User Preferred Behaviors for Robot Navigation Exploiting Previous Experiences

Lorenzo Nardi[a,*], Cyrill Stachniss[a]

[a] *University of Bonn, Institute for Geodesy and Geoinformation, Nussallee 15, 53115 Bonn, Germany*

Abstract

Industry demands flexible robots that are able to accomplish different tasks at different locations such as navigation and mobile manipulation. Operators often require mobile robots operating on factory floors to follow definite and predictable behaviors. This becomes particularly important when a robot shares the workspace with other moving entities. In this paper, we present a system for robot navigation that exploits previous experiences to generate predictable behaviors that meet user's preferences. Preferences are not explicitly formulated but implicitly extracted from robot experiences and automatically considered to plan paths for the successive tasks without requiring experts to hard-code rules or strategies. Our system aims at accomplishing navigation behaviors that follow user's preferences also to avoid dynamic obstacles. We achieve this by considering a probabilistic approach for modeling uncertain trajectories of the moving entities that share the workspace with the robot. We implemented and thoroughly tested our system both in simulation and on a real mobile robot. The extensive experiments presented in this paper demonstrate that our approach allows a robot for successfully navigating while performing predictable behaviors and meeting user's preferences.

Keywords: Navigation, Path planning, Collision avoidance, Trajectory prediction

1. Introduction

Over the last decades, robots have reshaped the manufacturing industry. Traditional industrial robots are manipulators designed to perform specific tasks at fixed locations. Recently, the demand for flexible robotic solutions that are able to perform different tasks at different locations has grown. These tasks include mobile manipulation and navigation to transport materials and tools from one location to another. To operate on factory floors, mobile robots are usually requested to fulfill some requirements. Often, it is desirable for robots to navigate only in certain areas of the environment and to follow definite patterns while navigating. Furthermore, robots frequently operate in environments populated by other moving entities such as human workers, forklifts, other robots, etc.. In this context, the reproducibility and the predictability of the navigation become essential. Especially in shared workspaces, these properties are important to avoid collisions without limiting the flexibility of the robot navigation.

In this paper, we propose an approach that allows a robot to navigate in such environments considering the preferences of the user. It implicitly collects preferences from robot's previous experiences and exploits them to perform new tasks. This is generally more convenient than hard-coding control strategies. Formalizing preferences can be difficult and hard-coded rules are often complex to maintain and update. In the same way, using cost functions to define behaviors can be problematic and usually requires experts to handle them. On the contrary, even non-expert users can easily teach behaviors to the robot by demonstration, e.g. joysticking the robot along desired routes in the environment.

Our approach aims at reproducing and generalizing previous successful behaviors to meet user's preferences. There exist several motion planning approaches that aim at reusing previous experiences, but most of them do not focus on the resulting robot behavior. Teach-and-repeat allows a robot for repeating trajectories, however it is hard to generalize them to different situations. This paper builds upon and extends our recent conference paper [24] in which we introduce a planning approach to reuse robot's previous experiences to meet user's preferences while maintaining the planning flexible. We extend this approach to consider preferences for robot navigation also in dynamic environments. Our objective is to avoid obstacles by computing local deviations from the path that satisfy user's preferences, while maintaining the global behavior. We achieve this by considering a probabilistic method that takes the uncertainty of the obstacle motion into account. We furthermore present an extended experimental evaluation in simulated and real world environments.

The main contribution of this paper is a system for robot navigation that allows the robot (i) to meet the preferences of the user exploiting the previous experiences of the robot, (ii) to reproduce behaviors over different environments and situations, (iii) to perform predictable behaviors providing similar solutions for similar situations,

*Corresponding author

Email addresses: lorenzo.nardi@uni-bonn.de (Lorenzo Nardi), cyrill.stachniss@igg.uni-bonn.de (Cyrill Stachniss)

and (iv) to navigate in environments populated by other moving objects while still accomplishing foreseeable behaviors. We achieve this by allowing the user to demonstrate desired behaviors to the robot or to provide feedback about its experiences. We store user's favorite behaviors and reuse them to guide the planning for a new but similar navigation task. We define a similarity relationship among tasks and a path representation for transferring experiences to new situations. Our planning approach takes advantages from theses concepts and considers previous experiences while maintaining at the same time the flexibility of a general planner. We implemented our system using C++ and ROS, and we realized our planner in the Open Motion Planning Library (OMPL) framework [33].

2. Related Work

Motion planning algorithms for robot navigation have been widely studied [19]. A class of commonly used approaches are sampling-based planners such as Rapidly-exploring random tree (RRT) [20] and its variants as RRT-Connect [18]. These approaches are fast to discover the connectivity of a configuration space but it is difficult to predict the resulting paths due to their random nature.

Bruce and Veloso [3] extend traditional RRT to reuse cached plans and bias a new search towards their waypoints. Zucker *et al.* [36] propose to adapt the sampling distribution of sampling-based motion planners considering the features of workspace. Jiang and Kallmann [13] propose the Attractor Guided Planner (AGP) that stores every successful path and biases a new search to reproduce the structure of an experienced path according to a similarity function. Our planning approach reuses and generalizes the idea of experience-guided planner.

Other motion planning approaches reuse experiences: Lien and Yu [21] construct roadmaps around geometric models and reused them to plan for avoiding obstacles; Fraichard and Delsart [5] deform previous trajectories in the configuration-time space to fit new situations; Phillips *et al.* [27] build a graph of experiences and exploit it for performing repetitive tasks. The Lightning framework [2] repairs previously generated trajectories to fit new planning problems. Coleman *et al.* [4] extend this idea by storing generated paths in a sparse roadmap. Such approaches take advantage from previous experiences to speed up the planning time and to find a path in high-dimensional configuration spaces. Our approach does not rely on any complex model or cost function and aims at reproducing robot's previous experiences to meet user's preferences.

Jetchev and Toussaint [11] learn a mapping between situations and trajectories and use a descriptor of the situation to transfer and optimize a previous trajectory in a new situation. They extend this work in [12], using a voxel representation of the environment to generalize trajectories to a wider range of situations. Our approach also uses the concept of situation descriptor to identify previous paths fitting new situations.

Many teach-and-repeat approaches exist to reproduce previously taught experiences. Sprunk *et al.* [32] reproduce trajectories with high accuracy by matching laser scans. Furgale *et al.* [9] propose a vision-based approach based on topologically connected submaps. Furgale *et al.* [10] extend standard teach-and-repeat by adding a local motion planner to account for dynamic environments. Perea Ström *et al.* [25] use a similar approach for guiding a robot home in case the mapping system fails during an exploration mission. Mazuran *et al.* [22] optimize the demonstrated trajectories within constraints defined according to user's preferences. In contrast to teach-and-repeat, our objective is to maintain the flexibility of a real planner while reproducing behaviors.

Case-based reasoning is another approach related to our work. It considers robot experiences to build experimental models and that are stored as cases and used in the future tasks. Meriçli *et al.* [23] adopt this approach to mobile push-manipulation, while Ros *et al.* [30] use it for action selection in the robot soccer domain.

There exist several approaches for robot navigation in dynamic environments. Many of them rely on reactive strategies as Dynamic Window Approach [7] that repeatedly selects a velocity yielding to a safe trajectory within a short time interval. Elastic Bands [28] combines a reactive strategy with global planning to avoid obstacles. Rösmann *et al.* [31] and successively Keller *et al.* [14] extend this approach to consider explicitly time information for optimal trajectory planning. These approaches enable robots to avoid dynamic obstacles, but do not allow for expressing preferences about their behaviors.

Other researchers study human motion to allow robots for compliant navigation in populated environments. Kruse *et al.* [16] conduct a survey about human-aware navigation. Ziebart *et al.* [35] learn a cost function of features to predict people trajectories and plan paths that avoid hindrances. Bennewitz *et al.* [1] learn collections of trajectories that characterize typical human motion patterns. Kuderer *et al.* [17] as well as Kretzschmar *et al.* [15] model cooperative navigation behaviors of humans to let the robot interact with people in a socially compliant way. Pfeiffer *et al.* [26] propose a navigation model trained on human-human interaction. Trautman *et al.* [34] introduce Interacting Gaussian Processes (IGPs) for navigating through dense human crowds. IGPs describe a probabilistic interaction between multiple navigating entities to accomplish cooperative collision avoidance. Fulgenzi *et al.* [8] use Gaussian Processes to model typical obstacle trajectories and planned with a variant of RRT that explicitly consider the probability of collision. Ellis *et al.* [6] show that Gaussian Processes allow to explicitly represent the uncertainty of the human motion. We consider a similar probabilistic approach to model the uncertain trajectories of the entities with whom a robot shares the workspace on a factory floor.

3. Our Approach

3.1. Use Case

We consider a use case throughout this paper to explain and motivate our work: a non-expert operator requests a mobile robot to perform navigation tasks on a factory floor. In this environment, other moving entities such as human workers, forklifts, other robots, etc. may share the workspace with the robot. We assume that both the robot and the operator know the environment. The operator asks the robot to perform a navigation task by specifying the start and goal poses. This scenario stems from the EU-funded H2020 project RobDREAM that focuses on automatically adapting and improving the behaviors of a robot over its previous experiences. Our work enables the robot to accomplish such tasks by adapting its navigation behaviors according to the user's preferences. The operator can express preferences by rating behaviors experienced by the robot as *good* or *bad*, for example through a simple GUI. The operator can also demonstrate good behaviors to the robot by joysticking the platform. We store the good experiences into a database that grows over time. Our system exploits such a database of examples to capture and reproduce the preferences of the user.

3.2. Overview of the Navigation System

Our system works on two levels: the *global level* and the *local level*. We aim at providing behaviors that reproduce previous successful experiences of the robot at both levels. Given a navigation task, the global level computes a path from the start to the goal pose in the static map of the environment. The global level can exploit previous experiences to perform a task in the same environment, but does not generalize the exploitation of experiences across different environments. The local level handles collision avoidance with unforeseen obstacles planning deviations from the global path. It considers only the local situation. Therefore, the local level is largely independent from the map of the environment and generalizes well previous experiences across different scenes and environments.

We consider four key concepts to realize a flexible navigation system that can exploit previous experiences. The remainder of this section *briefly* introduces these concepts, which we will describe in detail in the subsequent sections.

Path Representation. We describe an experienced path as an ordered list of its poses that captures its structure. We call these poses *attractors*. Fig. 1 depicts an example of attractors, details are in Sec. 4.

Situation Description. We define a similarity relationship among tasks to exploit previous experiences using *situation descriptors*. We define different descriptors at global and local level to capture their functions and properties. See Sec. 5 for further details.

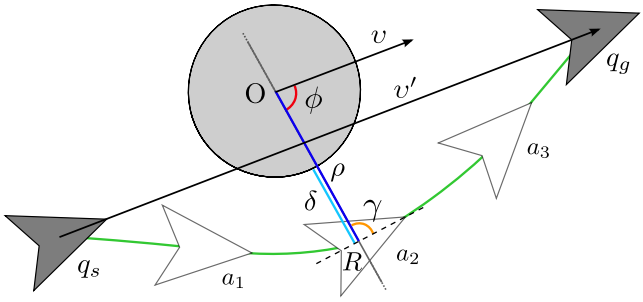


Figure 1: Attractor representation of a local path $P = \{q_s, a_1, a_2, a_3, q_g\}$. (δ, ϕ, γ) identify a local attractor a_i.

Planning Exploiting Experiences. We employ a planner inspired by the Attractor Guided Planner [13]. It guides the planning process to prefer trajectories that match previous experiences. As described in detail in Sec. 6, distinct planning instances run in parallel at global and local level.

Planning in Dynamic Environments. We consider a probabilistic approach based on Gaussian Processes [29] to predict obstacle trajectories. Our system define virtual constraints based on these predictions and plans deviations to avoid dynamic obstacles. We provide details in Sec. 7.

4. Path Representation

We use a path representation that effectively stores and retrieves previously experienced paths. We represent a path P as $P = \{q_s, a_1, \ldots, a_n, q_g\}$, where q_s and q_g are the start and goal poses and $a_1, \ldots, a_n$ are the poses along the path that allow to capture its structure. We call these poses *attractors*, recalling the concept introduced by Jiang and Kallmann [13]. Given an experienced path, we compute its attractors and store them in a database, so that they can be exploited for planning for a new similar task. In Sec. 6, we will describe how our planner takes advantage from this representation.

We compute the attractors of a path by considering iteratively a window of path points. We initialize the window with the first two path points. If a line fits through the points in the window, we insert the successive point. Otherwise, we identify the last inserted point as candidate attractor and check whether a straight motion from the previous attractor is valid and collision free. If this is the case, we include the candidate attractor to the list of attractors, and reinitialize the window. Otherwise, we select the previous point in the path as candidate attractor and perform a new check. We iterate this procedure until it processes all path points. The result is an ordered list of poses describing the path.

Global and local levels of our system rely on different assumptions, so it is necessary to define a different way to store and to reuse the attractors. We maintain the experiences in two distinct databases: D_G for the global paths and D_L for the local ones. As global experiences do not generalize across different environments, we represent the

attractors of the global paths in (x, y, θ) map coordinates and store them in a distinct database for each environment. On the contrary, to make local experiences available across different scenes, the local level should not depend on the environment. To this end, we apply a coordinate transformation to the attractors that makes them independent from the world frame and that can be inverted in different situations for local replanning. We introduce a local coordinate system based on the local path and on the corresponding obstacle. It is the polar coordinate system identified by the axis υ in Fig. 1. It has the pole in the center of the obstacle O, and the orientation of the vector υ' that connects the start to the goal. In this frame, the coordinates (ρ, ϕ, γ) identify unambiguously the attractors, where ρ is the distance of the attractor to O, ϕ and γ are the angles that the line passing by O and the center of the robot R form with υ and the robot axis. As this representation relies only on local information, it allows for transferring experiences across different environments. To further generalize over different obstacles, we consider δ instead of ρ, where δ is the distance of the attractor from the obstacle surface along ρ. The coordinates (δ, ϕ, γ) allow for reusing an experienced local behavior even in the cases in which the obstacle encountered by the robot has different dimension, while keeping a safe distance from it. To exploit an experienced local path for a new task, we transform local attractors in the global frame of the current environment and use them to guide the new plan.

5. Situation Description

To reproduce previous experiences, it is important to identify which of these fits a new situation. We compare navigation tasks using *situation descriptors*. A situation descriptor is an array of vectors that describes the task and the scene in which the robot will perform it. For each experienced path, we compute the corresponding situation descriptor and store it in the database together with the attractors. We measure the similarity between two tasks by computing the sum of the Euclidean distances of the attributes of their situation descriptor.

As stated by Jetchev and Toussaint [11], the ability to generalize experiences to new tasks depends on how we describe the situation. We specify two distinct situation descriptors to describe global and local tasks that fit the objectives and the assumptions considered at each level.

As the global level depends on the environment, we define a situation descriptor as the array composed by the start and goal poses of the task expressed in the map frame. Given the task to navigate from $q_s = (x_s, y_s, \theta_s)$ to $q_g = (x_g, y_g, \theta_g)$, the corresponding situation descriptor is

$$d_G = [x_s, y_s, \theta_s, x_g, y_g, \theta_g]. \quad (1)$$

This descriptor fits the requirements at global level: it allows for multiple queries in one environment and provides no information across different environments. Jiang and

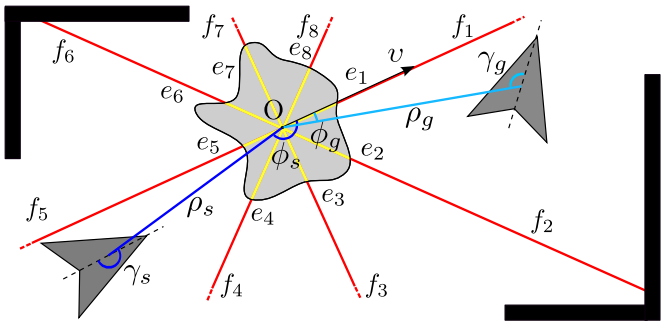


Figure 2: Local situation descriptor defined by the local start (dark blue) and goal (light blue), the extent of the obstacle (yellow) and of free space around it (red).

Kallmann [13] show that this descriptor is effective in practice for static environments. Combining this definition of situation descriptor with the representation of paths as attractors, we can consider at global level every sub-path as a distinct example. We store each attractor individually in the database with a reference to the path it belongs to. Given a new task, we search the database for the pair of robot poses (q_i, q_j) such that q_i and q_j belong to the same path P and the sum of the distances to the start and goal configurations of the new task is the smallest. In this way, a new plan can be guided by any of the sub-paths of P.

At local level, we want to generalize experiences across different environments and obstacles. As a result of that, we define a situation descriptor that relies only on local information, as illustrated in Fig. 2. First, we consider the local task expressed in the local coordinates (ρ, ϕ, γ) introduced in Sec. 4:

$$d_L{}^{task} = [\rho_s, \phi_s, \gamma_s, \rho_g, \phi_g, \gamma_g]. \quad (2)$$

We consider the distance from the obstacle center γ instead of the distance from the surface of the obstacle δ used for representing local attractors, as γ is straightforward to compute and δ does not provide any further information to describe a task. This descriptor represents the task with respect to the obstacle, but provides no information about the obstacle itself and the space around it. The geometry of the obstacle and of the space around it are fundamental to plan a path to avoid it. Therefore, we consider two additional components. The first one describes the shape and dimension of the obstacle by computing the extent of the obstacle from its center O in the 8 directions defined by the υ axis, each rotated by 45 degrees, see Fig. 2:

$$d_L{}^{obstacle} = [e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8]. \quad (3)$$

Considering this component, we favor the robot to accomplish similar behaviors at obstacles with similar shapes and dimensions. In fact, these obstacles are likely to correspond to similar objects and the user may want the robot to accomplish a specific behavior to avoid a class of objects. For example, to avoid moving or potentially moving obstacles the user may prefer a behavior that does not cross their front or their way. The second additional component describes the free space around the obstacle as the

extent of free space from the obstacle surface to the next obstacle along the same directions considered in ${d_L}^{obstacle}$:

$$ {d_L}^{freespace} = [f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8], \quad (4) $$

see Fig. 2 for an illustration. The geometry around an obstacle is a good indicator for deciding on which side of the obstacle the robot should pass to avoid it. Therefore, considering this component will favor paths in the same homotopy in tasks with similar geometry around the obstacle. The resulting local descriptor is

$$ d_L = [{d_L}^{task}, {d_L}^{obstacle}, {d_L}^{freespace}] \quad (5) $$

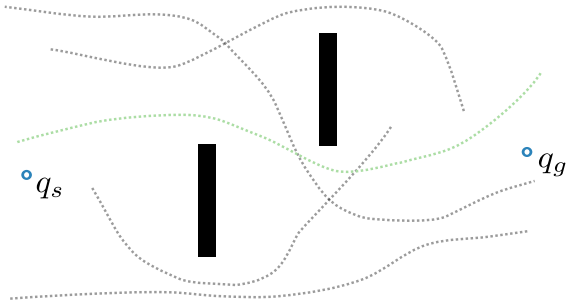
that allows for comparing local tasks across different situations, obstacles and environments. We combine different measure of similarity to keep the local descriptor as general as possible. Nevertheless, each component can be parametrized to obtain a descriptor biased towards a specific aspect of a local task, for example, to favor similar behaviors at similar obstacles.

6. Planning Exploiting Experiences

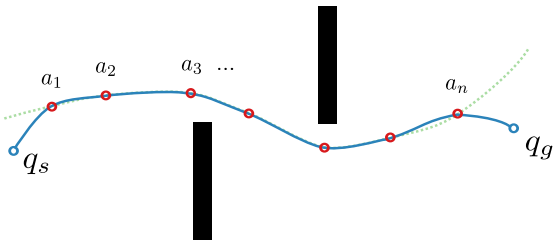
We introduce a path planning algorithm that exploits robot's previous experiences to plan for new tasks. It takes advantage of the notions of *attractors* to represent the experienced paths, and *situation descriptor* to identify similar tasks. Consider the use case introduced in Sec. 3.1 and assume that the operator requires the robot to perform the task illustrated in Fig. 3: navigate from q_s to q_g in the known environment M, where the robot has already successfully performed some tasks (dotted lines). Our system takes as input: the database of global paths experienced in M, $D_{G,M}$, the database of local paths D_L, the start q_s and the goal q_g poses.

6.1. Main

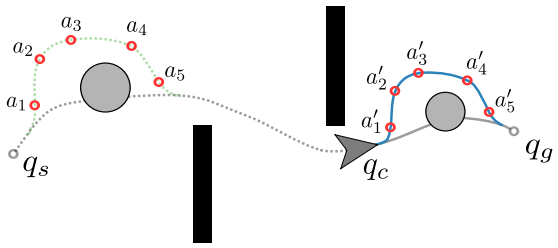
Alg. 1 describes the main procedure of our system. The method PlanGlobal computes a path in the static map (line 1). Once a path is available, the robot starts navigating along it. At each step of the path, we check the remaining poses for invalid configurations (line 4). If an unforeseen obstacle blocks the path at some point (line 5), PlanLocal plans a deviation that enables the robot to avoid the blocking obstacle and to get back to the global path (line 6). We update the path with the local deviation (line 7) and the robot safely navigates towards the next path pose (line 8). We repeat this procedure until the robot reaches the goal. Once the robot completed the task, we ask the user to rate the robot behaviors (line 10). In the following, we describe in detail how local and global planning work and how we collect user feedbacks.



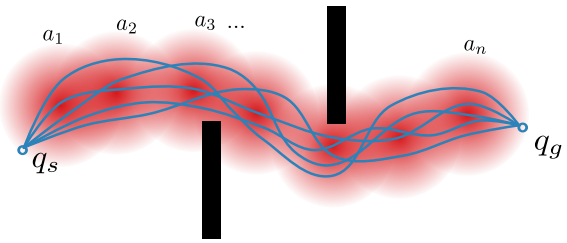
(a) New task and previous experiences (dotted lines).



(b) Global planning guided by the attractors of a similar experience.



(c) Local planning guided by the attractors of a similar experience.



(d) Exploration by relaxing the attractors of a previous experience.

Figure 3: Our planning approach exploits similar experiences at global and local level to plan for a new task.

6.2. Global Planning

The method PlanGlobal (Alg. 2) implements the global level of our system. Given the task to navigate from q_s to q_g and the static map of the environment M, it computes a path from q_s to q_g in M. First, it computes the global situation descriptor d_G corresponding to the task (line 1), as described in Sec. 5. We compare the descriptor d_G with the ones in $D_{G,M}$ to find a previous similar experience. If no similar experience is available, we plan a new path from scratch using standard bi-directional RRT

Alg 1 MAIN($q_s, q_g, D_{G,M}, D_L$)

1: $path_{global} \leftarrow$ PLANGLOBAL($q_s, q_g, D_{G,M}$)
2: $path_{actual} \leftarrow path_{global}$
3: **while** $p_i \in path_{actual} \wedge p_i \neq q_g$ **do**
4: $obstacle, invalid\_poses = \{p_j, ..., p_k\} \leftarrow$ checkPath($[p_i, p_g]$)
5: **if** $invalid\_poses \neq 0$ **then**
6: $path_{local} \leftarrow$PLANLOCAL($p_{j-1}, p_{k+1}, obstacle, D_L$)
7: $path_{actual} \leftarrow$ updatePath($path_{actual}, path_{local}$)
8: navigateTo(p_i)
9: $i \leftarrow i + 1$
10: FEEDBACK($q_s, q_g, path_{actual}, path_{global}, D_{G,M}, D_L$)

Alg 2 PLANGLOBAL($q_s, q_g, D_{G,M}$)

1: $d_G \leftarrow$ globalDescriptor(q_s, q_g)
2: **if** $attr_g \leftarrow$attractorsSimilarTask($d_G, D_{G,M}$) $\neq 0$ **then**
3: $path_{global} \leftarrow$ bi-RRT($q_s, q_g, guided\_sampling, attr_g$)
4: **else**
5: $path_{global} \leftarrow$ bi-RRT($q_s, q_g, uniform\_sampling$)
6: **return** $path_{global}$

Alg 3 PLANLOCAL($q_s, q_g, obstacle, D_L$)

1: $d_L \leftarrow$ localDescriptor($q_s, q_g, obstacle$)
2: **if** $attr_l \leftarrow$ attractorsSimilarTask(d_L, D_L) $\neq 0$ **then**
3: $attr_l \leftarrow$ toMapCoordinates($attr_l, obstacle$)
4: $path_{local} \leftarrow$ bi-RRT($q_s, q_g, guided\_sampling, attr_l$)
5: **else**
6: $path_{local} \leftarrow$ bi-RRT($q_s, q_g, uniform\_sampling$)
7: **return** $path_{local}$

Alg 4 FEEDBACK($q_s, q_g, path_{actual}, path_{global}, D_{G,M}, D_L$)

1: **if** feedback($path_{global}$) = good **then**
2: $d_G \leftarrow$ globalDescriptor(q_s, q_g)
3: $attr_g \leftarrow$ extractAttractors($path_{global}$)
4: storePath($d_G, attr_g, D_{G,M}$)
5: $deviations \leftarrow$ getDeviations($path_{actual}, path_{global}$)
6: **for each** $\{obstacle, path\}$ **in** $deviations$ **do**
7: **if** feedback($path$) = good **then**
8: $d_L \leftarrow$ localDescriptor($q_s, q_g, obstacle$)
9: $attr_{lm} \leftarrow$ extractAttractors($path$)
10: $attr_l \leftarrow$ toLocalCoordinates($attr_{lm}, obstacle$)
11: storePath($d_L, attr_l, D_L$)

(bi-RRT) [18] (line 5). Otherwise, if the robot experienced a similar task as illustrated in Fig. 3a (green dotted line), we retrieve its attractors $attr_g$ (line 2) and exploit them to compute the new path (line 3). In this case, we employ as planner a tailored version of the Attractor Guided Planner (AGP) [13]. It is based on bi-directional RRT and biases its search trees towards a previous experience. The attractors $attr_g = \{a_1, a_2, a_3, ..., a_n\}$ guide the sampling process. We iteratively select one of the attractors $a_i \in attr_g$ in an ordered fashion and sample a pose. For the start tree, we select the attractors from the closest to the start location to the closest to the goal location. For the goal tree, in the opposite order. As a search tree reaches an attractor a_i, we select the next one a_{i+1}. If changes occurred in the environment with respect to the previous experience, one or more attractors might not be reachable. In this case, we sample a new pose according to a dynamically updated Gaussian distribution centered in the unreachable attractor and with covariance growing proportionally to the number of non-valid states sampled around it. We repeat this procedure until we sample a valid state or we reach a maximum number of iterations. If we find a valid state, the new path will exploit a previous experience even if the environment slightly changed. Otherwise, the search continues sampling poses uniformly in the same way as standard bi-RRT. If the planner exploited a previous experience, the new path will reproduce its structure, as shown in Fig. 3b. Furthermore, in this case the planning time decreases as we will show in Sec. 8.1.

6.3. Local Planning

The method PLANLOCAL (Alg. 3) implements the local replanning procedure of our system. We trigger it if the robot encounters an obstacle on its way. It considers the blocking obstacle to compute the descriptor of the local situation d_L (line 1). We use the descriptor to query the database of local paths D_L for a similar experience (line 2). The planning scheme reproduces the one at global level. If no similar previous experience is available to be exploited, we search a new local path using standard bi-directional RRT (line 6). Otherwise, we compute the attractors $attr_l$ corresponding to the previous similar experience and transform them into the current situation (line 3) to guide the sampling process (line 4). If a previous experience guides the planning, the resulting path will reproduce it in the new situation. In the example shown in Fig. 3c, an obstacle blocks the path and we need to plan a deviation. Along its path, the robot has already avoided an obstacle in a locally similar situation. Therefore, we transform the attractors of that experience to the new situation and use them to guide the new plan. As a result, the robot avoids the new obstacle accomplishing a similar behavior to the experienced one.

6.4. User Feedback

The method FEEDBACK enables the user to rate the experienced robot behaviors (Alg. 4). First, we ask him to provide a feedback for the global path (line 1). If the user rates it as *good*, we compute the corresponding descriptor (line 2) and attractors (line 3), and store them in $D_{G,M}$ (line 4). Considering the deviations from the global path generated by local replanning (line 5), the user can also provide a feedback for robot's local behaviors (line 7). If he rates a local path as *good*, we compute the corresponding local descriptor (line 8) and attractors(line 9). At local level, we transform the attractors to local coordinates (line 10) before storing them in D_L (line 11). The experiences stored in this process are then made available to guide the planning for new tasks.

Our system does not need any initial data to work. Without examples, we automatically fall back to bi-RRT performance. We built the databases by storing the good experiences online while the robot performs its tasks. The higher the number of feedbacks provided by the user, the better the system fit his preferences. If the environment changes so that some experiences cannot be exploited for planning anymore, we remove them from the database. In this way, we keep track only of the experiences still valid, and we bound the growth of the database. If required, we can easily define distinct databases for different users such that each database contains only the preferences of a specific user.

6.5. Exploration vs. Exploitation of Experiences

Our planning approach focuses the search for a new path as close as possible to a similar previous experience. This allows for reproducing behaviors and generating paths that are predictable to the user. It comes however at the cost of exploring only a limited portion of the configuration space if a similar path exists.

In case an exploration towards new path is desired, we can give to the user the possibility to switch to an exploration mode. Two options are available to explore the space. First, *full exploration* does not consider any previous experience but plans new paths using standard bi-RRT that probabilistically covers the whole configuration space. Second, *attractor relaxation* relaxes the constraints imposed by the attractor by increasing the covariance around one, some, or all attractors during the sampling process in planning. One example in which we relaxed all of the attractors of an experience is illustrated in Fig. 3d. Similar paths can be generated by relaxing the attractors that still maintain some form of similarity to the original experience.

The user can evaluate such paths in simulation and, once he finds a path that satisfies his preferences, we replace the experience in the database with the new path. Attractor relaxation does not explore the whole configuration space but allows the user to find an optimal behavior according to his preferences starting from an experienced path.

7. Planning in Dynamic Environments

In the scenario we envisioned in Sec. 3.1, the robot might share the workspace with other moving entities such as persons, other robots, etc. To deal with dynamic obstacles, some path planners explicitly consider time information [5, 14]. If the state space of the planner contains a time dimension, planning becomes more complex and the planning time grows. This often results in slow reactions of the robot in the presence of an obstacle and, in the worst case, in a collision. We introduce a strategy to use our planning approach to avoid dynamic obstacles by replacing the notion of time by spatial constraints.

In the situation depicted in Fig. 4a, the robot R navigates along the blue dotted path. At the same time, another agent H is moving nearby. To ensure that H will not block the robot's path at some point, we first need to check whether a collision between the two agents may occur. We know robot's path and motion model, so we easily compute its future poses. On the other side, an entity moving on a factory floor can be a human worker, another robot, a forklift, etc. so we cannot make any specific assumption about the trajectory and motion model of H. As H's future motion presents high uncertainty, we employ a probabilistic approach to predict its trajectory. We rely on Gaussian Processes (GPs) [29] to model the trajectory out of the current observation data.

7.1. Gaussian Processes for Modeling Trajectories

A Gaussian Process is a collection of Gaussian random variables defined by a mean function m and a covariance function c. A function $f(x)$ following this distribution is denoted as

$$f(x) \sim GP(m(x), c(x, x')). \tag{6}$$

Given n observations u of f for the input $x \in X$, we can learn the model $GP(m(x), c(x, x'))$ that encodes all of our prior knowledge of it. For a new input x^*, GPs allow for computing the predictive mean μ^* and variance σ^* of $f(x^*)$ as follows:

$$\mu^* = k^\top K^{-1} u \tag{7}$$

$$\sigma^* = c(x^*, x^*) - k^\top K^{-1} k \tag{8}$$

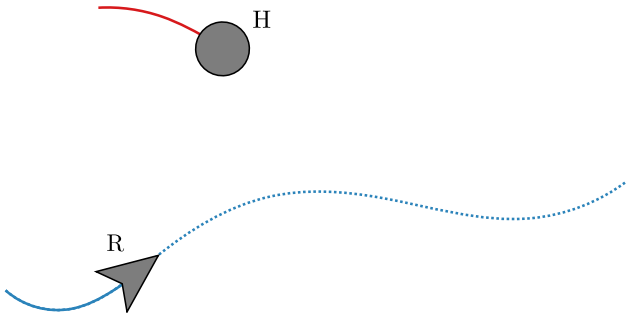
$$K = \begin{bmatrix} c(x_1, x_1) & \dots & c(x_1, x_n) \\ \dots & \dots & \dots \\ c(x_n, x_1) & \dots & c(x_n, x_n) \end{bmatrix}, k = \begin{bmatrix} c(x_1, x^*) \\ \dots \\ c(x_n, x^*) \end{bmatrix}. \tag{9}$$

Many works as [6],[8] and [34] demonstrate that GPs work well to model trajectories thanks to their property to provide Gaussian probability distribution over trajectory poses. Furthermore, approaches using GPs are more flexible than for example an Extended Kalman Filter (EKF) for which an explicit motion model needs to be defined.

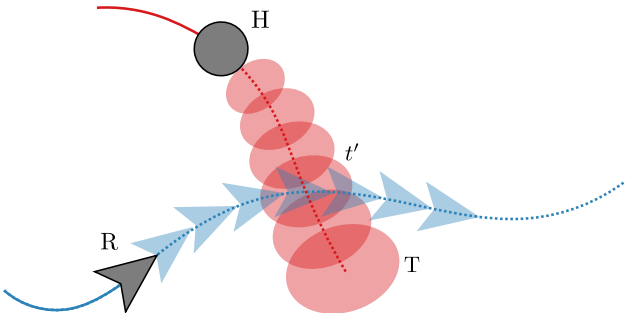
We consider a pair of GPs to model a trajectory. They take as input the time $t = 1, 2, 3, ..., T$ and provide as output the changes along the x and y axes.

$$\begin{aligned} \Delta x_t &= x_t - x_{t-1} \\ \Delta y_t &= y_t - y_{t-1} \end{aligned} \tag{10}$$

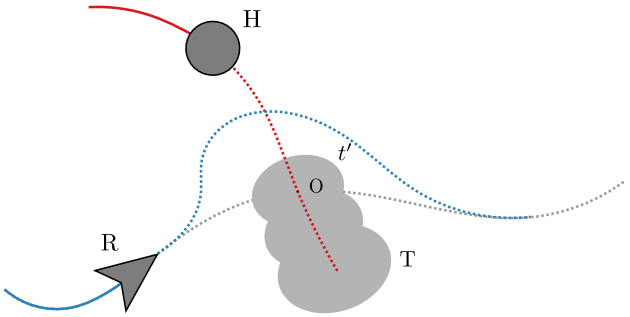
This representation assumes the movements along x and y to be independent. As noise along x and y is correlated, correlation is also induced in the posterior processes. For both GPs, we consider a zero mean function. This is a reasonable assumption as they represent functions of increments. Furthermore, this assumption allows for reducing the number of hyperparameters that define our model. We employ as covariance function the sum of a Matern kernel



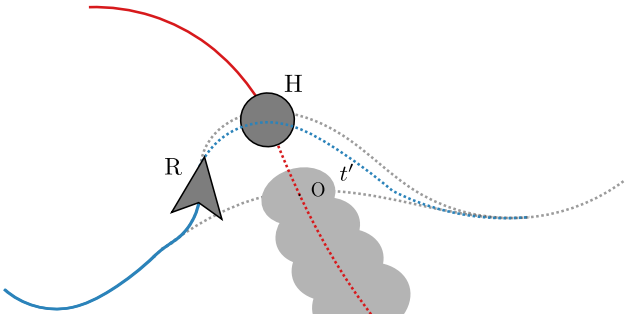
(a) Robot R navigates along the blue dotted path while another agent H moves nearby.



(b) Collision detected at t' by integrating R's future footprints over H's predicted uncertainty areas.



(c) Local replanning considering the gray obstacle defined by the uncertainty areas for $t' \leq t \leq T$.



(d) Deviation update to smoothly converge to the original path.

Figure 4: Planning a local deviation to avoid a collision with a dynamic obstacle.

with $\nu = 5/2$ and a noise kernel:

$$c(x, x') = \sigma_f \Big(1 + \frac{\sqrt{5}(x - x')}{l} + \frac{5(x - x')^2}{3l^2}\Big) \exp\Big(-\frac{\sqrt{5}(x - x')}{l}\Big) + \sigma_n^2 \delta(x - x') \tag{11}$$

where the hyperparameters are the length scale l, the expected variance of the output σ_f and the noise term σ_n. Matern kernel includes a large class of kernels and it is often used in real applications thanks to its flexibility. We preferred it over the squared exponential kernel as the latter assumes high smoothness of the function, which usually does not hold for noisy observations of real trajectories. We train the hyperparameters using the robot's previous experiences. Thus, we exploit general information about moving entities without the need of training parameters of an explicit parametric motion model. As a result, the trajectory model improves over time as the robot performs more and more experiences.

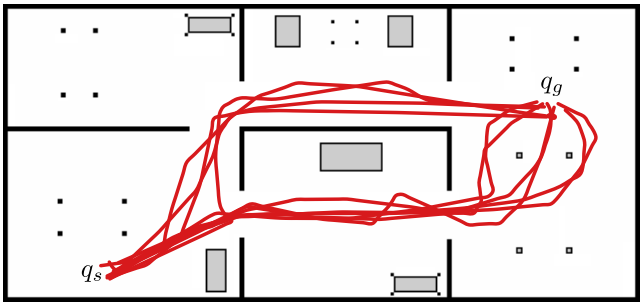
7.2. Future Collision Detection

We use this trajectory model to predict the future poses of agents moving nearby the robot while navigating. In Fig. 4b, we predict the trajectory of H within a time horizon T into the future. The red dotted line is the predicted mean trajectory. We visualize the predicted $2\cdot$ standard deviations confidence intervals in x, y coordinates as a continuous sequence of ellipsoids, which we call *uncertainty areas*. In the figure, the red ellipsoids represent the uncertainty areas generated by the prediction of H's trajectory. To determine whether a collision may occur, we integrate the future footprints of R over the uncertainty areas of H corresponding in time. If there is a significant probability that the robot and the moving agent are in the same area at some time t', we detect a future collision. This is similar to check if the future footprints of R overlaps with the uncertainty area of H. In Fig. 4b, we detect a future collision at $t = t'$.

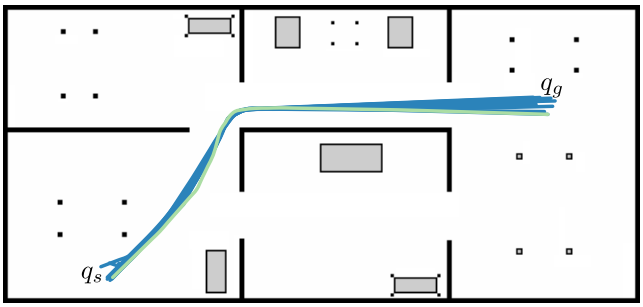
7.3. Replanning

If we detected a future collision, our system needs to plan a deviation from the path to allow the robot to avoid the other moving agent. We achieve this by defining an artificial obstacle composed by the uncertainty areas. As illustrated in Fig. 4c, it is centered in the center of the uncertainty area corresponding to the time of the future collision t', and has area delineated by the union of the uncertainty areas of H for $t' \leq t \leq T$. We exploit such obstacle for local replanning as described in the previous section. So, if a similar experience is available, we transform its attractors in the current situation and use them to guide the planning. The resulting path (blue dotted line) avoids passing by the uncertainty areas corresponding to $t' \leq t \leq T$. This may seem a conservative strategy, however it ensures safety in the time horizon T. Notice that an obstacle defined in this way expands in the direction in which H is moving, so the replanning favors deviations passing on the opposite direction. In Sec. 8, we will demonstrate that this strategy works well in practice.

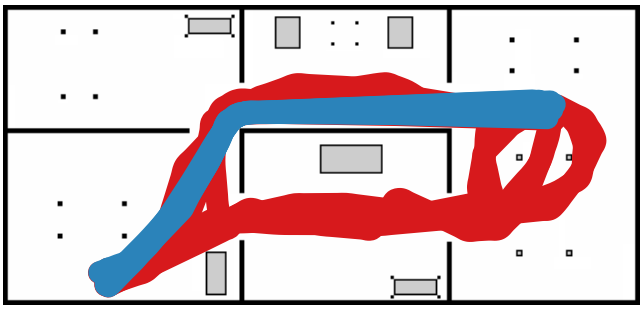
While avoiding a moving obstacle, we update the obstacle and plan the deviation from the global path every time a new observation of its trajectory is received. As the



(a) bi-RRT.

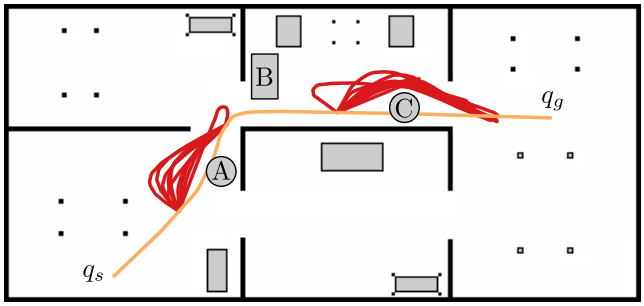


(b) Our planner given the green path as example.

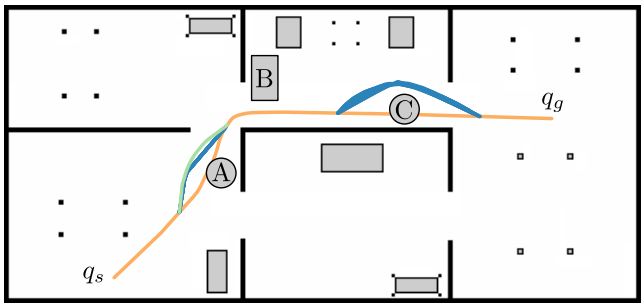


(c) Area covered by a circular robot to navigate along the paths generated by bi-RRT (red) and by our approach (blue).

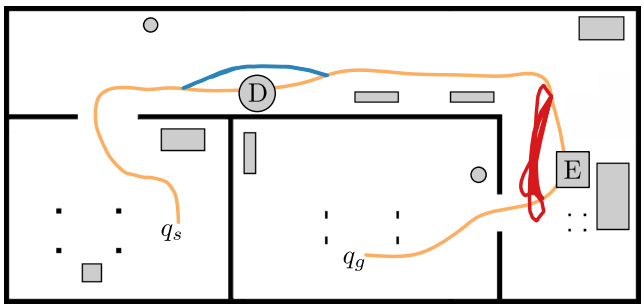
Figure 5: Global paths generated for a set of similar tasks.



(a) bi-RRT.



(b) Our planner given the green path as example.



(c) Our planner given the green path in Fig. 6b as example.

Figure 6: Local paths generated to avoid unforeseen obstacles.

time goes, the uncertainty of the obstacle's trajectory prediction at t' decreases and the corresponding uncertainty area becomes smaller (Fig. 4d). Accordingly, also the deviation from the global path decreases, so that the robot smoothly converges as soon as possible to it.

We apply the same strategy to avoid static obstacles. If an obstacle is not moving, the predicted mean is always equal to the current pose and the variance is zero. Therefore, we can exploit indistinctly local experiences to avoid static and dynamic obstacles.

8. Experiments

This section illustrates and discusses the capabilities and performances of our system. We designed experiments to support the claims made in the introduction. We evaluate our planning approach in Sec. 8.1. Sec. 8.2 focuses on our trajectory prediction model. We test our navigation system in simulation in Sec. 8.3, and we present some tasks performed running our system on a real KUKA Youbot mobile robot in Sec. 8.4.

8.1. Planning

In the first set of experiments, we evaluate the capabilities of our planning approach. Throughout this section, we consider the scenario of the use case introduced in Sec. 3.1 and use the implementation of bi-directional RRT (bi-RRT) available in OMPL as baseline.

To show the ability of our planner to reproduce experiences meeting user preferences, we assume an user requires the robot to perform the 10 navigation tasks represented in Fig. 5. These tasks are similar in a global sense, i.e. they have nearby start and goal poses. Fig. 5a shows the paths generated by bi-RRT for these tasks. Due to the random nature of bi-RRT, they reveal substantial differences to each other: some pass by the top central room, others by the bottom central room. Thus, the user can make only limited predictions about the resulting path for a similar task. He also cannot express any preference about where and how the robot should navigate.

Assume the user prefers that the robot navigates passing by the central top room to perform these tasks. Our system allows him to express this preference by rating the green path in Fig. 5b as *good*. Using this path as exam-

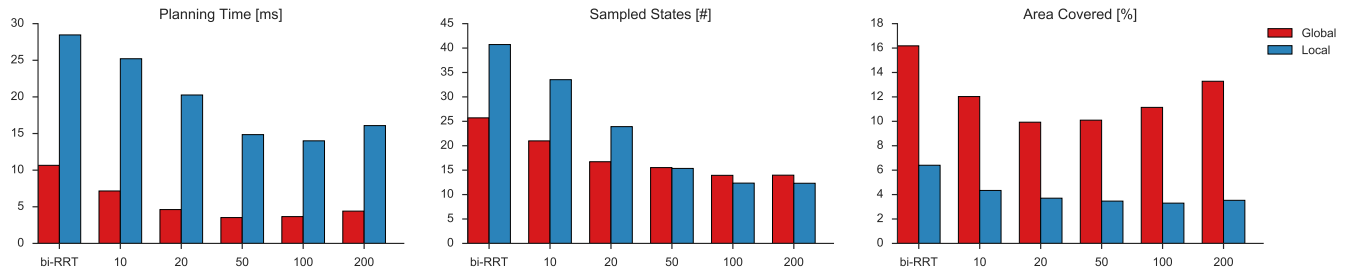


Figure 7: Performance comparison for planning using bi-RRT and our approach with 10, 20, 50, 100 and 200 examples.

ple, our planner generates for the same tasks of Fig. 5a the blue paths in the figure. These paths reproduce the structure of the example meeting the preferences of the user. Therefore, our planner allows for generating similar solutions for similar tasks and, thus, for making the robot behaviors predictable.

Our planner generates paths with such properties also for planning local deviations. Assume the robot is moving along the orange path in Fig. 6a. Along the path, the robot encounters two unforeseen static obstacles A and C that block the path at two different locations. The robot needs to replan to avoid them and to reach its goal. In Fig. 6a, we planned a local deviation 10 times for each obstacle using bi-RRT. The resulting paths cause the robot to behave differently for distinct runs of the same task. Our planner prevents this and it further enables the user to express preferences.

Assume the user rates the green path in Fig. 6b as a *good* example to avoid obstacle A. Our planner generates the blue paths exploiting this path. These paths successfully avoid obstacle A reproducing the example and so meeting the user's preferences. As the local situation at obstacle C is similar to the one at obstacle A, our planner transform the experience at obstacle A in the new situation and generates deviations to avoid obstacle C that reproduce it. This shows that our system can reproduce behaviors and user preferences across similar local situations. This holds also across different environments. Consider the scenario illustrated in Fig. 6c where the orange path is blocked by two obstacles. The situation at obstacle D is similar to the one for which the user provided an example. Thus, our system generates paths to avoid the new obstacle that reproduce the experience in the new situation. The local situation at obstacle E is different and so the example cannot be exploited. In this case, our planner generates new paths from scratch in the same way as bi-RRT.

To illustrate the performances of our planning approach, we considered 20 sets of 10 similar global navigation tasks and 20 sets of 10 similar local tasks. First, we plan for these tasks by using bi-RRT and then by using our planner with 10, 20, 50, 100 and 200 examples randomly selected among the ones generated by bi-RRT. We run this procedure 10 times for a total of 20,000 planning instances. We consider 3 measures to evaluate the performance of the planners: planning time, number of sampled states while planning, area of the environment covered by a circular robot to perform a set of similar tasks. The latter gives a measure of the similarity of the paths generated for similar tasks. The smaller the area covered, the more similar are the paths. Fig. 5c illustrates an example of the area covered by a robot when using bi-RRT (red) and our approach (blue) as planner.

Fig. 7 shows the performance of each planner setting at global and local level. The first chart shows the average planning time. As few examples are available, our system outperforms bi-RRT. The planning time decreases with increasing the number of examples up to approx. the 50% in the local case and the 60% in the global case. When the number of examples becomes large, the planning time tends to marginally increase due to time needed to query the database. The second chart shows the average number of states sampled during planning. If a similar experience is available, our planner tends to sample the corresponding attractors instead of attempting to explore the whole environment as bi-RRT. Thus, the larger the number of examples provided, the smaller is the number of sampled states. In the third chart, we compare the average percentage of area of the environment occupied by a circular robot to perform a set of similar tasks. Even with few examples, the area covered by our system is approx. 25% smaller than bi-RRT at global level and 35% at local level. The results at local level are explained by the ability of our approach to generalize experiences across different obstacles and situations. When the number of examples available for each task increases, the covered area grows accordingly.

8.2. Trajectory prediction

The second set of experiments aims at evaluating our approach to predict the trajectory of moving objects. We want to demonstrate that within a short time it provides good trajectory predictions, and that uncertainty areas captures well the uncertainty of the motion. To this end, we created a setup in our lab in which people could walk through and recorded the trajectories of 6 people walking in it for approx. 120 s each using a motion capture system. We considered 30 s of trajectory of each person to train the hyperparameters of our model. We used the resulting model to predict the future trajectory of the remaining data within the next 5 s every 0.25 s for a total of approx. 2500 predictions. The performance of our trajectory prediction approach are illustrated in Fig. 8. The

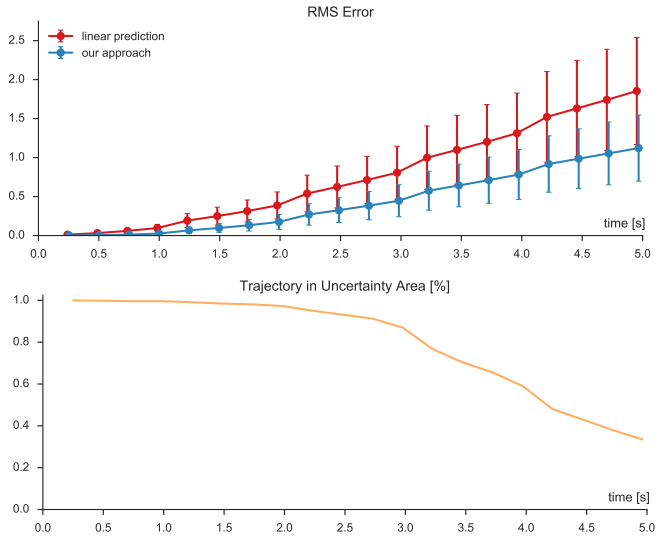


Figure 8: Performance of our trajectory prediction model.

first graph shows the root mean square error of the mean trajectory predicted by our approach with respect to the time predicted. We compare it with the trajectory predicted by a linear constant-velocity model. Our approach always outperforms the linear model and presents half the error for predictions up to 3 s. The second graph illustrates in percentage the number of times that the real trajectory belongs to the corresponding uncertainty area. The percentage is above 90% for predictions up to 3 s. Afterwards, it decreases linearly and at 5 s the real trajectory is in the corresponding uncertainty area around the 30% of the cases. This is partially due to the zero mean assumption made for the Gaussian Processes in our model. It means in practice that for long predicted time the increments Δx and Δy will tend to zero. Still, our approach provides good results for short-time predictions. Furthermore, 3 s is usually a reasonable time to react to an obstacle moving in an indoor environment. We also contrast this effect by continuously updating the prediction and the path while avoiding an obstacle. In the next section, we will show that our approach works well in practice for enabling the robot to avoid dynamic obstacles with foreseeable and safe behaviors.

8.3. Navigation in Simulation

We implemented our system using C++ and ROS, and tested it using V-REP robot simulator. We required a simulated robot to perform navigation tasks in some simulated environments. We illustrate and discuss the performance of our navigation system comparing it to other common approaches. We want to show specially that our system can generate similar robot behaviors for similar tasks.

First, we require the robot to perform 10 sets of 10 similar navigation tasks in some static environments to test our system for robot navigation at global level. We included in the database of global paths one example per set randomly generated using bi-RRT, so that our system

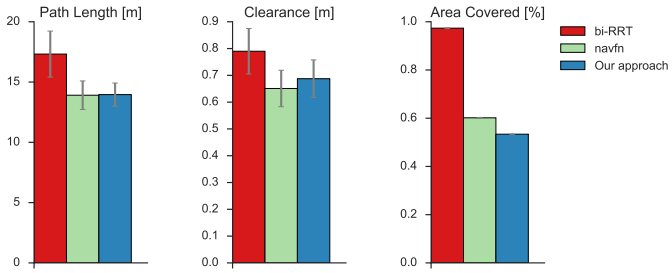


Figure 9: Performance of navigation in static environments.

	Replanning Frequency [Hz]	Obstacle Velocity					
		0.4 m/s		0.8 m/s		1.0 m/s	
		coll.	fail.	coll.	fail.	coll.	fail.
DWA	20.0	0.0	0.0	0.375	0.05	0.25	0.0
TEB	5.0	0.0	0.0	0.075	0.0	0.15	0.025
bi-RRT	20.0	0.0	0.0	0.0	0.0	0.075	0.0
Our appr.	20.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 1: Statistics for dynamic obstacle avoidance.

had some experiences available to exploit while planning. We compared the performances of our system with *bi-RRT* and *navfn* by ROS. *navfn* is a standard approach in ROS for robot navigation based on Dijkstra's algorithm, while we implemented *bi-RRT* by considering our system without any example. We analyze 3 measures to evaluate their performances: executed path length, average clearance during navigation and area covered by the robot to perform a set of similar tasks. Fig. 9 shows the results of our experiments. *navfn* generates in average the shortest paths, while *bi-RRT* presents the largest clearance. Using our system, these measures depend on the examples exploited while planning. A first indicator that our approach allows for providing similar solutions for similar tasks is that it presents the lowest standard deviation for both measures. To further support this, our system covers almost half of the area covered using *bi-RRT*. *navfn* also scores good. It attempts to minimize the path length and, by doing this, the paths for similar tasks tend to pass by the same locations. However, it does not allow to express any preference about which these locations should be.

We evaluate robot navigation at local level by introducing moving obstacles in the environments. We defined six situations that differ from each other for: task, obstacle velocity and trajectory, type of collision that could occur (frontal, lateral, etc.). We compared our system to a reactive approach as Dynamic Window Approach (*DWA*) [7], an approach that consider explicitly time as Time Elastic Bands (*TEB*) [14], and our system with no examples (*bi-RRT*). We used the available ROS implementation of *DWA* and *TEB*. For simplicity, we provided only two examples to our system: one avoiding an obstacle on the left and one on the right. We fixed the global path and performed each task 20 times for every navigation algorithm. We considered five measures to evaluate their performance: executed path length, deviation from the global path, execution time, clearance, and area covered to perform one task multiple times. Fig. 10 shows the results of our ex-

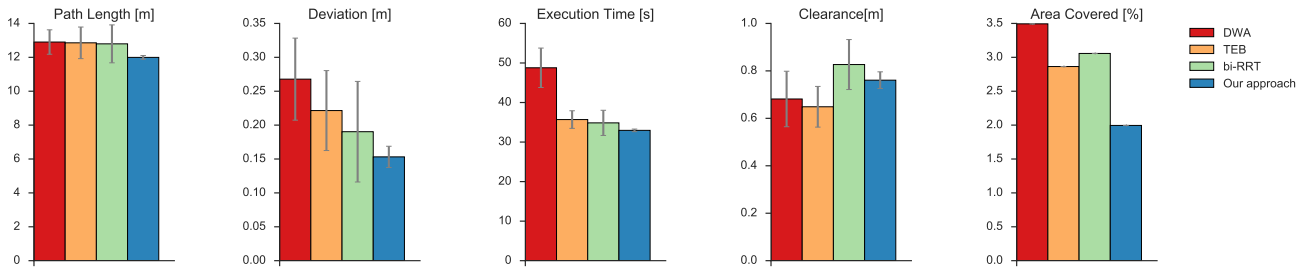


Figure 10: Performance of navigation in dynamic environments.

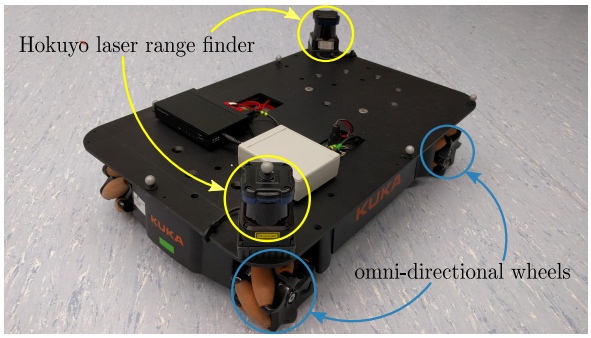
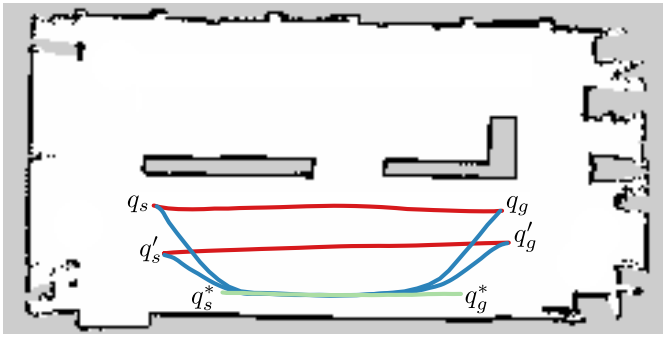


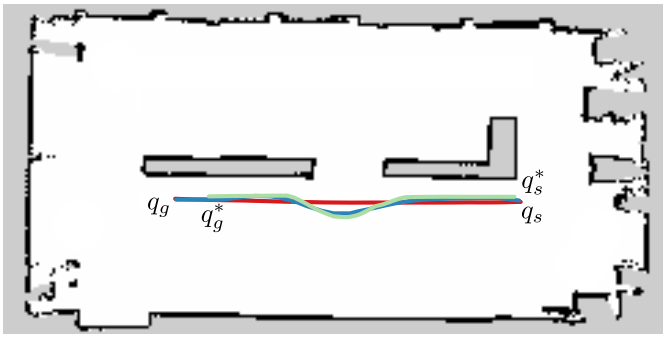
Figure 11: KUKA Youbot.

periments. Our system provides in average the shortest path length. It continuously updates the local path while avoiding an obstacle, and so it converges as soon as possible to the global path as shown also in the analysis of the deviation from the global path. *bi-RRT* also provides good results in average as it is implemented in our system, but it presents large standard deviation due to its random nature. The results for the execution time reproduce the ones for the path length except for *DWA*. The last chart shows that our system allows the robot to cover the smallest area to perform a task 20 times. This demonstrates its capability to generate similar behaviors for similar situations also at local level.

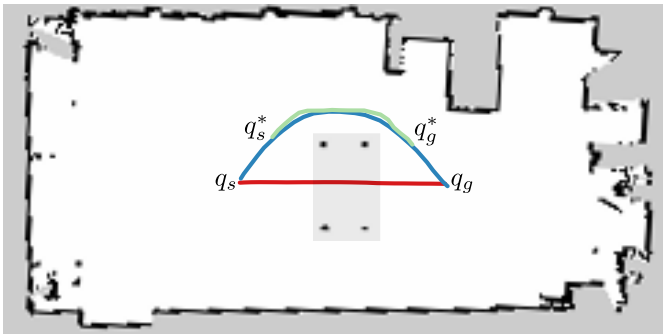
Table 1 reports for each algorithm the replanning frequency, the collision and failure rates with respect to the velocity of the obstacle that we experienced in our experiments. A collision occurs if the robot collides with an obstacle, but it is then able to reach the goal. A failure takes place if the robot is no able to reach the goal within a timeout. As *TEB* plans optimal trajectories considering explicitly time, it plans at a lower frequency than the other approaches. This is reflected in collision and failure rates: when the obstacle is fast, *TEB* does not always react fast enough. Even if replanning at 20 Hz, *DWA* also can cause collisions and failures when the obstacle velocity increases. Our system is safer. If no example is available (*bi-RRT*), it provoked collisions only in one case. Sampling states uniformly may generate completely different paths for successive planning instances. In this case, the robot oscillates and wastes time to react to the obstacle. Instead, we experienced no collision or providing examples to our system.



(a) Navigation in a hallway keeping on the right.



(b) Navigation in a hallway without obstructing the passage.

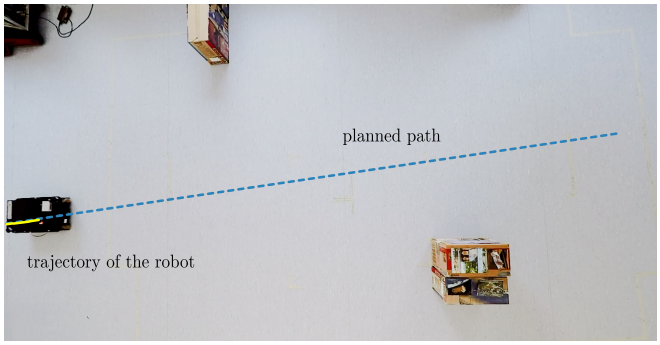


(c) Navigation without passing under the table.

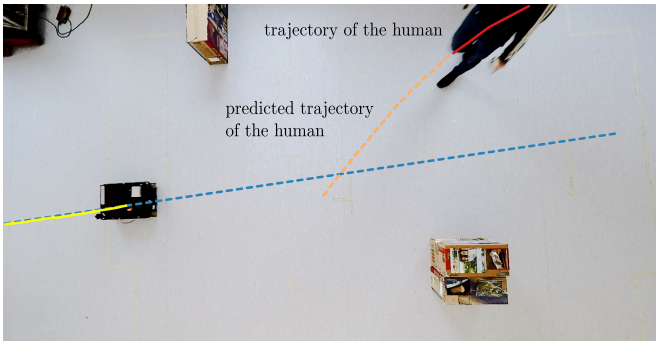
Figure 12: Examples of navigation tasks in real world.

8.4. Real Robot Navigation

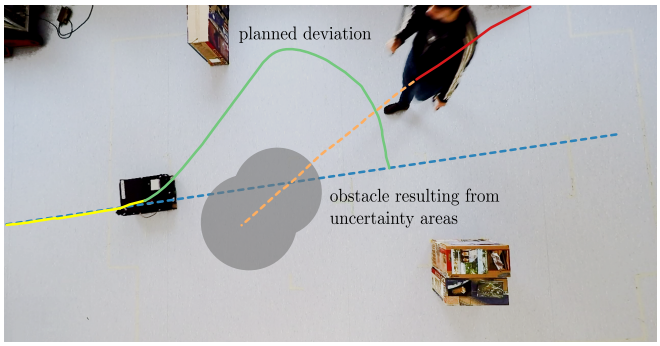
We ran our system on a real robot to demonstrate that our approach works for robot navigation in real world. The platform used is the KUKA Youbot illustrated in Fig. 11. It is an omni-directional mobile robot that we equipped with two Hokuyo laser range finders. Using these sensors, the robot localizes itself in the environment through a Monte Carlo localization approach.



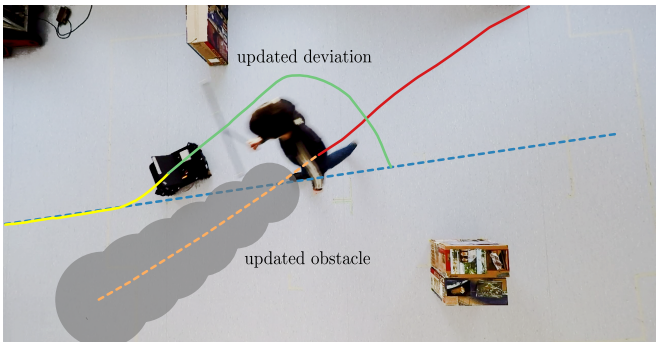
(a) Robot navigates along the planned path.



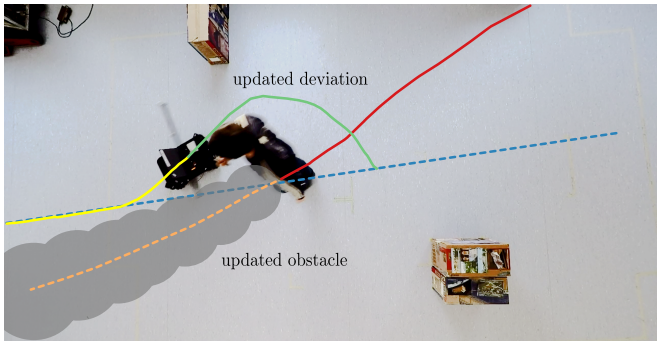
(b) Human walks nearby.



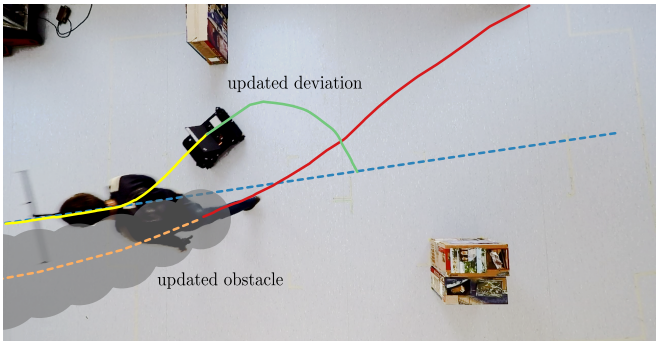
(c) Collision detection and planning deviation to avoid the obstacle.



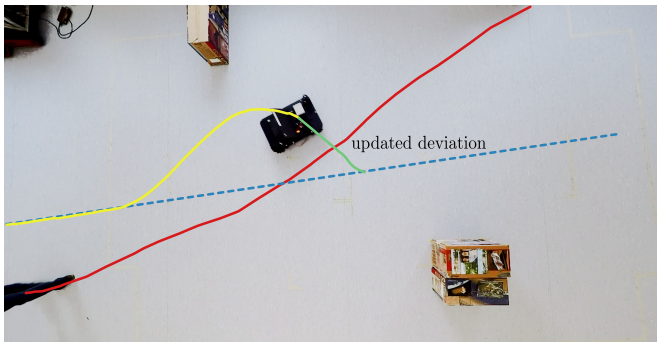
(d) Deviation update (I).



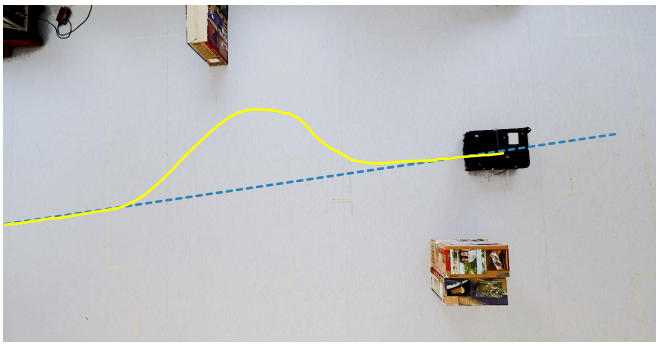
(e) Deviation update (II).



(f) Deviation update (III).



(g) Deviation update and convergence to the original path.



(h) Robot resumes navigation along the original path.

Figure 13: Robot navigates along a given path avoiding a human blocking its way.

We want to show that our system allows the robot for easily performing behaviors that may be hard to encode in typical navigation systems. To this end, we defined 3 simple navigation tasks that an user may require the robot to perform. The first task is illustrated in Fig. 12a and consists of navigating from one side to the other of the hallway by keeping on the right side. If no example is available, the robot just navigates towards the goal (red paths). So, we joysticked the robot along the green path that passes on the right side of the hallway, and gave it as example to our system. Exploiting such example, our system navigates from one side to the other of the hallway by keeping on the right (blue paths). The second task is illustrated in Fig. 12b. The robot has to navigate through

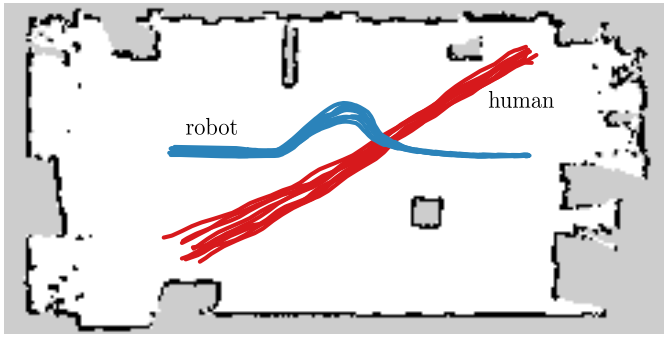


Figure 14: Trajectories of robot (blue) and human (red) for similar situations in which the robot replans to avoid collisions.

the hallway without to obstructing the passage. Without any example, the robot just moves towards the goal (red path) with the risk to block the passage for some time. As we demonstrate the green example, the robot navigates along the blue path to reach the other side of the hallway. In the third task, the robot has to navigate from one side to the other of a table avoid passing under it. Fig. 12c depicts in red the path that the robot navigates if no example is provided, and in blue the path executed when the green path becomes available as example.

We also tested the capabilities of our system to avoid walking people who may block robot's path. We designed different setups in which the robot and a human could move through, and used a motion capture system to observe their trajectories. Fig. 13 illustrates an example in which the robot avoids a human crossing its way using our system. The robot navigates along the blue dotted path (Fig. 13a). The yellow line shows the observed robot's trajectory. In Fig. 13b, a human walks into the scene. The observations of his trajectory are depicted in red, while the orange dotted line represents the predicted mean of his future trajectory. Our system detects a future collision in Fig. 13c that generates the gray artificial obstacle. So, it plans the green local deviation that avoids the human exploiting a previous experience. In Fig. 13(d-f), our system updates the local deviation according to the current the prediction of the trajectory of the human. As the human does not block the original path anymore (Fig. 13g), our system plans paths that allow the robot to converge to it. In Fig. 13h, the robot gets back to the global path and continues its navigation along it. Fig. 14 illustrates the trajectories of the robot and the human recorded for running the same task 10 times. It shows that our system reproduce similar behaviors for similar tasks.

9. Conclusion

In this paper, we presented a robot navigation system that leads to navigation behaviors that are predictable and meet user's preferences. This is especially important for mobile robots operating on factory floors where the operator often requires a robot to follow some criteria. Preferences are implicitly extracted collecting demonstrated examples or feedbacks about robot's previous experiences. Our planning approach reproduces experiences over situations and environments. This allows for accomplishing similar behaviors for similar tasks according to user's preferences. We also introduced a probabilistic approach to predict the trajectories of moving agents that allows a robot for avoiding dynamic obstacles by applying the same planning scheme. We implemented and evaluated our approach over an extensive set of experiments comparing it with different common approaches for robot navigation. We performed tests both in simulation and on a real mobile robot operating in real world. The experiments suggest that our system can reproduce and generalize previous behaviors for robot navigation meeting user's preferences both at global and local level. Furthermore, it allows for easily performing common tasks that may be hard to encode in typical navigation systems.

10. Acknowledgments

This work has partly been supported by the European Commission under the H2020 framework programme under grant agreement no. 645403 (RobDREAM).

References

[1] Bennewitz, M., Burgard, W., Cielniak, G., Thrun, S., 2005. Learning motion patterns of people for compliant robot motion. Int. Journal of Robotics Research 24, 31–48.

[2] Berenson, D., Abbeel, P., Goldberg, K., 2012. A robot path planning framework that learns from experience, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), pp. 3671–3678.

[3] Bruce, J., Veloso, M., 2002. Real-time randomized path planning for robot navigation, in: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp. 2383–2388.

[4] Coleman, D., Şucan, I.A., Moll, M., Okada, K., Correll, N., 2015. Experience-based planning with sparse roadmap spanners, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), pp. 900–905.

[5] Delsart, V., Fraichard, T., 2009. Navigating dynamic environments with trajectory deformation. Journal of Computing and Information Technology 17, 27–36.

[6] Ellis, D., Sommerlade, E., Reid, I., 2009. Modelling pedestrian trajectory patterns with gaussian processes, in: Proc. of the Int. Conf. on Computer Vision (ICCV Workshops), pp. 1229–1234.

[7] Fox, D., Burgard, W., Thrun, S., 1997. The dynamic window approach to collision avoidance. IEEE Robotics & Automation Magazine 4, 23–33.

[8] Fulgenzi, C., Tay, C., Spalanzani, A., Laugier, C., 2008. Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes, in: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp. 1056–1062.

[9] Furgale, P., Barfoot, T., 2010. Visual teach and repeat for long-range rover autonomy. Journal on Field Robotics 27, 534–560.

[10] Furgale, P., Krüsi, P., Pomerleau, F., Schwesinger, U., Colas, F., Siegwart, R., 2014. There and back again-dealing with highly-dynamic scenes and long-term change during topological/metric route following, in: ICRA14 Workshop on Modelling, Estimation, Perception, and Control of All Terrain Mobile Robots.

[11] Jetchev, N., Toussaint, M., 2009. Trajectory prediction: learning to map situations to robot trajectories, pp. 449–456.

[12] Jetchev, N., Toussaint, M., 2010. Trajectory prediction in cluttered voxel environments, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), pp. 2523–2528.
[13] Jiang, X., Kallmann, M., 2007. Learning humanoid reaching tasks in dynamic environments, in: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp. 1148–1153.
[14] Keller, M., Hoffmann, F., Hass, C., Bertram, T., Seewald, A., 2014. Planning of optimal collision avoidance trajectories with timed elastic bands. IFAC Proceedings Volumes 47, 9822–9827.
[15] Kretzschmar, H., Spies, M., Sprunk, C., Burgard, W., 2016. Socially compliant mobile robot navigation via inverse reinforcement learning. Int. Journal of Robotics Research 35, 1289–1307.
[16] Kruse, T., Pandey, A.K., Alami, R., Kirsch, A., 2013. Human-aware robot navigation: A survey. Journ. of Rob. & Aut. Systems 61, 1726–1743.
[17] Kuderer, M., Kretzschmar, H., Burgard, W., 2013. Teaching mobile robots to cooperatively navigate in populated environments, in: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp. 3138–3143.
[18] Kuffner, J.J., LaValle, S.M., 2000. Rrt-connect: An efficient approach to single-query path planning, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), pp. 995–1001.
[19] La Valle, S.M., 2011. Motion planning. IEEE Robotics & Automation Magazine 18, 108–118.
[20] LaValle, S.M., 1998. Rapidly-exploring random trees: A new tool for path planning. Technical Report No. 98-11 .
[21] Lien, J.M., Lu, Y., 2009. Planning motion in environments with similar obstacles, in: Proc. of Robotics: Science and Systems (RSS).
[22] Mazuran, M., Sprunk, C., Burgard, W., Tipaldi, G.D., 2015. Lextor: Lexicographic teach optimize and repeat based on user preferences, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), pp. 2780–2786.
[23] Meriçli, T., Veloso, M., Akın, H.L., 2015. A case-based approach to mobile push-manipulation. Journal of Intelligent & Robotic Systems 80, 189–203.
[24] Nardi, L., Stachniss, C., 2016. Experience-based path planning for mobile robots exploiting user preferences, in: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp. 1170–1176.
[25] Perea Ström, D., Bogoslavskyi, I., Stachniss, C., 2016. Robust exploration and homing for autonomous robots. Journ. of Rob. & Aut. Systems .
[26] Pfeiffer, M., Schwesinger, U., Sommer, H., Galceran, E., Siegwart, R., 2016. Predicting actions to act predictably: Cooperative partial motion planning with maximum entropy models, in: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp. 2096–2101.
[27] Phillips, M., Cohen, B.J., Chitta, S., Likhachev, M., 2012. E-graphs: Bootstrapping planning with experience graphs, in: Proc. of Robotics: Science and Systems (RSS).
[28] Quinlan, S., Khatib, O., 1993. Elastic bands: Connecting path planning and control, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), pp. 802–807.
[29] Rasmussen, C.E., Williams, C.K.I., 2005. Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). The MIT Press.
[30] Ros, R., De Màntaras, R.L., Arcos, J.L., Veloso, M., 2007. Team playing behavior in robot soccer: A case-based reasoning approach, in: Proc. of the Int. Conf. on Case-Based Reasoning, pp. 46–60.
[31] Rösmann, C., Feiten, W., Wösch, T., Hoffmann, F., Bertram, T., 2012. Trajectory modification considering dynamic constraints of autonomous robots, in: Proc. German Conf. on Robotics (ROBOTIK 2012), pp. 1–6.
[32] Sprunk, C., Tipaldi, G.D., Cherubini, A., Burgard, W., 2013. Lidar-based teach-and-repeat of mobile robot trajectories, in: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp. 3144–3149.
[33] Şucan, I.A., Moll, M., Kavraki, L.E., 2012. The Open Motion Planning Library. IEEE Robotics & Automation Magazine 19, 72–82. "http://ompl.kavrakilab.org".
[34] Trautman, P., Ma, J., Murray, R.M., Krause, A., 2013. Robot navigation in dense human crowds: the case for cooperation, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), pp. 2153–2160.
[35] Ziebart, B.D., Ratliff, N., Gallagher, G., Mertz, C., Peterson, K., Bagnell, J.A., Hebert, M., Dey, A.K., Srinivasa, S., 2009. Planning-based prediction for pedestrians, in: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp. 3931–3936.
[36] Zucker, M., Kuffner, J., Bagnell, J.A., 2008. Adaptive workspace biasing for sampling-based planners, in: Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), pp. 3757–3762.