRL cost function. Obviously this is not necessary if there are no dynamic obstacles in the vicinity of the robot, in which case a simple shortest path cost function (as optimized by the global planner) is quite sufficient. If the safe zone test and the obstacle detection test are both true, then the local planner optimizes a local path using Djkistra's algorithm over the learned feature-based IRL cost function. Figure 6 shows an example of local path planning.

Algorithm 2 Local Path Planning Algorithm

```
Input:FeatureVectors, WeightVector, GlobalPath,
CurrentWayPt
while RobotRunning do
  // Waiting for feature vectors
  WaitOn(FeatureVectors)

  // Calculate cost at each cell using Eqn. (1)
  GetCostAtEachCell(FeatureVectors, WeightVector)

  if SafeZone Reached then
    if Obstacle Not Detected then
      // Reach for the global goal
      return GlobalPath
    else
      // Plan a local path to the sub-goal
      LocalPath = DjkistraGraphSearch()
      NextDestination = ExtractWaypoint(LocalPath)
      return NextDestination
    end if
  else
    return CurrentWayPt
  end if
end while
```

6 Experiments

We assess our socially adaptive navigation framework by comparing the trajectories it produces with trajectories of human drivers, and trajectories obtained by the Dynamic Window Approach (DWA) [13], a shortest-path type planning method that uses laser data. We use scenario-based experiments, where the robot and human driver execute a similar scenario (using the same initial conditions) and repeat it many times, to objectively compare different measures of social adaptivity. We consider three metrics to measure the social adaptivity: closest distance to the pedestrian, avoidance distance to the pedestrian, and average time to reach the goal. Closest distance is calculated by measuring the distance from the center of the robot to the center of the closest pedestrian, when they are closest throughout the execution of a trajectory. Avoidance distance is measured by calculating the distance from the center of the robot to the center of the pedestrian when the angular velocity of robot is increased to a particular threshold, in an attempt to avoid the pedestrian. Closest distance to the pedestrian is a good measure of social adaptiveness because humans prefer to keep a particular distance to unknown individuals during a navigation. Avoidance distance is an important factor of social adaptivity because pedestrians expect the wheelchair to start avoiding them from a certain distance. Besides these two metrics, we also measure the average length of time to complete the trajectory. We first describe the experimental setup, and then the target scenarios. The rationale for selecting each scenario is also provided in the corresponding section.

6.1 Setup and Navigation Scenarios

We deployed our algorithm on the Smartwheeler robotic wheelchair [25]. To acquire the initial training data necessary to estimate the weights, we asked an expert to manually

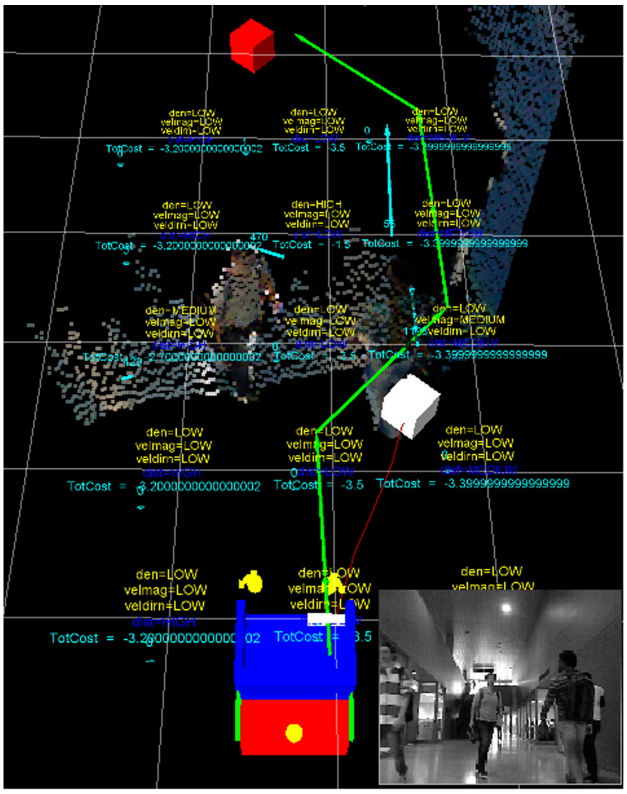

Fig. 6 Example of planning and its corresponding *black* and *white* image (bottom-right corner). The sub-goal is marked with a *red cube*, and a waypoint marked with a *white cube*. In an attempt to avoid the approaching pedestrians, the robot plans a path to join the pedestrians on the right, who are moving in the direction of the goal

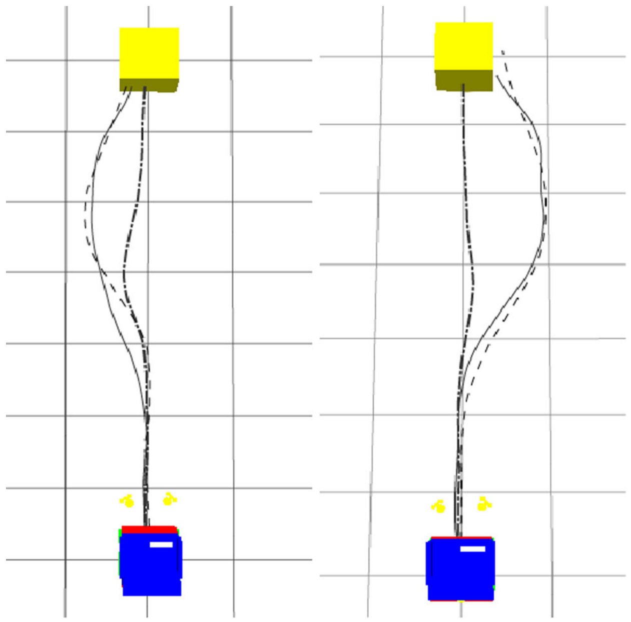

Fig. 7 Average trajectories executed by IRL (*solid*), human driver (*dashed*) and DWA (*circle-dashed*). *Left* figure depicts the case in which the pedestrian approached the robot and turned right (w.r.t to the robot) to avoid the robot, and the *right* figure is when the pedestrian approached the robot and turned left to avoid the robot. *Yellow cube* represents the goal that is approximately 7 m away. (Color figure online)

drive the Smartwheeler within our university building to gather human trajectories for the IRL module. Specifically, we recorded log files while driving, which included features extracted and actions taken (i.e. choosing to moving into a particular cell) during the drive. We collected 17 log files, each of 1–3 min. The log files were collected in conditions where there were multiple pedestrians approaching or moving away from the wheelchair, from various directions. These were collected mostly in a lab, which is an open space filled with people, and never in a hallway or through doors. These were sent to the IRL module as state and action sequences; the weights were then calculated offline. Note that the conditions in which the data were collected are not identical to the conditions for the experiments below. This is intentional, to show robustness of the approach to a variety of conditions. The same training set is used for the three test scenarios listed below. For each of the scenarios, we also used the same initial conditions (start positions and goals). For each path planning approach (IRL, human driver, and DWA), we repeated each scenario ten times and then calculated the average executed trajectory length, and the mean and 95 % confidence interval for the two metrics of social adaptivity. The robot was operating at a constant speed, for both training and testing.

6.2 Scenario 1: Pedestrian Walking Towards the Robot

In this scenario, a pedestrian and the robot start ten meters apart, facing each other. Their respective goals are seven meters directly forward of their initial position. We consider this scenario because this passing behaviour is a common one in everyday situations. Also, it is the type of scenario where the robot can learn socially adaptive behaviours by observing how human drivers use the pedestrian's pose to determine in which direction to turn, and by how much. For instance, we noticed that when a pedestrian notices our robot approaching, s/he often decides which direction to turn *before* the robot turns. Consequently, the robot learns to turn in the opposite direction to avoid the pedestrian. However, the robot needs to turn enough so that there is a comfortable distance between the pedestrian and the robot. Moreover, the robot needs to turn from a good enough distance so that the pedestrian can feel comfortable with the robot's navigation behaviour, by confirming the robot is indeed trying to avoid him.

Such behaviours are well illustrated in the average trajectory execution shown in Fig. 7. Note that in this scenario, the pedestrian randomly decides to turn left or right to avoid the robot. We separated these two cases for the average trajectory. As can be observed from the average trajectory, the trajectory of the DWA planner is far from that of the human driver, while the trajectory of the IRL planner is very close to that of the human driver.

Table 2 Metrics and their 95 % confidence interval for social adaptiveness for Scenario 1

Average vals	Human	DWA	IRL
Closest distance (m)	1.64 ± 0.12	1.16 ± 0.13	1.62 ± 0.15
Avoid distance (m)	3.04 ± 0.33	2.64 ± 0.22	3.21 ± 0.17
Time to goal (s)	10.53 ± 0.38	11.01 ± 1.05	11.57 ± 0.78

The metrics for navigation performance are shown in Table 2. The average value of closest distance measured in the IRL planner pedestrian is very similar to that of the human driver, and much more conservative than the DWA planner. The avoidance distance is a bit further than the human. The time to reach the goal is a bit more than that of the human driver.

6.3 Scenario 2: Pedestrian Walking Horizontally to the Robot

In this scenario, the pedestrian approaches the robot from the right side and moves towards the left, with respect to the robot. The robot is moving perpendicularly to the pedestrian's motion, trying to get to a goal that is seven meters from the current position. This scenario illustrates well how a human driver will trade-off the density feature of the pedestrian (represented in his current position), with the direction feature (representing the pedestrian's future position). Unlike traditional path planners that try to simply avoid obstacles, our planner can select a path to move to the pedestrian's current position, such as to avoid the pedestrian's future position, similar to what a human driver would do.

Figure 8 shows the comparison of average trajectory executed by the human and the robot. In the DWA planner's trajectory, the robot tried to avoid the incoming pedestrian by trying to avoid him at his current position. However, as the avoidance trajectory interfered with the pedestrian's trajectory, the robot often had to repeatedly stop to wait until the pedestrian passed. In contrast, the trajectories of the human driver and IRL planner took into account the pedestrian's direction and avoided his future position. Such trajectories are possible because the weights learned from the IRL module prioritize the direction feature (future position) over the density feature (current position).

Table 3 shows the results for the objective measures. These suggest that the DWA planner again gets very close, within 0.75 m to the pedestrian, while the IRL and human driver trajectories maintained a 1.5 m distance to the pedestrian. Also in terms of avoidance distance, DWA started avoiding the pedestrian relatively later than the human driver and the IRL planner. The avoidance distance of the IRL planner was a bit closer than that of the human driver. This is likely due to the limited field of view of the Kinect, compared to that of human vision, which makes horizontally approaching obstacles harder to detect. The time to reach the goal was almost

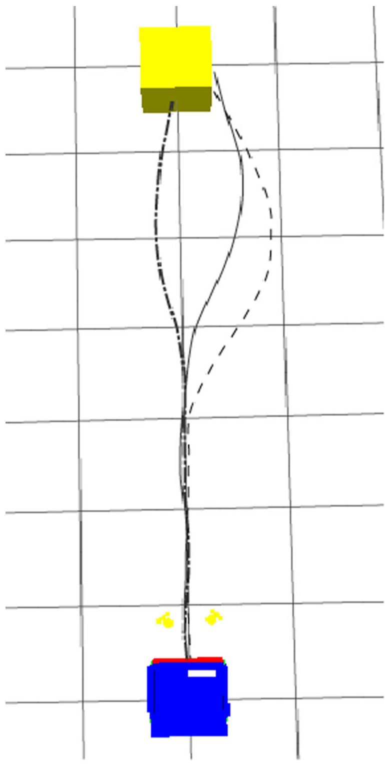

Fig. 8 Average trajectories executed by IRL (*solid*), human driver (*dashed*) and DWA (*circle-dashed*). when a pedestrian approached the robot from the right side. The IRL and human driver moves towards where the pedestrian is, while DWA moves away from him

the same for the IRL planner and human driver. The DWA planner required more time, due to its repeated stop-and-go behavior.

6.4 Scenario 3: Multiple Pedestrians

In this scenario, we have multiple pedestrians approaching the robot from the front at a distance of 3–4 m, and a pedestrian that is moving parallel to robot to its left (and in the direction of the goal). The goal is set 6m away from the front of the robot. This scenario shows the robot's ability to join a flow direction that is in the direction of the goal, while simultaneously avoiding an opposing flow direction. As argued in [15], this is one of the essential abilities of socially adaptive path planners.

The average trajectories are shown in Fig. 9. We observe that the trajectories of the human and IRL planner initially

Table 3 Metrics and their 95 % confidence interval for social adaptiveness for Scenario 2

Average vals	Human	DWA	IRL
Closest distance (m)	1.49 ± 0.17	0.75 ± 0.05	1.50 ± 0.15
Avoid distance (m)	1.77 ± 0.15	1.15 ± 0.14	1.54 ± 0.10
Time to goal (s)	11.60 ± 0.25	13.24 ± 0.38	11.63 ± 0.30

Table 4 Metrics and their 95 % confidence interval for social adaptiveness for Scenario 3

Average vals	Human	DWA	IRL
Closest distance (m)	1.00 ± 0.06	0.86 ± 0.13	1.13 ± 0.15
Avoid distance (m)	3.46 ± 0.07	3.05 ± 0.14	3.55 ± 0.12
Time to goal (s)	9.54 ± 0.50	12.10 ± 1.70	8.60 ± 0.30

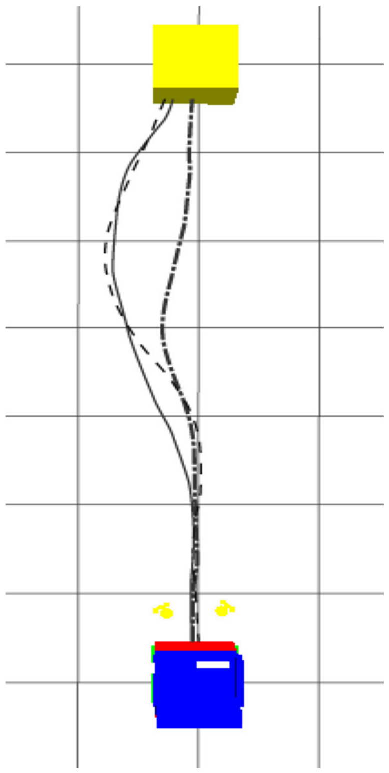
Fig. 9 Average trajectories executed by IRL (*solid*), human driver (*dashed*) and DWA (*circle-dashed*). It initially turns left to avoid the incoming crowds that moves towards the robot, and to join the pedestrian that is moving away from the robot. After the incoming crowds are avoided, it moves towards the goal

avoid the incoming crowd and join the pedestrian moving towards the goal, following him for a while until the crowd is avoided. After the avoidance, both trajectories approach the goal. The DWA planner's trajectory also initially avoids the incoming crowd, but it abruptly stops when the lateral pedestrian appears in front of the robot. As the pedestrian approaches the goal, the DWA planner tries to avoid him by making an unnecessary turn right before reaching the goal. The IRL planner shows a trajectory that is a bit different from the human driver, in that it reaches the goal sooner than the human driver does. This is because once the crowd is avoided, the high density feature in the corresponding cells are set to low, and the distance feature (which is more important than the low density feature) contributes more to the cost function. As a result, the robot immediately tries to approach cells that are closer to the goal.

The objective metrics are shown in Table 4. We observe that despite the disparity in the average trajectory shown in Fig. 9, the measured closest distance and avoidance distance of the IRL planner are very close to that of the human driver. Moreover, the IRL planner actually reaches the goal faster than the human driver, by reaching for cells that are closer to the goal cell. The DWA planner in this scenario performed worse than the other two planners. It got closer to the pedestrian, sometimes within 0.6 m, and avoided the incoming crowd relatively later than the human driver. Its average time to reach the goal is also significantly longer than the IRL planner.

6.5 Crowded Hallway

We place the robot in a busy hallways of a university building during normal school hours, as shown in Fig. 10. Note that the robot has never seen this environment before (i.e. no expert demonstrations are performed in this setting.) The goal for the robot is to get from one end of the hallways to the other while avoiding pedestrians as well as a few static obstacles such as stairs and garbage cans along the way. The purpose of this experiment is to show that the method is sufficiently robust for non-controlled environments, and in particular that the cost function learned can be used in previously unseen settings.

For this scenario, we do not provide trajectory comparisons. Since the pedestrians were not moving in a predefined manner as they did in our previous scenarios, crowd situations from one run to another varied significantly, making direct trajectory comparisons meaningless.

We repeated the experiment ten times using each of the three methods (IRL controller, DWA planner, and a human

Table 5 Metrics and their 95 % confidence interval for social adaptiveness for the Crowded Hallway scenario

Average vals	Human	DWA	IRL
Closest distance (m)	0.97 ± 0.06	0.73 ± 0.02	1.06 ± 0.08
Avg avoid distance (m)	1.06 ± 0.14	1.01 ± 0.07	1.32 ± 0.11
Time to goal (s)	41.25 ± 2.15	45.33 ± 2.61	49.40 ± 3.56
Human intervention (%)	100 ± 0	68.22 ± 0.68	18.18 ± 1.58

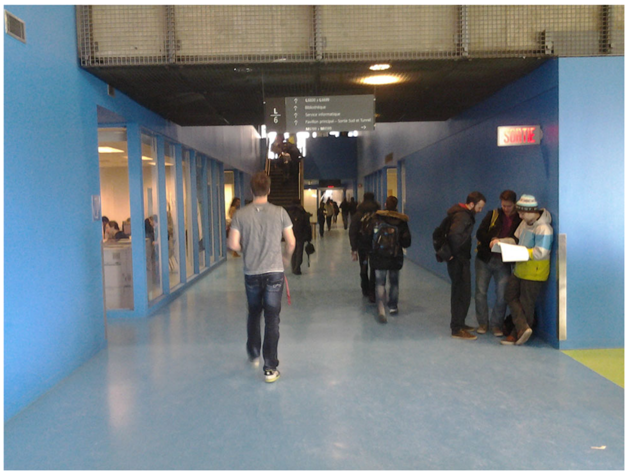

Fig. 10 A picture of the hallway

controller). These repetitions were used only for evaluating each approach, not for re-training the system (we used the IRL system that was trained in the lab without any modification). In all cases, the robot was required to travel from one end of the hallway to the other and back. The results are shown in Table 5. For this scenario, we allowed a human, sitting on the wheelchair, to intervene when deemed necessary during the navigations. We measured the human intervention percentage during the autonomous navigations computed by counting the number of velocity commands that human sent via remote controller, and divide it by the total number of velocity commands executed by the robot.

As the results show, IRL was very similar to the human driver in terms of closest distance to a pedestrian. However, IRL showed several conservative motions when making avoidance motion; compared to human, the avoid distance was higher on average. Most of human interventions for IRL was due to the limited sensing. In other words, because of limited field of view of Kinect, it could not see the people behind or right between the robot. However, we believe this problem can be resolved by fusing multiple sensor inputs (i.e. having Kinect at the back and sides, not just at the front). For DWA, the human often had to intervene when the robot was stuck in a human crowd, as it does not have the ability to join and follow the crowd. As for time to goal, DWA was able to get to the goal faster than IRL as the human operator was in control most of the time.

In addition to the social adaptiveness metric, we provide the video recorded from the on-board Kinect on the robot during a sample of this experiment.[1]

7 Discussion

In this paper, we proposed a socially adaptive path planning framework that closely resembles the navigation behaviour of a human operator. Our work was motivated by the realization that for assistive robots that interact with humans on a daily basis, it is crucial to take into account the social variables to provide seamless navigation in environments filled with humans.

There were several challenges involved in this work. The main challenge was in defining the cost function over social variables. Our intuition was that while it is hard to manually define a cost function based on such social variables, it is easier to learn the cost function from a human demonstrator, who is instinctively aware of such variables. We employed an IRL framework to resolve this challenge. The other challenge was the integration of the IRL framework with a real robot navigation system. This involved developing an optical flow algorithm based on the RGB-D sensor, designing an MDP and an appropriate navigation architecture, and proposing a new local path planning algorithm that works in accordance with the designed architecture.

Unlike previous works [15,37], our framework was submitted to a thorough empirical evaluation with a robot platform. Using scenario-based experiments, we showed that the behaviour achieved by our framework closely resembles trajectories produced by a human operator, as illustrated by the social adaptivity metrics and average trajectories presented in the experiment section. Specifically, in the first scenario we observed that trajectories of DWA, which employs a standard cost function based on occupancy of grid cells, are not socially adaptive as it drives too closely to the pedestrian and avoids the pedestrian too late. The IRL trajectory, in contrast, closely resembled the human driver's trajectory which likely made both pedestrian and the person on the robot feel more comfortable and safe. In the second scenario, our method successfully avoided the future position

[1] http://www.youtube.com/watch?v=T9p-prVXr0M

of the pedestrian, whereas the DWA planner tried to avoid the current position of the pedestrian and had to repeatedly stop and restart. This again shows that our navigation algorithm is socially adaptive as it learned from the human demonstrations that the future positions of moving pedestrians is more important than their current position. In the third scenario, we showed the essential ability of socially adaptive path planner - joining the flow of the crowd moving in the direction of the goal. We hypothesize that the pedestrians will also feel more comfortable with this behaviour, compared to the DWA behiavor which tried to actively avoid the crowd. It is noteworthy that our IRL approach used a low-cost RGB-D sensor, compared to the expensive laser range-finder used for the DWA strategy.

There are a few limitations in the work we described. First, the estimated RGB-D optical flow is inherently noisy and does not account for occlusions. This complicates the well-known correspondence problem (i.e. identifying which points from two different RGB scans correspond to the same physical item). Another limitation is the fact that the IRL framework performs best when given precise feature measurements. As the cost function is linear in the features, if these features are not measured precisely, the cost estimates can be noisy, which can lead to poor planning. We employed a binary feature vector representation with regularization to alleviate this problem; however, the choice of base features remains an open problem.

We performed experiments in three well-defined controlled scenarios, as well as in an uncontrolled dynamic human environment. The controlled experiments allowed us to perform an objective comparison of the performance of the three navigation planning approaches, while the uncontrolled experiment allowed us to demonstrate the ability of our navigation framework in a real-world human environment. However, the noteworthy caveat is that due to the limitations of our RGB-D optical flow mentioned in the previous paragraph, our approach could not be fully autonomous. For instance, occlusions make our framework to avoid a person right in front, but not the person occluded by the person that is behind the person that the robot just avoided. However, we believe that by employing more sophisticated approaches for pedestrian movement tracking, such as [21], that our approach can be fully autonomous.

In our experiments, we primarily compared our method with a DWA planner. The original DWA has mostly proven successful in static environments, and thus did not perform very well in our domain. And although there are extensions of the original DWA proposed for dynamical environment, such as [31], these typically assume full observability of the linear and angular velocities of all the dynamic obstacles in the environment, which is infeasible in most human environments, such as those used in our evaluations. As mentioned in Sect. 2.1, computing the velocity flow is already a difficult computer vision problem. One of the advantages of our approach is that it can use any available feature, including noisy, partially observable ones. While we consider a limited set of features in the current implementation, extending the learning framework to a richer feature vector is trivial.

In future work, we also hope to combine natural language commands to make the navigation easier for wheelchair robot users. Currently, the wheelchair uses a tactile interface to allow users to specify the global goal; this poses limitations for severely disabled individuals that cannot use the touch interface [40]. In the short term, we should be able to facilitate navigation via voice commands, by specifying the global goal, or using commands such as “follow the right wall” in a very crowded environment where the robot cannot find a valid path. There has been significant work in the area of language grounding, where the goal is to map the meaning of semantics of natural language sentences with physical systems [22,38], which suggests promising opportunities in this direction.

Acknowledgments The authors gratefully acknowledge financial and organizational support from the following research organizations: Regroupements stratgiques REPARTI and INTER (funded by the FQRNT), CanWheel (funded by the CIHR), the CRIR Rehabilitation Living Lab (funded by the FRQS), and the NSERC Canadian Network on Field Robotics. Additional funding was provided by NSERC’s Discovery program. We are also very grateful for the support of Andrew Sutcliffe, Anas el Fathi, Hai Nguyen and Richard Gourdeau for the realization of the empirical evaluation.

References

1. Abbeel P, Coates A, Ng AY (2010) Autonomous helicopter aerobatics through apprenticeship learning. Int J Robot Res 29(13):1608–1639
2. Abbeel P, Dolgov D, Ng A, Thrun S (2008) Apprenticeship learning for motion planning, with application to parking lot navigation. In: IEEE/RSJ international conference on intelligent robots and systems (IROS-08)
3. Abbeel P, Ng A (2004) Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the twenty-first international conference on machine learning. ACM, New York, p 1
4. Bellman R (1957) A markovian decision process. J Math Mech 6:679–684
5. Bennewitz M, Burgard W, Cielniak G, Thrun S (2005) Learning motion patterns of people for compliant robot motion. Int J Robot Res 24(1):31–48
6. Bruhn A, Weickert J, Schnörr C (2005) Lucas/kanade meets horn/schunck: combining local and global optic flow methods. Int J Comput Vis 61(3):211–231
7. Choi J, Kim K-E (2011) Map inference for bayesian inverse reinforcement learning. In: NIPS, pp 1989–1997
8. CVX Research I (2012) CVX: Matlab software for disciplined convex programming, version 2.0 beta. http://cvxr.com/cvx
9. Fan J, Li R (2001) Variable selection via nonconcave penalized likelihood and its oracle properties. J Am Stat Assoc 96:1348–1360
10. Farnebäck G (1999) Spatial domain methods for orientation and velocity estimation. Ph.D. thesis, Linköping
11. Farnebäck G (2000) Fast and accurate motion estimation using orientation tensors and parametric motion models. In: Proceedings

of 15th international conference on pattern recognition, vol 1. pp 135–139
12. Foka AF, Trahanias PE (2010) Probabilistic autonomous robot navigation in dynamic environments with human motion prediction. Int J Soc Robot 2(1):79–94
13. Fox D, Burgard W, Thrun S (1997) The dynamic window approach to collision avoidance. Robot Autom Mag 4:23–33
14. Gat E (1998) On three-layer architectures. Artificial intelligence and mobile robots. MIT Press, Cambridge
15. Henry P, Vollmer C, Ferris B, Fox D (2010) Learning to navigate through crowded environments. In: ICRA. pp 981–986
16. Horn BKP, Schunck BG (1981) Determining optical flow. Artif Intell 17:185–203
17. Knutsson H, Westin CF, Andersson M (2011) Representing local structure using tensors II. Image analysis. Springer, Berlin, pp 545–556
18. Kruse T, Kirsch A, Sisbot EA, Alami R (2010) Exploiting human cooperation in human-centered robot navigation. In: ROMAN
19. Kruse T, Pandey AK, Alami R, Kirsch A (2013) Human-aware robot navigation: a survey. Robot Auton Syst 61:1726–1743
20. Lavalle S (2006) Planning algorithms. Cambridge University Press, Cambridge
21. Luber M, Arras KO (2013) Multi-hypothesis social grouping and tracking for mobile robots. Robotics: science and systems. Springer, Berlin
22. Matuszek C, Herbst E, Zettlemoyer L, Fox D (2012) Learning to parse natural language commands to a robot control system. In: 13th international symposium on experimental robotics (ISER)
23. Montemerlo M, Thrun S, Whittaker W (2002) Conditional particle filters for simultaneous mobile robot localization and people-tracking. In: IEEE international conference on robotics and automation (ICRA)
24. Ng AY, Russell S (2000) Algorithms for inverse reinforcement learning. In: International conference on machine learning
25. Pineau J, Atrash A (2007) Smartwheeler: a robotic wheelchair test-bed for investigating new models of human–robot interaction. In: AAAI spring symposium: multidisciplinary collaboration for socially assistive robotics. AAAI, pp 59–64
26. Puterman ML (2009) Markov decision processes: discrete stochastic dynamic programming, vol 414. Wiley-Interscience, New York
27. 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
28. Ramachandran D, Amir E (2007) Bayesian inverse reinforcement learning. In: IJCAI. pp 2586–2591
29. Ratsamee P, Mae Y, Ohara K, Takubo T, Arai T (2013) Human-robot collision avoidance using a modified social force model with body pose and face orientation. Int J Hum Robot 10:1350008
30. Schulz D, Burgard W, Fox D, Cremers AB (2003) People tracking with mobile robots using sample-based joint probabilistic data association filters. Int J Robot Res 22:99–116
31. Seder M, Petrovic I (2007) Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In: International conference on robotics and automation
32. Shiomi M, Zanlungo F, Hayashi K, Kanda T (2014) Towards a socially acceptable collision avoidance for a mobile robot navigating among pedestrians using a pedestrian model. Int J Soc Robot 6:443–455
33. Sisbot EA, Marin-Urias LF, Alami R, Simeon T (2007) A human aware mobile robot motion planner. IEEE Trans Robot 23:874–883
34. Stentz A, Mellon IC (1993) Optimal and efficient path planning for unknown and dynamic environments. Int J Robot Autom 10:89–100
35. Sun D, Roth S, Black MJ (2010) Secrets of optical flow estimation and their principles. In: 2010 IEEE conference on computer vision and pattern recognition (CVPR), IEEE. pp 2432–2439
36. Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT Press, Cambridge
37. Svenstrup M, Bak T, Andersen H (2010) Trajectory planning for robots in dynamic human environments. In: IEEE/RSJ international conference on intelligent robots and systems
38. Tellex S, Kollar T, Dickerson S, Walter M, Banerjee A, Teller S, Roy N (2011) Understanding natural language commands for robotic navigation and mobile manipulation. In: Proceedings of AAAI
39. Tibshirani R (1996) Regression shrinkage and selection via the lasso. J R Stat Soc Ser B 58:267–288
40. Tsang E, Ong SCW, Pineau J (2012) Design and evaluation of a flexible interface for spatial navigation. In: Canadian conference on computer and robot vision. pp 353–360
41. Yuan F, Twardon L, Hanheide M (2010) Dynamic path planning adopting human navigation strategies for a domestic mobile robot. In: Intelligent robots and systems (IROS)
42. Ziebart BD, Maas A, Bagnell JA, Dey AK (2008) Navigate like a cabbie: probabilistic reasoning from observed context-aware behavior. In: Proceedings of Ubicomp. pp 322–331
43. Ziebart BD, Ratliff N, Gallagher G, Mertz C, Peterson K, Bagnell JA, Hebert M, Dey AK, Srinivasa S (2009) Planning-based prediction for pedestrians. In: Proceedings of the international conference on intelligent robots and systems

Beomjoon Kim is currently a Ph.D. student at Massachusetts Institute of Technology, Cambridge, MA, US. He received BMath in Statistics and Computer Science from University of Waterloo, Waterloo, ON, Canada, in 2011, and M.Sc in Computer Science at McGill University, Montreal, QC, Canada, in 2013, under the supervision of Professor Joelle Pineau. This work was completed while this author was an M.Sc candidate at McGill University. His research interest includes developing planning and control algorithms for robots, using tools from machine learning, optimization, and control theory.

Joelle Pineau is currently an Associate Professor with the School of Computer Science, McGill University, Montreal, QC, Canada, where she is also a member of the Centre for Intelligent Machines. She received the B.A.Sc. degree in engineering from the University of Waterloo, Waterloo, ON, Canada, in 1998, and the M.Sc. and Ph.D. degrees in robotics from Carnegie Mellon University, Pittsburgh, PA, USA, in 2001 and 2004, respectively. Her research focuses on developing models and algorithms for learning and decision-making in partially observable stochastic domains, and applying these results to complex problems in robotics and healthcare