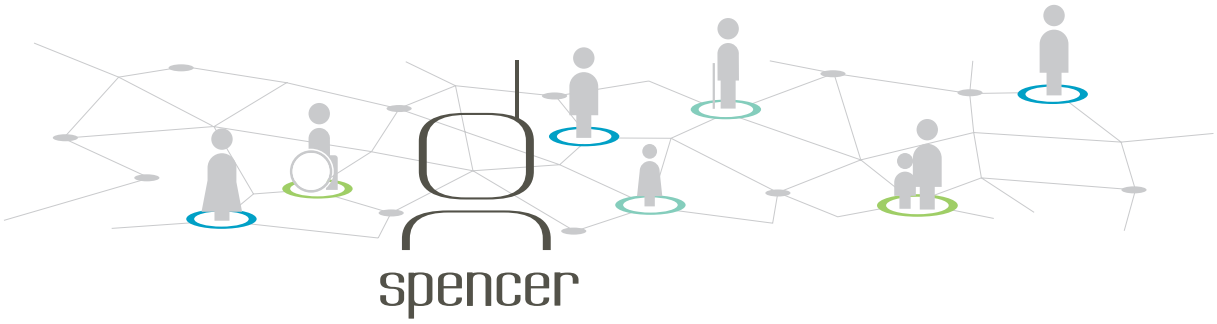


Grant agreement no: FP7-600877

SPENCER:

Social situation-aware perception and action for cognitive robots

Project start: April 1, 2013

Duration: 3 years

DELIVERABLE 5.4

Off- and on-line learning for socially normative task planning

Due date: month 34 (January 2016)

Lead contractor organization: CNRS

Dissemination Level: PUBLIC

Contents

1 Introduction

This deliverable addresses the problem of learning complex normative behaviors for robots (task T5.3) and their on-line adaptation to a variety of situations (task T5.4). Learning complex normative behaviors aims at finding appropriate compact representations of social situations and to react accordingly. In particular, we develop solutions for approaching and engaging complex entities such as a group of person. The level of complexity is specified by the nature of the targeted entity ranging from a single person to a small group of persons (5 to 7 persons).

As in Deliverable D5.1 for simple behaviors learning, we employ Inverse Reinforcement Learning (IRL) algorithms. The issue here is to collect complex demonstrations and then learn a compact representation. Two approaches have been proposed: (i) learn from demonstrations a direct mapping between social situation assessment and robot motion, (ii) learn from demonstrations and then generate cost maps in similar way of Deliverable D5.1. The latter allows to encode specific social constraints, fuse cost maps of different nature such as simple behaviors based, obstacle based. Both approaches have been designed to explicitly allow on-line adaption to social situation assessment (task T5.4).

Section 2 describes of the complexity of complex behaviors. Then, Section 3 presents the procedural steps to get a robot approaching a person given some demonstrations.

2 Complex Behavior

This deliverable targets *socially normative robot behaviors for navigation and non-verbal interaction* which have been described in the DoW:

- Learning to move efficiently and safely through densely crowded spaces by adhering, for instance, to pedestrian traffic social conventions such as walking on that side of the hallway where people move in the same direction as the robot, allowing people to overtake that need to rush, consider people's viewing direction to choose proper avoidance maneuvers, etc.
- Behave in a 'group-friendly' manner. i.e. not cutting through a group or a couple, give wider berth to a family with small children.
- Complying to etiquette rules in pedestrian traffic such as slowing down for elderly people and toddlers, not hassle them for overtaking but looking for a good opportunity to safely pass, leaving priority to elderly people or individuals that carry heavy goods (such as luggage)
- Engaging in interaction with groups through identification of likely spokespersons (through detection of variables such as age, relative rapport and dominance).
- Taking into account human queues such as going around a queue rather than cutting through, or cutting through at an acceptable position in the queue. Learning queueing conventions for joining, standing in and leaving queues.

Simple behaviors have been defined in Deliverable D5.1 as a set of actions that a robot applies in a given social context. *Complex behaviors* can be considered as (i) a combination of simple behaviors or (ii) as a unique block of behaviors. In this line, we frame these approaches as two different schemes: *Combination of Simple Behaviors*, *Block of Complex Behaviors*

2.1 Combination of Simple Behaviors

The combination of simple behaviors builds upon simple behaviors defined in Task T5.1. For example, to reach a pre-defined place, a robot will combine several socially normative behaviors such as navigation, detecting people, engaging and then interacting (Figure 1). The issue here is to explicitly learn the combinations by collecting adequate demonstrations and then employing IRL frameworks.

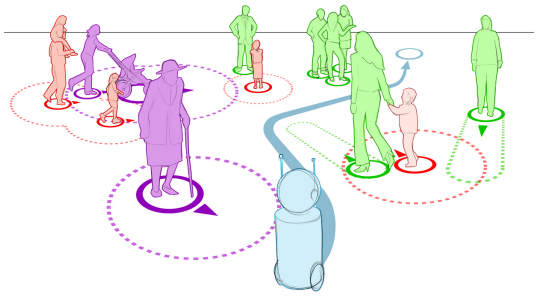
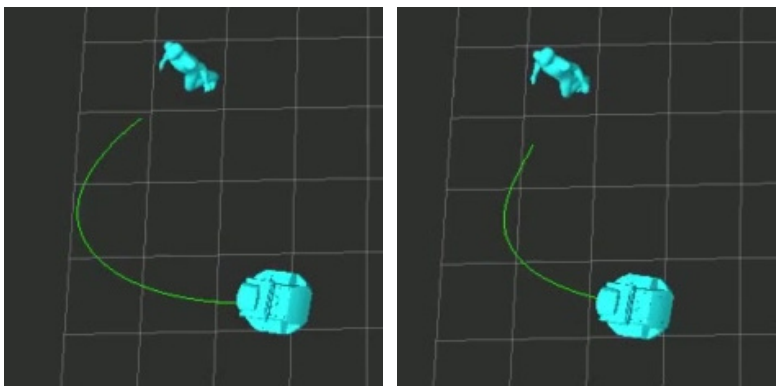


Figure 1: Combination of simple behaviors framework: a robot combines pre-defined simple behaviors such as navigation, perception, engagement and interaction.

2.2 Block of Complex Behaviors

This scheme focuses on situations where robots have to generate behaviors that are considered as a whole. For example, during the approaching scenario (Figure 2), the human and robot continuously adapt their motion. Modeling such a situation requires to explicitly learn adequate behaviors models from demonstrations. The block of complex behaviors approach is required for addressing scenarios such as approaching, engaging and interacting with a group of people.



(a) (b)

Figure 2: Differences in path planning during continuous human and robot motions.

3 Learning complex behaviors

We describe the framework to learn block of complex behaviors (Subsection 2.2) is used by firstly describing the individual case: approaching one person. We extend the models to a small group of person.

3.1 Social constraints during motion planning

In robot navigation, path planners usually minimize time or distance. However, most of the time this is not the case for social paths, because we usually respect the private and social spaces of a person or group of people. This topic is handled by Human Aware Navigation [Kruse et al., 2013]. Several authors [Kruse et al., 2014, Sisbot et al., 2007] have taken into account proxemics as costs and constraints in the path planner to obtain acceptable paths with hard-coded proxemics values derived from social sciences. However, these values are not necessarily relevant for all situations, as they could depend on the velocities of the people, as discussed in [Luber et al., 2012].

Other works also deal with the topic of approaching humans [Satake et al., 2009, Kato et al., 2015], but they focus on tackling the problem of task planning, considering *pedestrians' intentions* such as people approaching a robot. Shomin's work [Shomin et al., 2014] proposes hardcoded waypoints in order to navigate and interact during collaborative tasks. In this work, we focus in the way the robot shall move in order to reach an engagement given previous generated demonstrations.

An Inverse Reinforcement Learning (IRL) method enables to learn the policy using a discrete and finite MDP in which the states are derived from the robot's relative position and orientation with respect to the human. Lately, IRL has been shown being used to teach machines to act as humans do. For example, in the Human Aware Robotics domain, recent works address robot navigation in crowds [Henry et al., 2010, Vasquez et al., 2014]. These examples tackle navigation from point A to point B while avoiding people, not for approaching them. The closest work to ours is [Ramon-Vigo et al., 2014] where they develop a method based on IRL for enabling a robot to move among people and in their vicinity (4mx4m) in a human-like manner.

We specifically address the problem of *approaching people* to interact with them from a distance of several meters. This requires a specific model representing the space around the humans and appropriate trajectories for homing on them. Our work exploits detection and tracking of people (Task 2.1) for the extraction of features such as position, velocity and orientation.

3.2 Models

The problem was handled by learning from demonstrations. Thus, we use inverse reinforcement learning (IRL) which is based on the Markov Decision Processes (MDP). The states and features vary for both planners. The general process summarizing the methodology is depicted next:

1. Definition of the Space-Feature representation
2. Definition of transitions for the MDP modeling
3. Offline learning process given the examples

4. Online process creating the final results

3.3 Experimentation

A simulator that allows to control both robot and human behaviors was developed. Here, positions and velocities can be controlled. Then, it is employed to generate trajectories of robot while approaching humans. The robot is manually controlled during different approaching scenarios.

A set of demonstrations was performed in this experimental platform for the learning process. The path taken by the robot in different positions with different orientations can be seen in Figure 3. This represents the path followed by the robot in the human reference frame.

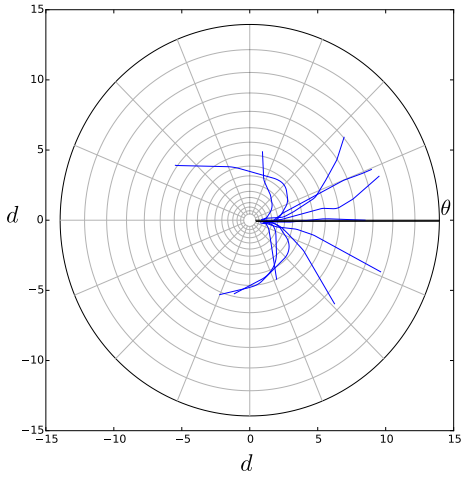
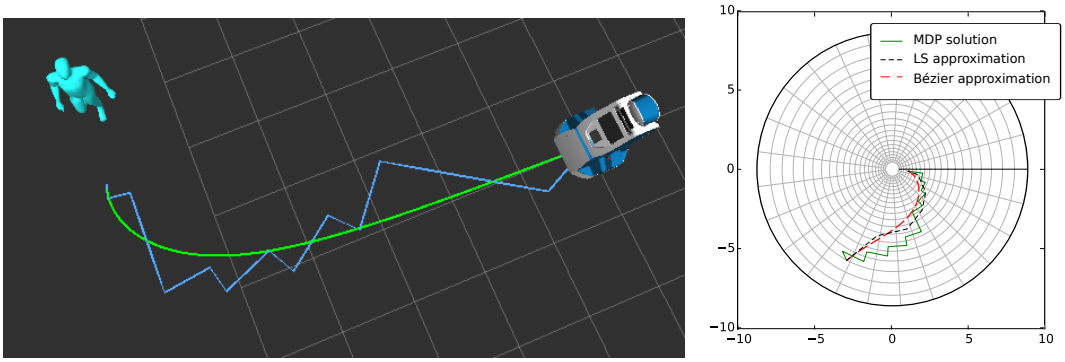


Figure 3: Demonstration of the robot approaching the target person.

3.4 Naive Global Planner

In order to build the state-action vector, first we create a base feature vector based on our number of states S, as follows $\Phi(s) = [\phi_1(s), \phi_2(s), \ldots, \phi_S(s)]$. In which $\phi_i(s)$ is a Kronecker delta function where $\phi_i(s) = [i = s]$ using Iverson bracket notation. In order represent $\Phi(s, a)$, the technique used in [Lagoudakis and Parr, 2003] is applied, creating a feature vector with size of the features is $\Phi(s)$, multiplied by the number of actions. Let's say a is equal to 2, given the possible 5 actions, then $\Phi(s, a) = [\mathbf{0}, \Phi(s), \mathbf{0}, \mathbf{0}, \mathbf{0}]$. Where $\mathbf{0}$ is a zero vector with the size of $\Phi(s)$.

The result of this IRL provides the rewards to the MDP, and by applying the optimal navigation policy in this MDP, the robot moves along the sequence of states which form the optimal trajectory to approach a person. Each state (e.g., cell in the representation described in the previous section) is represented by its center. As a result the trajectory is a discontinuous line as shown in green in Figure 4(b). We hence need to smooth this trajectory taking into account the robot orientation and human orientation. This is developed in next and it is also shown in Figure 4(b). These trajectories are the global plan, nonetheless they do not take into account other constraints such as obstacle avoidance. The result in the simulated scenario is depicted in Figure 4(a).



(a) (b)

Figure 4: **a)** Proposed path to approach the person. Violet line: MDP resolution in a deterministic or the most probable transition case. Green line: fitted curved treated with least squares and Bézier lines.**b)** IRL post-processing. The green line represent the result of the MDP. The black line represents the least square approximation as a parametric function in x and y. The red line is a Bézier curve created from the set of points of this parametric function and the initial orientation of the robot.

3.5 Layered Cost map Navigation

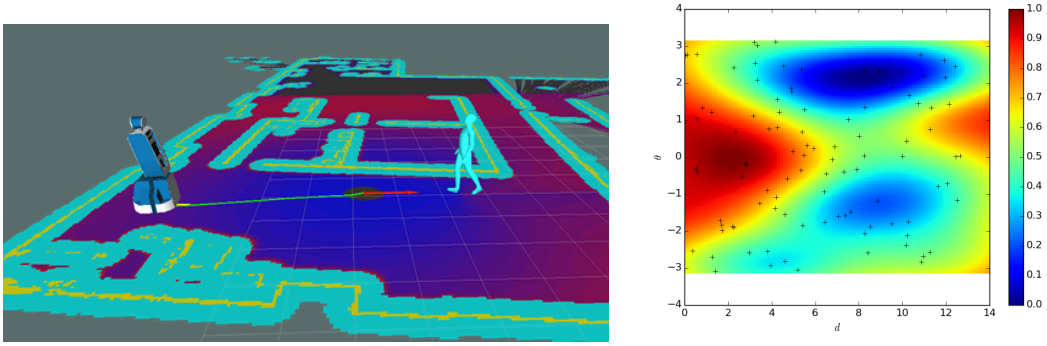
The main difference with the previous case is the use of continuous state features. From this learning, we build a cost map from which navigation is made possible.

Since the states taken into account correspond to the polar human representation, we set n number of random points in the environment within a range for each axis of $r_d = [0, 14]$ and $r_\theta = [-\pi, \pi)$, where r represents range. This draw can be seen as the points in Figure 5(b) and they represent the mean in the 2D gaussian used for the RBF. As for the value of the standard deviation, all RBF bins have the same value which is a quarter of the range for each axis. Thus, the vector state representation is $\Phi(s) = [\phi_1(s_{\text{coord}}), \phi_2(s_{\text{coord}}), \ldots, \phi_n(s_{\text{coord}})]$, where $\phi_i(s_{\text{coord}})$ is the ith RBF and s_{coord} is the cartesian center of the state s. Then we set $\Phi(s, a) = \Phi(s)$ given than it is intended to use this information in a cost map, which is only represented by the states and not the actions, differently from *Naive Global Planner*.

For the Layered Cost map methodology, after the learning process the weight vector w is set. One important point is that $\Phi(s, a) = \Phi(s)$ and s is represented by spatial features. Thus, a cost map can be generated in the environment. Figure 5(b) shows a cost map like result of the demonstrations given in Figure 3, this is feasible due to the representation of features as continuous functions. Even when we have discrete states, the values of the coordinate system is in $\Re$ for distance and angle. The result in the simulated scenario is depicted in Figure 5(a).

4 Group membership modeling

To learn how to approach a group of people, we introduced a group membership metric. The aim is to continuously evaluate the structure of the group including the robot. Figure 6 depicts a robot approaching a group of people where it forms links with each group member. From a set of demon-



(a) (b)

Figure 5: **a)** Layered Cost map Navigation with IRL learned layer. **b)** *Layered Cost map Navigation*: Cost map generated with $w^T\Phi(s)$ in an unfolded polar map. The blue $+$ signs represent the center of all the RBF used in this task.

strations, we optimize the values of this graph. Several approaches have been investigated for the estimation of the graph using relative position or velocity. Of particular interest, we developed a global method exploiting synchrony [Delaherche et al., 2012] of individual behaviors, described by their position and velocity, during a given window analysis. Here, each individual has been modeled as a polygon representing a projection in a 2D plane of the binocular vision in the human field of view ($124°$) limited by the personal distance described in proxemics ($1.2m$). We exploit intersection of these fields of view.

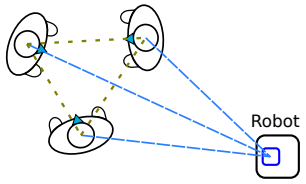


Figure 6: Links of relationship.

To model dynamics of groups such as forming, we employ a forgetting mechanism inspired by the Ebbinghaus forgetting curve [Ebbinghaus, 1913] with some adaptation for the learning of the parameters. The forgetting function is expressed as:

$$R = e^{-\frac{t}{S}} \tag{1}$$

Where R is the memory retention, t is time and S is a relative strength of memory.

Learning is then performed with a forgetting mechanism estimated from the group membership models (i.e. the relationship between people). A demonstration of how learning and forgetting mechanisms are changing during time is shown in Figure 7.

In Figure 8(a) and Figure 8(b) we can see one of the results of this algorithms evaluated in simulation with the social force model [Helbing and Molnar, 1995] and with a manually annotated database [Alameda-Pineda et al., 2015].

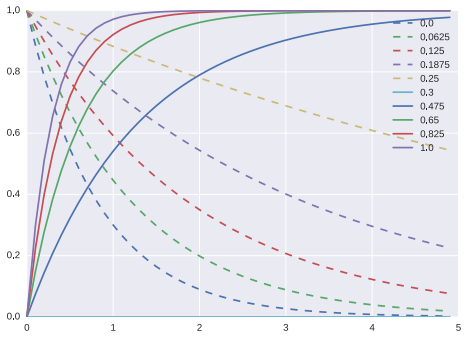
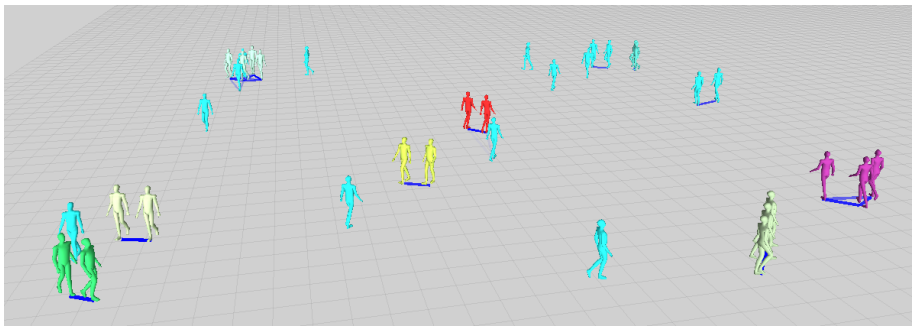


Figure 7: Learning and Forgetting mechanisms for graph modeling with different timing parameters. Dotted lines represent forgetting mechanisms with several different factors and continuous line representing learning with different learning factors.

(a)



(b)

Figure 8: **a)** A frame of this metrics analysis evaluated with the SALSA database [Alameda-Pineda et al., 2015]. **b)** The metrics is tested with a simulated environment based on social forces.

5 Current Work

So far, we developed computational models for learning adequate robot behaviors for approaching and engaging individuals from demonstrations. The framework proposed allows to achieve learning (Task 5.3) and on-line adaptation (Task 5.4). Description and evaluation of two models have been

submitted to IEEE RO-MAN 2016. Currently, we are extending the framework to tackle the problem of approaching and adapting to the dynamics of group of people.

6 Planning considering the learned behaviors

To execute the learned complex behaviors we encode them in cost-maps which could be fed into motion and path planners. Different planning strategies could be used. ALU-FR recently introduced two algorithms [Palmieri et al., 2015, Palmieri et al., 2016] that generate a path or trajectory according to a given cost-map: namely *Theta*-RRT* and *RHCF*.

Theta-RRT* is a fast variant of RRT that, by exploiting a path biasing technique, generates a kinodynamic compliant trajectory over cost-maps.

RHCF is an efficient randomized approach (based on weighted random walks) that finds a set of K paths lying in distinct homotopy classes: having a set of diverse paths (belonging to different homotopy classes) is an appealing strategy to deal with unexpected obstacles and quickly react to dynamic world's changes in robot motion planning.

6.1 Fast Motion Planning with Any-Angle Path Biasing

In [Palmieri et al., 2016] ALU-FR introduces Theta*-RRT. The approach is a variant of RRT that generates a trajectory by expanding a tree of geodesics toward sampled states whose distribution summarizes geometric information of the any-angle path.

Theta*-RRT, see the algorithm in Fig.6.1, first generates a geometrically feasible any-angle path **P** using only geometric information about the workspace. Then, it computes the trajectory by growing a tree τ of smooth local geodesics around path **P** (path-biasing heuristic) satisfying the system's nonholonomic constraints. It repeatedly samples a state x_{rand} mainly from a subspace centered around path **P**. It then makes x_{rand} a new tree vertex and connects it to x_{nearnd}, which is selected among several ones as the vertex that connects with minimum cost (that syntheses the learned behavior) to x_{rand}. It was shown experimentally, for both a differential drive system and a high-dimensional truck-and-trailer system, that Theta*-RRT finds shorter trajectories significantly faster than four baseline planners (RRT, A*-RRT, RRT*, A*-RRT*) without loss of smoothness, while A*-RRT* and RRT* (and thus also Informed RRT*) fail to generate a first trajectory sufficiently fast in environments with complex nonholonomic constraints. Moreover it has been proven that Theta*-RRT retains the probabilistic completeness of RRT for all small-time controllable systems that use an analytical steer function.

6.2 Social Navigation exploiting homotopy classes

In [Palmieri et al., 2015] ALU-FR introduces and shows preliminary results of a fast randomized method, named as *RHCF*, that finds a set of K paths lying in distinct homotopy classes. The path planning task is framed as a graph search problem, where the navigation graph is based on a Voronoi diagram. The search is biased by a the cost function derived from the social force model (or learned off-line that encodes social complex behaviors) that is used to generate and select the paths. The ap-

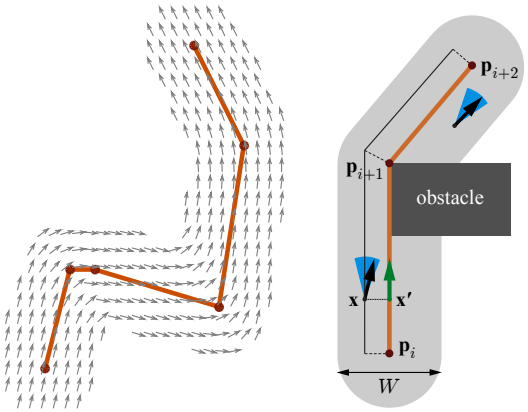


Figure 9: Path-biased sampling strategy. **Right:** Example strip (in grey) around an any-angle path (in **orange**), in which samples are randomly generated. **Black** arrows are samples, and **green** arrows are their projections onto the any-angle path. **Blue** sectors are the angular ranges from which the sample orientations are drawn. The mean sector orientations are computed as weighted averages of the orientations of the any-angle path segments. The individual weight contributions are evaluated in geodesic path coordinates along the offset black line. **Left**: Resulting mean orientations around an example any-angle path

Algorithm 1 *Theta*-RRT*

function Theta*-RRT($\mathbf{x}_{init}$, $\mathbf{x}_{goal}$)
$\mathbf{P} \Leftarrow$ AnyAngleSearch($\mathbf{x}_{init}$, $\mathbf{x}_{goal}$)
if $\mathbf{P} = \emptyset$ **then**
  return *failure*
end if
τ.AddNode($\mathbf{x}_{init}$)
$g(\mathbf{x}_{init}) \Leftarrow \mathbf{0}$
$k \Leftarrow 1$
while $k \leq$ MAX_ITERATIONS **do**
  $\mathbf{x}_{rand} \Leftarrow$ AnyAngleSampling($\mathcal{X}, \mathbf{P}$)
  $\mathbf{x}_{near} \Leftarrow$ NearestNeighborSearch($\tau, \mathbf{x}_{rand}, \mathbf{P}$)
  $\mathbf{u}_{new}, \sigma_{new} \Leftarrow$ Steer($\mathbf{x}_{near}, \mathbf{x}_{rand}$)
  if $\sigma_{new} \in \mathcal{X}_{obs}$ **then**
    continue
  end if
  τ.AddNode($\mathbf{x}_{rand}$)
  τ.AddEdge($\mathbf{x}_{near}, \mathbf{x}_{rand}, \mathbf{u}_{new}$)
  $g(\mathbf{x}_{rand}) \Leftarrow g(\mathbf{x}_{near}) + C(\mathbf{x}_{near}, \mathbf{x}_{rand})$
  if $\mathbf{x}_{rand} \in \mathcal{X}_{goal}$ **then**
    return ExtractTrajectory($\mathbf{x}_{rand}$)
  end if
  $k \Leftarrow k + 1$
end while
return *failure*

Figure 10: Theta*-RRT algorithm.

proach is compared to Yens algorithm, and after an empiric evaluation it was shown that the approach is faster to find a subset of homotopy classes. Furthermore *RHCF* computes a set of more diverse paths with respect to the baseline while obtaining a negligible loss in path quality.

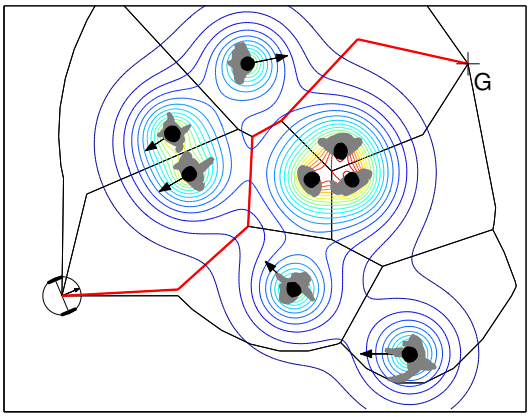


Figure 11: An example path selected from the K best homotopy classes in the Voronoi diagram. The robot is enclosed in the **black circle**, in **red** the path selected to reach the goal position **G**. The **black** Voronoi diagram describes the possible ways to go through a crowd by implicitly encoding different homotopy classes.

References

[Alameda-Pineda et al., 2015] Alameda-Pineda, X., Staiano, J., Subramanian, R., Batrinca, L., Ricci, E., Lepri, B., Lanz, O., and Sebe, N. (2015). SALSA: A Novel Dataset for Multimodal Group Behaviour Analysis.

[Delaherche et al., 2012] Delaherche, E., Chetouani, M., Mahdhaoui, A., Saint-Georges, C., Viaux, S., and Cohen, D. (2012). Interpersonal Synchrony: A Survey of Evaluation Methods across Disciplines. *IEEE Transactions on Affective Computing*, 3(3):349–365.

[Ebbinghaus, 1913] Ebbinghaus, H. (1913). *Memory: A contribution to experimental psychology*. Number 3. University Microfilms.

[Helbing and Molnar, 1995] Helbing, D. and Molnar, P. (1995). Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282.

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

[Kato et al., 2015] Kato, Y., Kanda, T., and Ishiguro, H. (2015). May I Help You?: Design of Human-like Polite Approaching Behavior. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, HRI '15, pages 35–42, New York, NY, USA. ACM.

[Kruse et al., 2014] Kruse, T., Kirsch, A., Khambhaita, H., and Alami, R. (2014). Evaluating directional cost models in navigation. In *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction*, pages 350–357. ACM.

[Kruse et al., 2013] Kruse, T., Pandey, A. K., Alami, R., and Kirsch, A. (2013). Human-aware robot navigation: A survey. *Robotics and Autonomous Systems*, 61(12):1726–1743.

[Lagoudakis and Parr, 2003] Lagoudakis, M. G. and Parr, R. (2003). Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149.

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

[Palmieri et al., 2016] Palmieri, L., Koenig, S., and Arras, K. O. (2016). Rrt-based nonholonomic motion planning using any-angle path biasing. In *Proc. of ICRA 2015, Stockholm, Sweden*.

[Palmieri et al., 2015] Palmieri, L., Rudenko, A., and Arras, K. O. (2015). A fast randomized method to find homotopy classes for socially-aware navigation. In *Proc. of IROS 2015, Workshop on Assistance and Service Robotics in a Human Environment, Hamburg, Germany, http://arxiv.org/abs/1510.08233*.

[Ramon-Vigo et al., 2014] Ramon-Vigo, R., Perez-Higueras, N., Caballero, F., and Merino, L. (2014). Transferring human navigation behaviors into a robot local planner. In *Robot and Human Interactive Communication, 2014 RO-MAN: The 23rd IEEE International Symposium on*, pages 774–779. IEEE.

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

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

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

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

RRT-Based Nonholonomic Motion Planning Using Any-Angle Path Biasing

Luigi Palmieri Sven Koenig Kai O. Arras

***Abstract*— RRT and RRT* have become popular planning techniques, in particular for high-dimensional systems such as wheeled robots with complex nonholonomic constraints. Their planning times, however, can scale poorly for such robots, which has motivated researchers to study hierarchical techniques that grow the RRT trees in more focused ways. Along this line, we introduce Theta*-RRT that hierarchically combines (discrete) any-angle search with (continuous) RRT motion planning for nonholonomic wheeled robots. Theta*-RRT is a variant of RRT that generates a trajectory by expanding a tree of geodesics toward sampled states whose distribution summarizes geometric information of the any-angle path. We show experimentally, for both a differential drive system and a high-dimensional truck-and-trailer system, that Theta*-RRT finds shorter trajectories significantly faster than four baseline planners (RRT, A*-RRT, RRT*, A*-RRT*) without loss of smoothness, while A*-RRT* and RRT* (and thus also Informed RRT*) fail to generate a first trajectory sufficiently fast in environments with complex nonholonomic constraints. We also prove that Theta*-RRT retains the probabilistic completeness of RRT for all small-time controllable systems that use an analytical steer function.**

I. Introduction

Any-angle search is a family of discrete search techniques which, unlike A* or Dijkstra's algorithm, find paths that are not constrained to grid edges. Daniel *et al.* [1] introduce Theta*, an any-angle search technique whose paths are only slightly longer than true shortest paths. The authors show that the basic variant of Theta* finds shorter paths than Field D*, A* with post smoothing and A* on grids, see Fig. 2. Rapidly exploring Random Trees (RRT) [2] is a sampling-based motion planner that expands trees in the state space toward newly sampled states. An optimal variant, RRT* [3], rewires the trees based on the notion of cost. To improve the performance of sampling-based motion planners, recent research has combined them with discrete search techniques [4, 5, 6, 7]. None of these studies, however, combine any-angle search with RRT variants although its properties (such as finding shorter paths than A* with fewer heading changes) are likely beneficial for the performance of the combination.

L. Palmieri and K.O. Arras are with the Social Robotics Lab, Dept. of Computer Science, University of Freiburg, Germany, {palmieri,arras}@cs.uni-freiburg.de. Kai Arras is also with Bosch Corporate Research.

S. Koenig is with the IDM Lab, Dept. of Computer Science, University of Southern California, USA, skoenig@usc.edu.

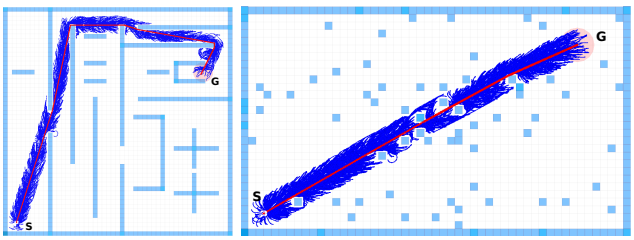


Fig. 1. Theta*-RRT trees in two example environments used in the experiments. **Left:** *Maze* environment. **Right**: *Random map* environment. The trees (in **blue**) grow smoothly towards the goal in a subspace centered around the any-angle path (in **red**).

In this paper, we present Theta*-RRT with the objective of rapidly generating smooth and short trajectories for high-dimensional nonholonomic wheeled robots. Theta*-RRT is a hierarchical technique that combines (discrete) any-angle search with (continuous) RRT motion planning. It improves the efficiency of RRT in high-dimensional spaces substantially by transferring the properties of the any-angle path to the final trajectory. Theta*-RRT considers a continuous control space during planning: It uses steer functions instead of random control propagations to exploit as much knowledge of the nonholonomic constraints of the system as possible and to ensure both high planning efficiency and high trajectory quality. Since heuristics can also be misleading and degrade planning performance, we prove that Theta*-RRT retains the probabilistic completeness of RRT for all small-time controllable systems that use an analytical steer function. We evaluate the approach using a 3D differential drive robot and a 8D truck-and-trailer system. We compare it to four baseline planners: RRT, RRT* (and thus also Informed RRT* [8] which behaves like RRT* until a first trajectory is found), A*-RRT, and A*-RRT* [7]. The evaluation shows that Theta*-RRT is significantly faster and produces shorter high-quality trajectories than those of the baselines.

The paper is structured as follows: We describe related work in Sec. II and Theta*-RRT in Sec. III. We present experiments in Sec. IV and discuss their results in Sec. V. Probabilistic completeness of Theta*-RRT is proven in Sec. VI.

II. Related Work

Prior research has combined discrete search with continuous sampling-based motion planning. For example, Plaku *et al.* [4, 5] propose a planner where a search-based planner finds a sequence of decomposition regions

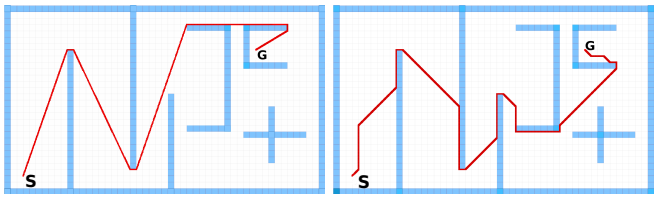


Fig. 2. Comparison of Theta* and A*. **Left:** The any-angle path of Theta* (in **red**) is not constrained to grid edges. **Right**: The grid path of A* (in **red**) is constrained to grid edges and part of a different homotopy class. It is longer and has more heading changes.

that are then used to guide how RRT grows the tree. Bekris and Kavraki propose the Informed Subdivision Tree technique [6] that uses a heuristic to direct the tree growth and improve the coverage of the state space. In contrast to these two planners, Theta*-RRT biases the tree growth mainly in the homotopy class found by Theta* and considers a continuous control space (by utilizing steer functions instead of a discrete set of randomly generated control propagations) to exploit as much knowledge of the nonholonomic constraints as possible. Brunnen *et al.* [7] propose a two-phase motion planner where A* finds a geometrically feasible path, which then biases the tree growth of RRT*. This planner is applied only to a high-dimensional holonomic robot, where the RRT* vertices (sampled from a Gaussian distribution centered around the A* path) are connected using motion interpolation. In contrast, Theta*-RRT focuses on more complex nonholonomic systems and uses steer functions. Cowlagi and Tsiotras [9] propose a planner that constructs a discrete control set using expensive model-predictive control techniques. In contrast, Theta* adopts a continuous control space. Rickert *et al.* [10] propose the EET planner for holonomic systems that sacrifices probabilistic completeness by using workspace information to continuously adjust the sampling behavior of the planner. In contrast, Theta*-RRT is probabilistically complete.

III. Combining Any-Angle Search with RRT

Let $\mathcal{X} \subset \mathbb{R}^d$ be the state space, $\mathcal{U} \subset \mathbb{R}^m$ the control space, and $\mathcal{X}_{obs} \subset \mathcal{X}$ and $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}_{obs}$ the obstacle and free spaces, respectively. A (control) system Σ on state space $\mathcal{X}$ is a differential system such that

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t))\, u + g(\mathbf{x}(t)) \qquad \mathbf{x}(0) = \mathbf{x}_{init}, \qquad (1)$$

where $\mathbf{x}_{init} \in \mathcal{X}$ and, for all t, $\mathbf{x}(t) \in \mathcal{X}$ and $\mathbf{u}(t) \in \mathcal{U}$. g describes the drift, and f describes the system dynamics. Theta*-RRT is a feasible motion planner for small-time controllable nonholonomic systems: It finds controls $\mathbf{u}(t) \in \mathcal{U}$ for $t \in [0, T]$ such that the unique trajectory $\mathbf{x}(t)$ that satisfies Equation (1) connects a given start state $\mathbf{x}_{init} \in \mathcal{X}_{free}$ to a given goal state $\mathbf{x}(T) = \mathbf{x}_{goal} \in \mathcal{X}_{goal} \subset \mathcal{X}_{free}$ in the free space $\mathcal{X}_{free}$.

A. Geodesic Distance for Nonholonomic Wheeled Robots

Let us consider small-time controllable nonholonomic systems.

Definition 1: System Σ is locally controllable from $\mathcal{X}$ if the set of states reachable from $\mathcal{X}$ by an admissible trajectory contains a neighborhood of $\mathcal{X}$. It is small-time controllable from $\mathcal{X}$ if, for any time T, the set of states reachable from $\mathcal{X}$ before time T contains a neighborhood of $\mathcal{X}$.

For small-time controllable nonholonomic wheeled robots, we define the geodesic distance $D_{\mathbf{P}}(\mathbf{x}_1, \mathbf{x}_2)$ of two states $\mathbf{x}_1$ and $\mathbf{x}_2$ to a path $\mathbf{P}$ through $\mathbb{R}^2 \times \mathbb{S}^1$. Consider a path $\mathbf{P}$ and let $\mathbf{x}'_1$ and $\mathbf{x}'_2$ be the orthogonal projections of $\mathbf{x}_1$ and $\mathbf{x}_2$ onto $\mathbf{P}$ and their Euclidean distances be $d_1 = \|\mathbf{x}_1 - \mathbf{x}'_1\|$ and $d_2 = \|\mathbf{x}_2 - \mathbf{x}'_2\|$ (respectively). Then, the geodesic distance $D_{\mathbf{P}}(\mathbf{x}_1, \mathbf{x}_2)$ is the sum of the lengths of the geodesics from each of the two states to path $\mathbf{P}$, that is,

$$\begin{aligned} D_{\mathbf{P}}(\mathbf{x}_1, \mathbf{x}_2) \quad = \quad & w_e\,(d_1 + d_2) + w_\theta\,\left(1 - |\mathbf{q}_{\mathbf{x}_1} \cdot \mathbf{q}_{\mathbf{x}'_1}|\right) \\ & + w_\theta\,\left(1 - |\mathbf{q}_{\mathbf{x}_2} \cdot \mathbf{q}_{\mathbf{x}'_2}|\right) \end{aligned}$$

(for parameters w_e and w_θ), where $\mathbf{q}_{\mathbf{x}_1}$ and $\mathbf{q}_{\mathbf{x}_2}$ are the quaternions of states $\mathbf{x}_1$ and $\mathbf{x}_2$, and $\mathbf{q}_{\mathbf{x}'_1}$ and $\mathbf{q}_{\mathbf{x}'_2}$ the quaternions of the segments of path $\mathbf{P}$ to which $\mathbf{x}'_1$ and $\mathbf{x}'_2$ belong. The geodesic distance of two states is the smaller, the closer they are to path $\mathbf{P}$ in Euclidean distance, heading orientations and steering orientations.

B. Our Technique: Theta*-RRT

Theta*-RRT (detailed in Algorithm 1) first generates a geometrically feasible any-angle path $\mathbf{P}$ using only geometric information about the workspace. Then, it computes the trajectory by growing a tree τ of smooth

Algorithm 1 *Theta*-RRT*

```
function Theta*-RRT(x_init , x_goal)
P ⇐ AnyAngleSearch(x_init , x_goal)
if P = ∅ then
  return failure
end if
τ.AddNode(x_init)
g(x_init) ⇐ 0
k ⇐ 1
while k ≤ MAX_ITERATIONS do
  x_rand ⇐ AnyAngleSampling(X, P)
  x_near ⇐ NearestNeighborSearch(τ, x_rand, P)
  u_new, σ_new ⇐ Steer(x_near, x_rand)
  if σ_new ∈ X_obs then
    continue
  end if
  τ.AddNode(x_rand)
  τ.AddEdge(x_near, x_rand, u_new)
  g(x_rand) ⇐ g(x_near) + C(x_near, x_rand)
  if x_rand ∈ X_goal then
    return ExtractTrajectory(x_rand)
  end if
  k ⇐ k + 1
end while
return failure
```

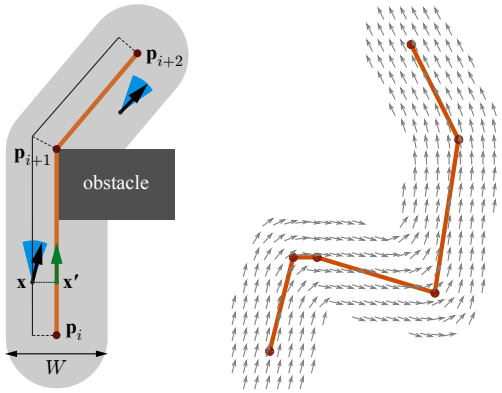


Fig. 3. Path-biased sampling strategy. **Left:** Example strip (in **grey**) around an any-angle path (in **orange**), in which samples are randomly generated. **Black** arrows are samples, and **green** arrows are their projections onto the any-angle path. **Blue** sectors are the angular ranges from which the sample orientations are drawn. The mean sector orientations are computed as weighted averages of the orientations of the any-angle path segments. The individual weight contributions are evaluated in geodesic path coordinates along the offset black line. **Right:** Resulting mean orientations around an example any-angle path.

local geodesics around path $\mathbf{P}$ (path-biasing heuristic) satisfying the system's nonholonomic constraints. It repeatedly samples a state $\mathbf{x}_{rand}$ mainly from a subspace $\mathcal{X}_{local} \subset \mathcal{X}_{free}$ centered around path $\mathbf{P}$. It then makes $\mathbf{x}_{rand}$ a new tree vertex and connects it to $\mathbf{x}_{near}$, which is selected among several ones as the vertex that connects with minimum cost to $\mathbf{x}_{rand}$. The cost depends on the length and smoothness of the trajectory from the candidate tree vertex to state $\mathbf{x}_{rand}$ and the geodesic distance of both vertices to the any-angle path. The subroutines of Algorithm 1 are described below:

`AnyAngleSearch`($\mathbf{x}_{init}$, $\mathbf{x}_{goal}$) uses Theta* to search an eight-neighbor grid from start grid vertex s_{init} to goal grid vertex s_{goal}, where S is the set of all grid vertices. $s_{init} \in S$ is the grid vertex that corresponds to the start vertex $\mathbf{x}_{init}$, and $s_{goal} \in S$ is the grid vertex that corresponds to the goal vertex $\mathbf{x}_{goal}$. We assume obstacles cells to be inflated so as to reflect the robot shape. Theta* uses the consistent straight-line distances as heuristics. It returns an any-angle path $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_N\}$ (a discrete Cartesian path) if one exists and the empty path otherwise.

`AnyAngleSampling`($\mathcal{X}, \mathbf{P}$) samples mainly from a connected subspace $\mathcal{X}_{local} \subset \mathcal{X}$ according to a distribution that conveys geometric information of path $\mathbf{P}$ and returns the sampled state $\mathbf{x}_{rand}$. Concretely, the Cartesian components of the samples are generated uniformly from a strip with width W (for parameter W, called position bias) centered around path $\mathbf{P}$. To convert the any-angle path into a smooth trajectory, the heading orientation x_θ and steering orientation x_δ of the samples are generated uniformly from angular intervals centered around a mean orientation $\bar{\alpha}$, which is a linear combination of the orientations of the segments of path $\mathbf{P}$, that is,

$$\bar{\alpha} = \sum_{i=1}^{N} w_i \, \alpha_{\mathbf{p}}^i, \tag{2}$$

where $\alpha_{\mathbf{p}}^i$ is the orientation of segment $\overline{\mathbf{p}_i\mathbf{p}_{i+1}}$. The weights w_i are calculated from trapezoidal membership functions that are associated with each segment. The functions are centered around the centers of their segments with tails that overlap into the neighboring segments such that their values at the path vertices $\mathbf{p}_i$ are exactly 0.5 and their slopes are no less than a minimal slope δ_S (for parameter δ_S). The influence of each membership function on a given sample $\mathbf{x}$ is computed along geodesic path coordinates, obtained by offsetting path $\mathbf{P}$ with the perpendicular distance of $\mathbf{x}$ to $\mathbf{P}$ (see Fig. 3, left). The orientations x_θ and x_δ of the samples are then generated uniformly from the interval $(\bar{\alpha} - \Delta\theta, \bar{\alpha} + \Delta\theta)$ (for parameter $\Delta\theta$, called orientation bias). The components of the samples that are not related to the workspace (such as velocities and accelerations) are generated uniformly. Moreover with a frequency $f_{uniform}$, this function generates uniformly a sample from the entire $\mathcal{X}_{free}$ (for parameter $f_{uniform}$).

`NearestNeighborSearch`(τ, $\mathbf{x}_{rand}$, $\mathbf{P}$) returns the tree vertex $\mathbf{x}_{near}$ that connects with minimum cost $C(\mathbf{x}_{near}, \mathbf{x}_{rand})$ to state $\mathbf{x}_{rand}$. Instead of determining tree vertex $\mathbf{x}_{near}$ directly, Theta*-RRT determines a set of tree vertices $\mathcal{X}_{near}$ within distance δ_R from $\mathbf{x}_{rand}$ (for parameter δ_R). If this set is empty, it returns the tree vertex nearest to $\mathbf{x}_{rand}$. Otherwise, it returns the tree vertex from set $\mathcal{X}_{near}$ that connects with minimum cost $C(\mathbf{x}_{near}, \mathbf{x}_{rand})$ to state $\mathbf{x}_{rand}$, that is,

$$\mathbf{x}_{near} = \arg \min_{\mathbf{x} \in \mathcal{X}_{near}} C(\mathbf{x}, \mathbf{x}_{rand}) \tag{3}$$

with

$$C(\mathbf{x}, \mathbf{x}_{rand}) = g(\mathbf{x}) + C_\sigma + D_{\mathbf{P}}(\mathbf{x}, \mathbf{x}_{rand}), \tag{4}$$

where $g(\mathbf{x})$ is the sum of the costs from the tree root $\mathbf{x}_{init}$ to the tree vertex $\mathbf{x}$ and $D_{\mathbf{P}}(\mathbf{x}, \mathbf{x}_{rand})$ the geodesic distance of states $\mathbf{x}$ and $\mathbf{x}_{rand}$ from path $\mathbf{P}$. The cost C_σ measures the length and smoothness of the trajectory σ from tree vertex $\mathbf{x}$ to state $\mathbf{x}_{rand}$ returned by the steer function. It is defined as

$$C_\sigma = \sum_{i=0}^{N_e-1} w_d ||\sigma_{i+1} - \sigma_i|| + w_q \left(1 - |\mathbf{q}_{i+1} \cdot \mathbf{q}_i|\right)^2$$

(for parameters w_d and w_q), where $N_e + 1$ is the number of intermediate states σ_i on trajectory σ and $\mathbf{q}_i$ are the associated quaternions. The cost C_σ can be computed on-line or very efficiently with a regression approach [11].

`Steer`($\mathbf{x}_{near}$, $\mathbf{x}_{rand}$) returns controls $\mathbf{u}_{new}$ and a trajectory σ_{new} from state $\mathbf{x}_{near}$ to state $\mathbf{x}_{rand}$ with terminal time T. The analytical steer function connects any pair of states and respects the *topological property* [12], that is, for any $\epsilon > 0$ there exists some $\eta > 0$ such that, for any two states $\mathbf{x}_{near} \in \mathcal{X}$ and $\mathbf{x}_{rand} \in \mathcal{X}$ with

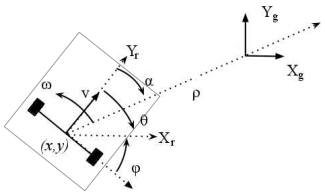


Fig. 4. Differential drive system in polar coordinates: ρ is the Euclidean distance between the Cartesian coordinates of the robot pose (x, y, θ) and of the goal state, ϕ the angle between the x-axis of the robot reference frame $\{X_r\}$ and the x-axis of the goal state frame $\{X_g\}$, α the angle between the y-axis of the robot reference frame and the vector connecting the robot with the goal position, v the translational and ω the angular robot velocity.

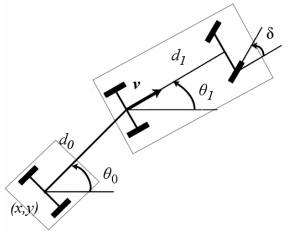


Fig. 5. Truck-and-trailer system: $(x, y) \in \mathbb{R}^2$ are the coordinates of the trailer axle's midpoint, θ_0 and θ_1 the orientations of the trailer and truck, v the translational velocity of the truck, δ its steering angle, d_1 the distance between the front axle and the rear axle of the truck, and d_0 the distance between the trailer axle and the hitch joint on the rear truck axle.

$||\mathbf{x}_{near} - \mathbf{x}_{rand}|| < \eta$, it holds that $||\mathbf{x}_{near} - \sigma_{new}(t)|| < \epsilon$ for all $t \in [0, T]$. If σ_{new} is collision-free, it is added to τ as the tree branch (or edge) that connects $\mathbf{x}_{near}$ to $\mathbf{x}_{rand}$.

IV. Experimental Setup

We now investigate how well Theta*-RRT performs against the baseline planners RRT, A*-RRT, RRT* and A*-RRT*. All planners extend their trees using steer functions. RRT and RRT* sample in the entire state space. RRT uses C_σ as distance metric, and RRT* uses C_σ as cost function. A*-RRT and A*-RRT* sample along A* paths. A*-RRT generates samples and selects the tree vertex that connects to the sampled state with minimum cost in the same way as Theta*-RRT. A*-RRT* generates the samples from a Gaussian distribution centered around the A* path as in [7] and uses C_σ as cost function.

All experiments are carried out with a C++ implementation on a single core of an ordinary PC with a 2.67 GHz Intel i7 processor and 10 GB RAM. The weights of the cost function and the parameters of the distance metric are $w_d = w_e = w_q = w_\theta = 0.5$ and $\delta_S = \delta_R = 4m$. $f_{uniform}$ is set to 1 over 5000 samples.

A. Nonholonomic Systems

We consider two small-time controllable nonholonomic systems, namely a 3-dimensional differential drive system and an 8-dimensional truck-and-trailer system.

Differential drive system: We use a unicycle system with state (x, y, θ), where $(x, y) \in \mathbb{R}^2$ is the Cartesian position and $\theta \in [-\pi, \pi)$ is the heading orientation. After a Cartesian-to-polar coordinate transformation, see Fig.4, the equations of motions are

$$\begin{aligned} \dot{\rho} &= -\cos\alpha \; v \\ \dot{\alpha} &= \frac{\sin\alpha}{\rho} v - \omega \\ \dot{\phi} &= -\omega, \end{aligned} \tag{5}$$

where v and ω are the translational and the angular velocities, respectively. For this system, we use the efficient and smooth steer function POSQ [13]. Width and length of the robot are set to $0.4m$ and $0.6m$, respectively.

Truck-and-trailer system: In order to obtain high-quality trajectories, we use the extended state $(x, y, \theta_0, \theta_1, v, \dot{v}, \delta, \dot{\delta})$, where $(x, y) \in \mathbb{R}^2$ are the coordinates of the trailer axle's midpoint, θ_0 and θ_1 the orientations of the trailer and truck, respectively, v the translational velocity of the truck, $\dot{v}$ its acceleration, δ the steering angle of the truck and $\dot{\delta}$ its derivative, see Fig. 5. The equations of motions are

$$\begin{aligned} \dot{x} &= v\cos(\theta_1 - \theta_0)\cos\theta_0 \\ \dot{y} &= v\cos(\theta_1 - \theta_0)\sin\theta_0 \\ \dot{\theta}_0 &= \frac{v}{d_0}\sin(\theta_1 - \theta_0) \\ \dot{\theta}_1 &= \frac{v}{d_1}\ \tan(\delta). \end{aligned} \tag{6}$$

For this system, we use the η^4 splines [14] as steer function since they are known to generate high-quality trajectories for truck-and-trailer systems. We set $d_0 = d_1 = 1$, width and length of the trailer to $0.4m$ and $0.6m$ and the truck width to $0.4m$.

B. Environments

To stress-test the planners and study how they behave in environments of varying complexity, we design three simulated test environments shown in Figs. 1 and 6. The *maze* environment in Fig. 1 contains many different homotopy classes, has local minima (such as U-shaped obstacles) and narrow passages. Its size is 50m×50m. The *random* environment contains randomly generated square obstacles, its size is 50m×30m. The *narrow corridor* environment in Fig. 6 stresses the ability of the planners to generate smooth trajectories in narrow corridors, its size is 25m×25m. The grid cell size for the any-angle search is 1 m in all environments.

C. Performance Metrics

For each planner and environment, we perform 100 runs for the differential drive system and 50 runs for the truck-and-trailer system. We are solely interested in the first trajectories found. We compute the means and standard deviations of the following four performance metrics for all planning problems that are solved within the planning time limit of 1,000 seconds: tree size N_v (measured in the number of stored tree vertices), planning time T_s (measured in milliseconds or seconds) and resulting trajectory length l_p (measured

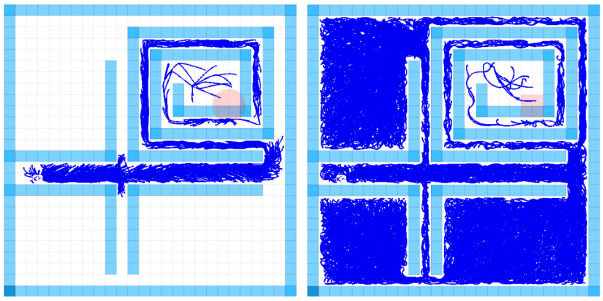


Fig. 6. *Narrow corridor* environment with the goal position (in **red**) and the trees (in **blue**). **Left:** Tree of Theta*-RRT. **Right:** Tree of RRT. Theta*-RRT generates a smaller tree than RRT, which makes Theta*-RRT faster.

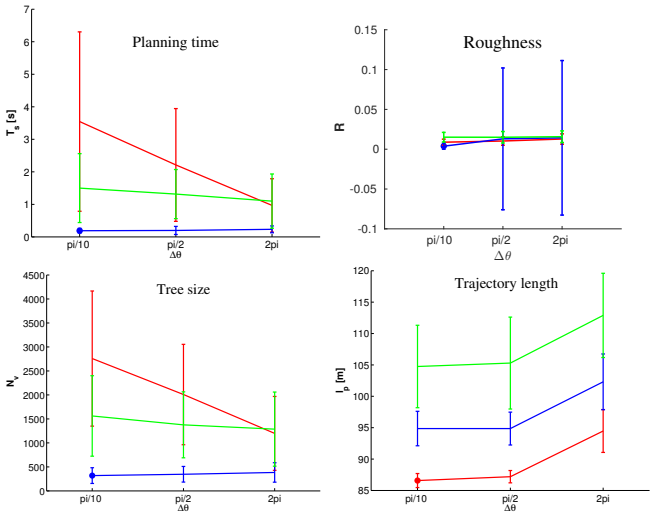


Fig. 7. Performance trends for different strengths of the position bias W ($W = 1m$ in **red**, $W = 4m$ in **blue** and $W = 10m$ in **green**) and orientation bias $\Delta\theta$ in the *maze* environment for the metrics planning time, roughness, tree size and trajectory length (smaller values are better for all performance metrics).

in meters). Smoothness, although an intuitive concept, is less straightforward to assess. In our previous work [11, 13], we used performance metrics based on the velocity profile of the robot (such as the average speed arc lengths, velocity profile peaks or normalized jerk). Here, we use a metric that is better suited for measuring *geometric* trajectory smoothness and thus human-perceived smoothness (namely how sharp the turns are): roughness R, defined as the square of the change in curvature κ of the robot, integrated along the trajectory and normalized by the trajectory length L,

$$R = \int_{t_0}^{t_l} \left| \frac{1}{L} \frac{d\kappa}{dt} \right|^2 dt.$$

A smaller roughness indicates smoother trajectories. We also compute the percentage of trajectories found (problems solved) within the planning time limit.

D. Theta*-RRT Parameters

Prior to the main experiment, we analyze the impact of the parameters W and $\Delta\theta$ on the performance of Theta*-RRT. Position bias W is related to the geometry of the wheeled robot and should be set to a value no less than the maximum value of its length and width. We use the *maze* environment and the ranges $W = \{1m, 4m, 10m\}$ and $\Delta\theta = \left\{\frac{\pi}{10}, \frac{\pi}{2}, 2\pi\right\}$. For each pair of parameter values, we compute the mean and standard deviation of the four performance metrics over multiple runs. Fig. 7 shows the results for the differential drive system. The results for the truck-and-trailer system are qualitatively similar. We observe three trends: *(i)* With a larger orientation and position bias (that is, smaller $\Delta\theta$ and W), the trajectories tend to be shorter and smoother, which is expected since the trajectories then follow the any-angle paths more closely. *(ii)* With a smaller orientation bias (that is, larger $\Delta\theta$), the tree sizes and planning times tend to be smaller. The optimum is at the medium value $W = 4m$ where the value of $\Delta\theta$ has almost no influence (but the optimum is at the smallest value $\Delta\theta = \frac{\pi}{10}$). Given these trends, we select the medium position bias $W = 4m$ and the strong orientation bias $\Delta\theta = \frac{\pi}{10}$.

V. Experimental Results

The experimental results for Theta*-RRT and the four baseline planners are given in Tables I-II. Smaller values are better for all performance metrics. The best values are highlighted in boldface. Theta*-RRT outperforms the four baselines with respect to all performance metrics, with only two exceptions. It is a close second with respect to trajectory smoothness to RRT* for the differential drive system in the *random* environment and to A*-RRT for the truck-and-trailer system in the *narrow corridor* environment. We make the following observations:

(i) The path-biasing heuristic of Theta*-RRT avoids the time-consuming exploration of the entire state space and thus results in small tree sizes and planning times. This advantage comes at the cost of having to find an any-angle path first but Tab. III shows that the runtime of the discrete search is negligible compared to the overall planning time. Theta*-RRT thus has an advantage over RRT and RRT* that explore large parts of the state space, especially in environments with local minima and narrow passages. For this reason, RRT* (and even A*-RRT*) fail to find any trajectory within 1,000 seconds for the high-dimensional truck-and-trailer system in all runs in two of the three environments.

(ii) The path-biasing heuristic of Theta*-RRT results in trajectories that fall into good homotopy classes and are thus short. Theta*-RRT thus has an advantage over A*-RRT and A*-RRT*, whose path-biasing heuristics suffer from the A* paths typically being in worse homotopy classes than the Theta* paths, which results in longer trajectories and thus also larger planning times and tree sizes.

(iii) The sampling strategy of Theta*-RRT results in smooth trajectories. Theta*-RRT thus has an advantage over A*-RRT*, whose sampling strategy is not quite as sophisticated.

Random environment									
Planner	Tree size $\mathbf{N_v}$		Planning time $\mathbf{T_s}$ $[s]$		Trajectory length $\mathbf{l_p}$ $[m]$		Roughness $\mathbf{R}$		Problems solved
Theta*-RRT	54	$\pm$ 59	**0.011**	$\pm$ 0.007	**43.11**	$\pm$ 1.485	0.001	$\pm$ 0.003	100%
A*-RRT	49	$\pm$ 46	0.020	$\pm$ 0.01	44.16	$\pm$ 1.76	0.003	$\pm$ 0.004	100%
RRT	137	$\pm$ 150	0.09	$\pm$ 0.05	65.25	$\pm$ 13.54	0.009	$\pm$ 0.008	100%
RRT*	168	$\pm$ 154	9.57	$\pm$ 13.73	43.84	$\pm$ 1.45	**0.00074**	$\pm$ 0.00140	100%
A*-RRT* [7]	**32**	$\pm$ 34	0.40	$\pm$ 0.93	52.88	$\pm$ 19.0	0.0057	$\pm$ 0.0098	100%
Maze environment									
Planner	Tree size $\mathbf{N_v}$		Planning time $\mathbf{T_s}$ $[s]$		Trajectory length $\mathbf{l_p}$ $[m]$		Roughness $\mathbf{R}$		Problems solved
Theta*-RRT	**319**	$\pm$ 164	**0.19**	$\pm$ 0.07	**94.86**	$\pm$ 2.74	**0.0038**	$\pm$ 0.0038	100%
A*-RRT	1470	$\pm$ 777	3.73	$\pm$ 4.8	98.45	$\pm$ 1.12	0.015	$\pm$ 0.007	100%
RRT	2615	$\pm$ 960	4.85	$\pm$ 4.62	139.16	$\pm$ 21.63	0.018	$\pm$ 0.01	100%
RRT*	658	$\pm$ 37	16.13	$\pm$ 0.34	129.61	$\pm$ 5.65	0.024	$\pm$ 0.01	100%
A*-RRT* [7]	356	$\pm$ 193	47.66	$\pm$ 48.47	96.57	$\pm$ 5.37	0.013	$\pm$ 0.009	100%
Narrow corridor environment									
Planner	Tree size $\mathbf{N_v}$		Planning time $\mathbf{T_s}$ $[s]$		Trajectory length $\mathbf{l_p}$ $[m]$		Roughness $\mathbf{R}$		Problems solved
Theta*-RRT	**1206**	$\pm$ 258	**8.16**	$\pm$ 3.63	**77.78**	$\pm$ 1.44	**0.0027**	$\pm$ 0.004	100%
A*-RRT	1799	$\pm$ 715	18.84	$\pm$ 15.2	77.8	$\pm$ 1.33	0.023	$\pm$ 0.01	100%
RRT	8488	$\pm$ 1639	180.45	$\pm$ 58.55	78.36	$\pm$ 1.82	0.0069	$\pm$ 0.005	100%
RRT*	45310	$\pm$ 7012	2667.5	$\pm$ 481.7	79.47	$\pm$ 0.9	0.03	$\pm$ 0.008	100%
A*-RRT* [7]	3236	$\pm$ 572	309.4	$\pm$ 119.4	78.37	$\pm$ 0.65	0.0125	$\pm$ 0.006	100%

TABLE I

Experimental results: Trajectory quality and planning efficiency for the differential drive system.

Random environment									
Planner	Tree size $\mathbf{N_v}$		Planning time $\mathbf{T_s}$ $[s]$		Trajectory length $\mathbf{l_p}$ $[m]$		Roughness $\mathbf{R}$		Problems solved
Theta*-RRT	**52.2**	$\pm$ 48.3	**0.0547**	$\pm$ 0.0790	**44.331**	$\pm$ 2.8418	**0.0057**	$\pm$ 0.0060	100%
A*-RRT	75.7	$\pm$ 52.4	0.1019	$\pm$ 0.0984	51.74	$\pm$ 7.89	3.4993	$\pm$ 8.8502	58%
RRT	836	$\pm$ 378	1.32	$\pm$ 0.84	66.96	$\pm$ 14.7	2.17	$\pm$ 2.00	100%
RRT*	3957	$\pm$ 2756	816.16	$\pm$ 656.58	52.39	$\pm$ 13.12	0.54	$\pm$ 1.01	76%
A*-RRT* [7]	3582	$\pm$ 3138	949.6	$\pm$ 823.7	49.30	$\pm$ 12.79	0.1013	$\pm$ 0.2647	100%
Maze environment									
Planner	Tree size $\mathbf{N_v}$		Planning time $\mathbf{T_s}$ $[s]$		Trajectory length $\mathbf{l_p}$ $[m]$		Roughness $\mathbf{R}$		Problems solved
Theta*-RRT	**522**	$\pm$ 167	**2.57**	$\pm$ 1.50	**98.59**	$\pm$ 4.95	**1.0073**	$\pm$ 0.7226	100%
A*-RRT	661	$\pm$ 181	4.56	$\pm$ 2.0858	101.79	$\pm$ 8.26	1.1317	$\pm$ 1.0372	100%
RRT	4858	$\pm$ 1276	38.88	$\pm$ 15.83	126.34	$\pm$ 16.52	2.0788	$\pm$ 1.2985	100%
RRT*	0	$\pm$ 0	0	$\pm$ 0	0	$\pm$ 0	0	$\pm$ 0	0% – **failed**
A*-RRT* [7]	0	$\pm$ 0	0	$\pm$ 0	0	$\pm$ 0	0	$\pm$ 0	0% – **failed**
Narrow corridor environment									
Planner	Tree size $\mathbf{N_v}$		Planning time $\mathbf{T_s}$ $[s]$		Trajectory length $\mathbf{l_p}$ $[m]$		Roughness $\mathbf{R}$		Problems solved
Theta*-RRT	**1513**	$\pm$ 492	**20.87**	$\pm$ 13.77	**77.10**	$\pm$ 6.75	2.15	$\pm$ 1.12	100%
A*-RRT	2139	$\pm$ 573	33.46	$\pm$ 16.74	79.66	$\pm$ 5.94	**1.9352**	$\pm$ 0.8722	100%
RRT	1794	$\pm$ 5473	733.98	$\pm$ 438.28	83.77	$\pm$ 7.04	2.31	$\pm$ 1.47	100%
RRT*	0	$\pm$ 0	0	$\pm$ 0	0	$\pm$ 0	0	$\pm$ 0	0% – **failed**
A*-RRT* [7]	0	$\pm$ 0	0	$\pm$ 0	0	$\pm$ 0	0	$\pm$ 0	0% – **failed**

TABLE II

Experimental results: Trajectory quality and planning efficiency for the truck-and-trailer system.

Additionally, we tested Theta*-RRT in a real-world setting by deploying it on a passenger guidance robot for complex and busy airport environments (Fig. 8).

VI. Probabilistic Completeness of Theta*-RRT

The results clearly demonstrate the benefit of Theta*-RRT. However, its path-biasing heuristic – as any heuristic – can mislead and even degrade the performance of RRT, for example when the any-angle path is infeasible to follow under kinodynamic constraints, although a geometric solution (in the inflated grid world) exists. In such cases, the probabilistic completeness, a key property of RRT, is lost. In this section, we prove that Theta*-RRT retains the probabilistic completeness for all small-time controllable nonholonomic systems which use an analytical steer function. Probabilistic completeness is well established for systems with geometric constraints [15] and kinodynamic systems under some strong assumptions (that is, forward simulations [2], uniform sampling and optimal steering [16, 17] and holonomic systems with state-space based interpolation [18]). Our proof follows the one introduced in [2] but we consider a special class of nonholonomic systems, namely systems that are *small-time controllable*, see Definition 1.

Environments	$\mathbf{T_{Theta^*}}$ $[ms]$	$\mathbf{T_{A^*}}$ $[ms]$
Random	5.34	8.07
Maze	12.06	19.91
Narrow corridor	45.14	37.74

TABLE III

Experimental results: Planning times of Theta* and A*

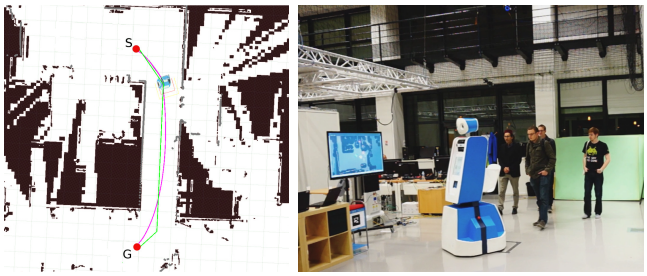


Fig. 8. Theta*-RRT on a real differential drive robot. **Right:** The robot guides a group of people. **Left:** The dots S and G (in **red**) represent the start and goal positions (respectively). The any-angle path (in **green**) is generated first, followed by the smooth trajectory (in **purple**).

Theorem 1: Consider a small-time controllable non-holonomic system. Define a non-zero and non-uniform continuous sampling distribution f_s over $\mathcal{X}_{free}$ generated by the path-biasing technique. Let Theta*-RRT use an analytical steer function that connects any pair of states in $\mathcal{X}$. Then, Theta*-RRT is probabilistically complete since the probability of connecting the start state $\mathbf{x}_{init} \in \mathcal{X}_{free}$ to the goal state $\mathbf{x}_{goal} \in \mathcal{X}_{free}$, if possible, approaches one asymptotically.

Proof: Let $\mathcal{B}(\mathbf{x}_i, \rho)$ denote the ball of radius $\rho > 0$ centered on $\mathbf{x}_i \in \mathcal{X}_{free}$. Consider all the tree vertices $\cup_{i=0,..,k}\ \mathbf{x}_i \in \tau$ at iteration k. Since the volume $\Omega = \cup_{i=0,..,k}\ \mathcal{B}(\mathbf{x}_i, \rho \geq \delta_R > 0)$ is non-zero for the Lebesgue metric the event of sampling a state $\mathbf{x}_{rand} \in \Omega$ will happen with probability one as the number of iterations goes to infinity. Given that the system is small-time controllable, the connection (performed by the *steer* function) of $\mathbf{x}_{rand}$ to $\mathbf{x}_{near}$ (chosen among multiple vertices in τ), will be successful and therefore (if collision free) $\mathbf{x}_{rand}$ will be added to τ. The set $\tilde{\mathcal{X}}_k = \{\mathbf{x} \in \mathcal{X}_{free} \setminus \forall \mathbf{x} \in \tau\}$ represents the uncovered part of the space $\mathcal{X}_{free}$ by τ. By induction following the above property, as k approaches infinity, $\mu(\tilde{\mathcal{X}}_k)$ (the volume of $\tilde{\mathcal{X}}_k$) approaches zero, therefore the state $\mathbf{x}_{goal}$ will be added to τ with probability one. ■

Theorem 1 extends to RRT with any path-biasing heuristic as long as it uses analytical steer functions for systems that are small-time controllable since the proof does not exploit any geometric properties of $\mathcal{X}_{local}$.

VII. Conclusions

In this paper, we introduced Theta*-RRT, a hierarchical technique that combines (discrete) any-angle search with (continuous) RRT motion planning for small-time controllable nonholonomic wheeled robots. We evaluated the approach using two different non-holonomic systems in three different environments and compared it to four different baseline planners, namely RRT, A*-RRT, RRT* and A*-RRT*. The results show that Theta*-RRT finds shorter trajectories significantly faster than the baselines without loss of smoothness, while A*-RRT* and RRT* (and thus also Informed RRT* [8]) fail to generate a first trajectory sufficiently fast in environments with complex nonholonomic constraints. We also proved that Theta*-RRT retains the probabilistic completeness of RRT for all small-time controllable systems that use an analytical steer function.

Acknowledgments

The authors thank Aurelio Piazzi for providing access to the η^4 splines closed-form expressions. This work has been supported by the EC under contract number FP7-ICT-600877 (SPENCER). Sven Koenig's participation was supported by NSF under grant numbers 1409987 and 1319966 and a MURI under grant number N00014-09-1-1031. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies or the U.S. government.

References

[1] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," *Artificial Intelligence Research, Journal of*, vol. 39, no. 1, 2010.

[2] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. Journal of Robotics Research*, vol. 20, 2001.

[3] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, 2010.

[4] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Discrete search leading continuous exploration for kinodynamic motion planning," in *Robotics: Science and Systems (RSS)*, Philadelphia, USA, 2007.

[5] E. Plaku, E. Kavraki, and M. Y. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *Robotics, IEEE Transactions on*, vol. 26, no. 3, 2010.

[6] K. Bekris and L. Kavraki, "Informed and probabilistically complete search for motion planning under differential constraints," in *First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR), Chicago, IL*, 2008.

[7] M. Brunner, B. Bruggemann, and D. Schulz, "Hierarchical rough terrain motion planning using an optimal sampling-based method," in *Int. Conf. on Robotics and Automation (ICRA)*, Karlsruhe, Germany, 2013.

[8] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *Int. Conf. on Intelligent Robots and Systems (IROS)*, Chicago, USA, 2014.

[9] R. V. Cowlagi and P. Tsiotras, "Hierarchical motion planning with dynamical feasibility guarantees for mobile robotic vehicles," *Robotics, IEEE Transactions on*, vol. 28, no. 2, 2012.

[10] M. Rickert, A. Sieverling, and O. Brock, "Balancing exploration and exploitation in sampling-based motion planning," *Robotics, IEEE Transactions on*, vol. 30, no. 6, 2014.

[11] L. Palmieri and K. O. Arras, "Distance metric learning for RRT-based motion planning with constant-time inference," in *Int. Conf. on Robotics and Automation (ICRA)*, Seattle, USA, 2015.

[12] J.-P. Laumond, S. Sekhavat, and F. Lamiraux, *Guidelines in nonholonomic motion planning for mobile robots*. Springer, 1998.

[13] L. Palmieri and K. O. Arras, "A novel RRT extend function for efficient and smooth mobile robot motion planning," in *Int. Conf. on Intelligent Robots and Systems (IROS)*, Chicago, USA, 2014.

[14] F. Ghilardelli, G. Lini, and A. Piazzi, "Path generation using η^4-splines for a truck and trailer vehicle," *Automation Science and Engineering, IEEE Transactions on*, vol. 11, no. 1, 2014.

[15] A. M. Ladd and L. E. Kavraki, "Measure theoretic analysis of probabilistic path planning," *Robotics and Automation, IEEE Transactions on*, vol. 20, no. 2, 2004.

[16] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Int. Journal of Robotics Research*, vol. 21, 2002.

[17] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," in *American Control Conference*, vol. 1, 2001.

[18] S. Caron, Q.-C. Pham, and Y. Nakamura, "Completeness of randomized kinodynamic planners with state-based steering," in *Int. Conf. on Robotics and Automation (ICRA)*, Hong Kong, China, 2014.

A Fast Randomized Method to Find Homotopy Classes for Socially-Aware Navigation

Luigi Palmieri Andrey Rudenko Kai O. Arras .

Abstract— **We introduce and show preliminary results of a fast randomized method that finds a set of K paths lying in distinct homotopy classes. We frame the path planning task as a graph search problem, where the navigation graph is based on a Voronoi diagram. The search is biased by a cost function derived from the social force model that is used to generate and select the paths. We compare our method to Yen's algorithm, and empirically show that our approach is faster to find a subset of homotopy classes. Furthermore our approach computes a set of more diverse paths with respect to the baseline while obtaining a negligible loss in path quality.**

I. Introduction

In socially-aware navigation, like in the case where a robot assists, informs and guides passengers in large and busy airports [1], the motion planner deals with non-static scenarios where crowds of pedestrians and dynamic obstacles regularly invalidate paths generated by using standard approaches (e.g. A* [2], RRT [3], PRM [4]). Each time a path is invalidated, a new motion planning problem has to be solved or in the case of replanning algorithm like D* [5] a repairing phase should recompute a new path by first updating costs over a grid map. Having a set of K precomputed distinct paths, that may be checked for validity in case of the appearance of unexpected obstacles, is a more reasonable approach than solving from scratch the motion planning problem or to replan for each environment's change. Moreover a more rational approach would be to generate K distinct paths from K different homotopy classes. A homotopy class is defined by the set of paths with the same start and goal points which can be continuously deformed into one another without intersecting obstacles.

Different approaches have been already introduced to generate a set of paths belonging to different homotopy classes. Demeyen and Buro [6] introduce a method for efficient path planning that searches on a graph built using constrained Delaunay triangulations. The obstacles are described via polygonal representation. The paths, found in the graph, represent different homotopy classes. Eriksson *et al* [7] find K homotopy classes solving the K shortest paths problem in a two-dimensional environment with polygonal obstacles. They introduce the Kth shortest path map: a map of the entire free workspace, partitioned into equivalence class regions such that the k_{th} shortest path from a start vertex s to any point in a single region has the same structure. Moreover they introduce a simple visibility-based algorithm, based on Yen's algorithm [8], for computing the K shortest paths between two fixed points. Bhattacharya *et al.* [9] propose a method to find different homotopy classes based on A* search over an augmented graph. The graph represents the topological information via the H-signature, a complex analysis value that characterizes a homotopy class, the graph may contain multiple paths to the goal within the same homotopy class. Kuderer *et al* in [10] select *K* best homotopy classes by generating *K* shortest paths using Katoh's algorithm [11]. During navigation the paths feed an optimization algorithm used to generate homotopically distinct kinodynamic trajectories. Among those the best one is selected for the navigation. They show the method is one order of magnitude faster than [9]. Vela *et al* [12] detail a decision support tool to aid air traffic controllers and managers in re-routing traffic: they generate a set of homotopy classes for a given pair of start and goal poses, by computing the K shortest paths via the Dijkstra algorithm on a Voronoi graph. The latter is generated from a map that encodes

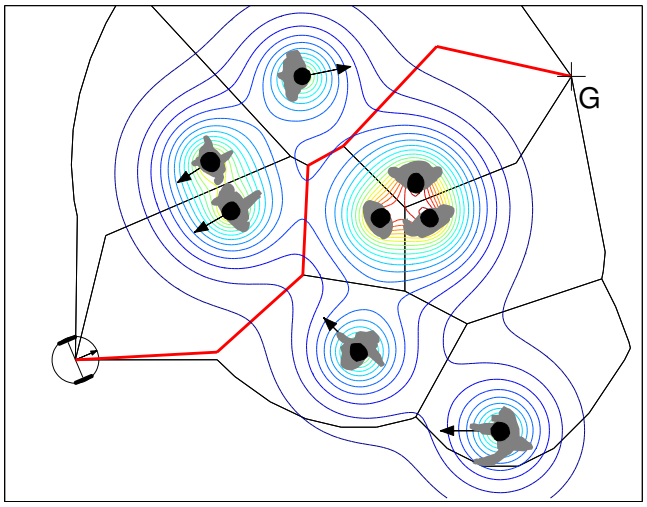


Fig. 1. An example path selected from the K best homotopy classes in the Voronoi diagram. The robot is enclosed in the **black circle**, in **red** the path selected to reach the goal position **G**. The **black** Voronoi diagram describes the possible ways to go through a crowd by implicitly encoding different homotopy classes.

L. Palmieri, A. Rudenko, K.O. Arras are with the Social Robotics Lab, Dept. of Computer Science, University of Freiburg, Germany. {palmieri,arras}@cs.uni-freiburg.de, andrey.rudenko@saturn.uni-freiburg.de

weather conditions. A final step optimize the set of K-shortest paths with respect to path length and weather avoidance. Voss *et al* [13] introduce an algorithm that seeks to find a set of diverse, short paths through a roadmap graph. The algorithm finds diverse homotopy classes by finding diverse shortest paths avoiding a collection of balls imposed on the graph as simulated obstacles. The authors compare their approach to the Eppstein algorithm [14] that finds the optimal set of K shortest paths with loops and show that, with tolerable loss in shortness, they produce equally diverse path sets orders of magnitude more quickly. Hernandez *et al* [15] propose and compare three different path planning algorithms that exploit the set of homotopy classes generated for any 2D workspace: Homotopic A*, Homotopic RRT and Homotopic Bug. Their method to generate homotopy classes modifies the one introduced by Jenkins [16]: it first builds a reference frame in the workspace which is used to identify the homotopy classes and afterwards it builds a topological graph which allows an easy and systematic computation of homotopy classes. The homotopy classes are sorted according to a lower bound heuristic estimator, then they are used to guide and to constrain topologically the path search.

Instead of solving the K shortest paths problem with deterministic graph search algorithms like in [9, 10, 12], we introduce and show preliminary results of a fast randomized method based on random walks that finds a set of K homotopically distinct paths, according to a social cost. As in [10] we build a navigation graph from the Voronoi diagram. Each path found in the Voronoi diagram represents a distinct homotopy class. Differently from [13, 10] we compare our method to Yen's algorithm, a fast algorithm that finds loopless paths. In [13] the authors compare their approach to Eppstein's algorithm which finds paths with loops therefore having a lower chance to find a more diverse set of paths. In [10] the authors use Katoh's algorithm, however, it was shown by Brander and Sinclair [17] that for small size graphs and paths of small number of vertices, like in the case we consider, Yen's is faster than Katoh's. Furthermore in our approach, instead of using a polygonal representation of the obstacles as in [6, 7], we use occupancy grids where obstacles are represented by blocked cells, therefore permitting an easier integration with existing mapping frameworks. We assume in our work that the pedestrians' poses are given by a people (or group) tracker [18, 19].

The contribution of our paper is as follows:

- we introduce a fast, and easy to implement, randomized approach to find a set of K paths belonging to K different homotopy classes.
- we perform an extensive evaluation and compare our method to Yen's algorithm. Our approach is faster than Yen's to find a subset of homotopy classes in a Voronoi diagram;
- moreover our approach generates more diverse paths than Yen's (the robot has a more diverse set from where to choose the path to follow), while obtaining a negligible loss in path quality.

The paper is structured as follows: we detail our new approach and its analysis in Section II. We present the experiments in Section III and discuss the results in Section IV. Section V concludes the paper.

II. Our Approach

We introduce a probabilistic approach to find paths belonging to different homotopy classes for socially-aware robot motion planning. Our approach is complete, it can find all the possible paths, namely all the homotopy classes implicitly encoded in the navigation graph built from a Voronoi diagram that describes the scenario. Having a set of possible paths, we choose the best one according to a cost that considers social interactions between humans.

A. Navigation Graph

Our method could be used with discrete information received from a people tracker, however, it is equally well suited for use on general occupancy grids. In our case the input scenario is given as a collection of discrete people poses, (x, y, θ), in the 2D workspace, see Fig.2 (left). To frame the motion planning task as a graph search problem, we build the navigation graph *G* of the scenario from the Voronoi diagram *VD*, generated considering as obstacles also the people poses, see Fig.2 (middle). We create two additional vertices for the initial robot position and goal position, and connect them to the closest point of the Voronoi diagram, see Fig.2 (right). In the navigation graph, built in this way, different paths from the initial robot position to the goal position belong to different homotopy classes.

The graph *G(V,E)* consists of a set of nodes (or vertices) *V* and a set of edges *E*. In this work, *N* is the number of nodes in the graph and *M* the number of edges. We associate to each edge e_{ij}, connecting the node v_j to its neighbor v_i, an attribute or cost c_{ij}. $E(v_j)$ denotes the set of incoming and outgoing edges of v_j.

We compute the set of homotopy classes by running our random walk based algorithm on *G*. A walk w of length $k-1$ in a graph is a sequence of nodes $v_1, v_2, \ldots, v_k$, where each pair of nodes is connected by an edge, $(v_{i-1}, v_i) \in E$ for $1 < i \leq k$. The adjacency matrix A of G expresses the topology of the graph and is defined as

$$[A]_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E, i \neq j \\ 0 & otherwise \end{cases}$$

Walks are usually referred to as *paths*.

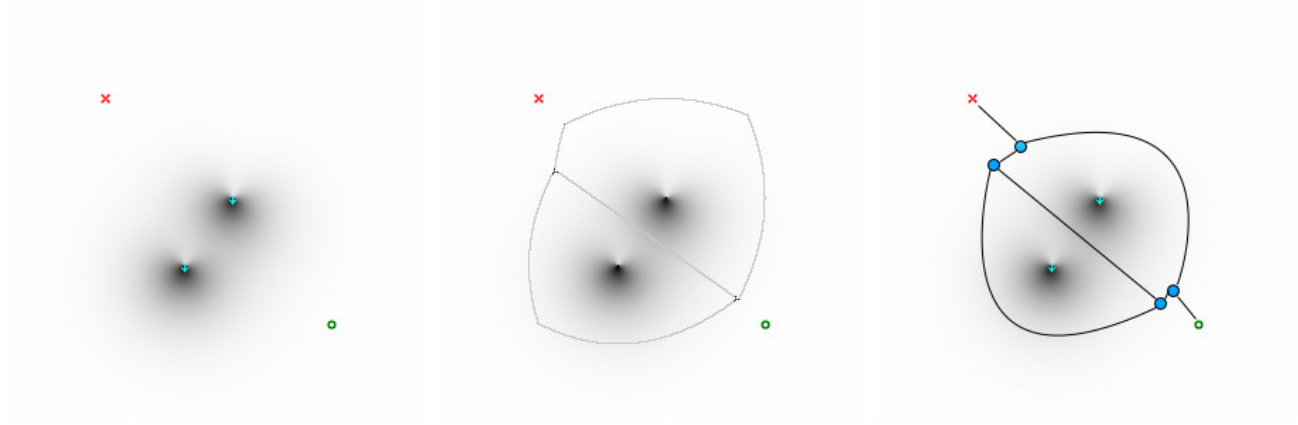


Fig. 2. In all the figures the **red** cross is the robot position, the **green** circle is the goal. **Left:** pedestrians' poses, marked with **blue** arrows, are provided by the people tracker, a social force field is generated over the two pedestrians. **Middle:** a Voronoi diagram is built considering the pedestrian positions as obstacles. The Voronoi diagram points are displayed in **darker grey**. **Right:** navigation graph built from the Voronoi diagram. The robot and goal nodes are mapped to their closest Voronoi points. The navigation graph **vertices** are depicted in **blue**, its edges in **black** .

B. Randomized Homotopy Classes Finder

To find different homotopy classes we introduce the Randomized Homotopy Classes Finder (**RHCF**), detailed in Alg.1. We iteratively run the random walk algorithm, see Alg.2, on the weighted undirected graph until K distinct paths are found.

Given the weighted graph G, we start at the initial node v_s, the one where the robot position is mapped to.

At each step of the random walk we choose a random neighbor of the current node v_j (see RandomNeighbor(v_j) in Alg.1) with probability p_{ij} inversely proportional to the cost c_{ij} associated to the edge e_{ij}

$$p_{ij} = \frac{\frac{1}{c_{ij}} A_{ij}}{\sum_k \frac{1}{c_{kj}} A_{kj}} \tag{1}$$

where A_{ij} is an element of the adjacency matrix A.

The transition matrix P of the graph G is defined as the $N \times N$ matrix where each of its elements is defined as in Eq.1.

Each time we leave a node, we mark it as visited (by removing it from the local copy of the graph G_p) and do not allow the algorithm to walk through it again in the current run of the random walk. The walk stops when the goal node is found or when we reach a node with all neighbors marked as visited. Each time a path **P** is generated, we compare it to the ones already found. If the same path was not generated before and it is a valid path, we save it in our homotopy classes set H. A path is valid if its last node is v_g. All the visited nodes are then marked unvisited.

After the K paths have been found, the robot chooses the best one to follow in terms of the social cost function. In the case that the followed path is invalidated by an unexpected obstacle, the planner selects the best path from the set of available paths, given the current status of the robot and of the environment.

Algorithm 1 Randomized Homotopy Classes Finder

function RHCF(v_s, v_g, G, K)
$k = 0$
while $k < K$ **do**
  $\mathbf{P} \leftarrow RandomWalk(v_s, v_g, G)$
  if $\mathbf{P} \notin H$ **and** $isvalid(\mathbf{P})$ **then**
    $H \leftarrow H \cup \mathbf{P}$
    $k = k + 1$
  end if
end while

Algorithm 2 Random Walk

function RandomWalk(v_s, v_g, G)
$v_j \leftarrow v_s$
$G_p \leftarrow G$
$\mathbf{P} \leftarrow v_j$
while $v_j \neq v_g$ **do**
  $v_i \leftarrow RandomNeighbor(v_j)$
  $\mathbf{P} \leftarrow \mathbf{P} \cup v_i$
  $G_p \leftarrow G_p \setminus E(v_j)$
  $v_j \leftarrow v_i$
end while
return **P**

C. Cost definition

Several state of the art approaches focus on finding the K-shortest paths [9, 10, 12, 13]. Here we are interested in finding the K best homotopy classes for socially-aware motion planning, therefore we compute the edges weights c_{ij} by using a cost function derived from the social force model introduced by Helbing [20].

The cost c_{ij} is defined by the line integral of the pedestrians' interaction forces on the planar curve s_{ij} described by the edge e_{ij} and the edge length l_s.

$$c_{ij} = \int_{s_{ij}} F_s \; ds + l_s \tag{2}$$

The force F_s represents the force generated from the interactions of all the pedestrians p_i with the robot, defined as pedestrian p_j,

$$F_s = \sum_{i \in \mathcal{P}} \mathbf{f}_{i,j} \tag{3}$$

with $\mathcal{P} = \{p_i\}_{i=1}^{N_p}$ being the set of N_p pedestrians. The forces $\mathbf{f}_{i,j}$ decrease proportional to the distance of their sources to the robot, and are modelled as

$$\mathbf{f}_{i,j} = a_j e^{\left(\frac{r_{i,j} - d_{i,j}}{b_j}\right)} \boldsymbol{n}_{i,j} \tag{4}$$

where a_j specifies the magnitude and b_j the range of the force. The distance $d_{i,j}$ is given by the Euclidean distance between the pedestrian p_i and robot, $r_{i,j}$ is the sum of their radii (we approximate each pedestrian with a circle). The term $\boldsymbol{n}_{i,j}$ is the normalized vector pointing from p_i to the robot which describes the direction of the force.

To better describe the limited field of view of the pedestrian, the forces are scaled with an anisotropic factor (see Fig.3)

$$\mathbf{f}_{i,j} = a_j e^{\left(\frac{r_{i,j} - d_{i,j}}{b_j}\right)} \boldsymbol{n}_{i,j} \left(\lambda + (1-\lambda)\frac{1 + \cos\left(\varphi_{i,j}\right)}{2}\right) \tag{5}$$

where λ defines the strength of the anisotropic factor and

$$\cos\left(\varphi_{i,j}\right) = -\boldsymbol{n}_{i,j} \cdot \widehat{\boldsymbol{e}}_i \,. \tag{6}$$

with $\widehat{\boldsymbol{e}}_i$ representing the direction of the pedestrian p_i.

Notice that our approach is not limited to a single definition of cost, different definitions from the social navigation literature could be used [10, 21, 22].

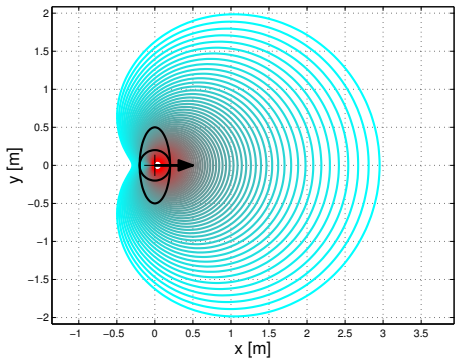


Fig. 3. Anisotropic influence model for $\lambda = 0.1$, $a_j = 2$, $b_j = 1$, $r_{ij} = 0.4$ generated by the social force F_s for a single pedestrian. **Red** regions denote higher cost regions.

III. Experiments

To evaluate our approach in terms of planning performance and quality of the solutions, we design a set of experiments by choosing proper environments and metrics. We compare our approach to Yen's algorithm. In both algorithms we adopt the cost defined in Sec.II-C. The algorithms are implemented in C++. All experiments were running on an ordinary PC with 2.3 GHz Intel Core i5 and 8 GB of RAM.

A. Yen's Algorithm

In [8] Yen introduces an algorithm to find K-shortest loopless paths for a given pair of start and goal poses. The algorithm computational upper bound increases linearly with the value of K: with modern data structure it can be implemented in $O(KN(M + Nlog(N)))$ worst-case time. We use the C++ implementation introduced by Martins and Pascoal in [23], which is reported to have better performance than the straightforward implementation. We compare our approach to the Yen's algorithm, because it finds a set of k best paths with an higher diversity than the ones found by Eppstein's and it was shown by Brander and Sinclair [17] that for small size graphs and paths of small length (like the graphs generated from a Voronoi diagram), it is faster than Katoh's.

B. Environments

We design four different environments (shown in Fig 4), to stress different properties of the planner and to study how the algorithm behaves in environments of varying complexity. We choose scenarios resembling potential situations that could occur while a robot is navigating into an airport. In the *wall of people* scenario, the robot needs to find different ways to the goal through a queue of standing people, this scenario has 33 possible homotopy classes. In the *crowd A* (670 homotopy classes) and in the *crowd B* (576 homotopy classes) scenarios, the people are placed in a sparser way forming different groups. In the scenario *surrounded*, that has 1826 possible homotopy classes, the robot is placed in the crowd, surrounded by several people.

C. Metrics

To quantify planning performance and difference in quality with respect to Yen's search method we compute the averages of the following metrics: T_k time to get K different homotopy classes, nCG_k normalized cumulative gain, RD_k robust diversity of a set $\mathbf{P}_K$ of K paths returned by the algorithms. The *robust diversity* measures how large are the intra-set distances between pairs of paths in the set $\mathbf{P}_K$. Let us consider $d_f(p_a, p_b)$ to be the Fréchet distance between two paths p_a and p_b evaluated at the vertices, as in [13]. We define RD_k as

$$RD_k = \frac{1}{|\mathbf{P}_K|} \sum_{p_a \in P} \min_{p_b \in P, p_a \neq p_b} d_f(p_a, p_b).$$

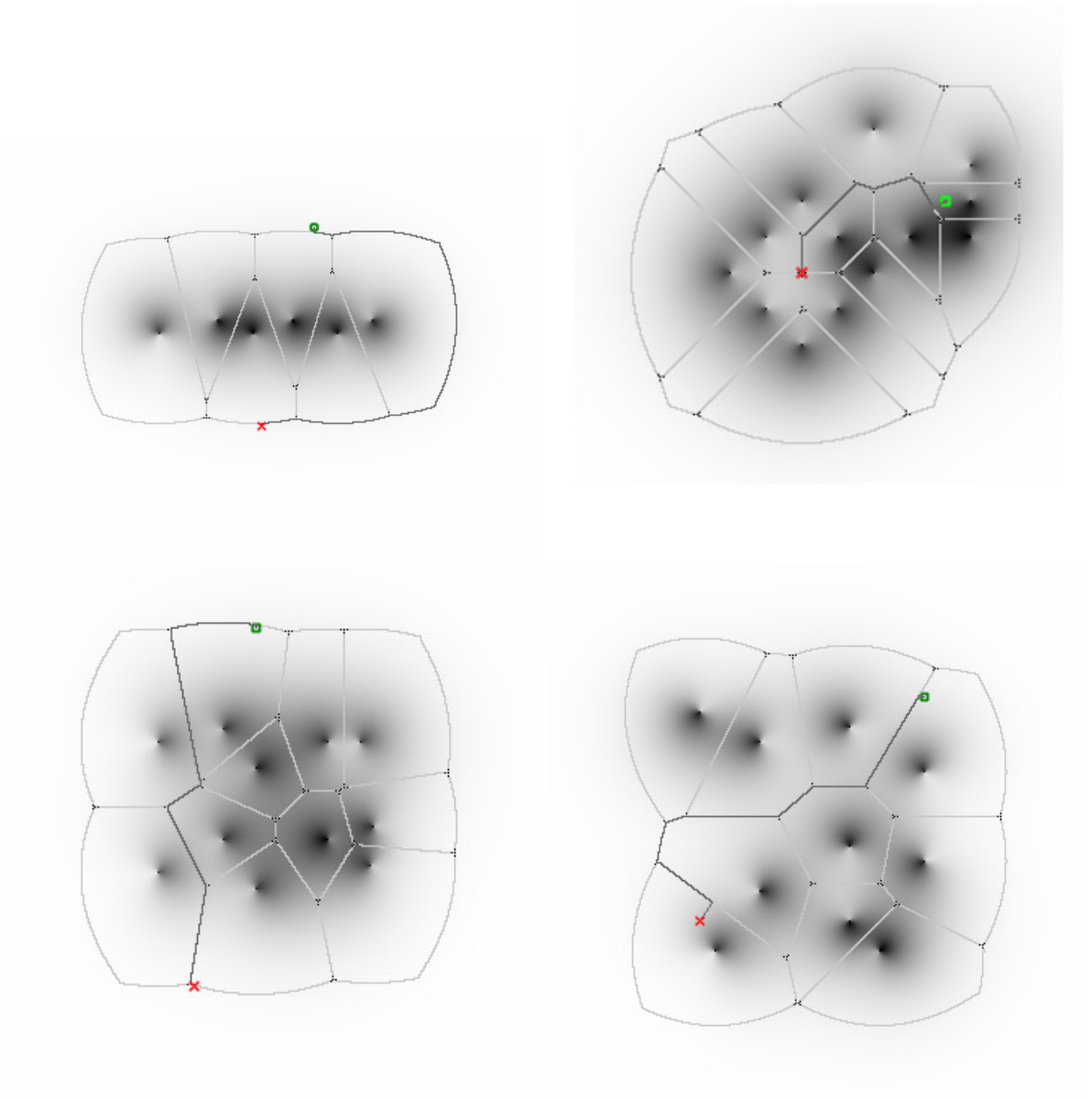


Fig. 4. In all the environments, the social force is displayed in grey scale: darker regions have a higher social cost. The peaks of the social force field represent the agent positions. The scenario at **top left** is the *wall of people* environment. The one at **top right** part of the figure is the *surrounded* environment. At the **bottom left** and at the **bottom right** are respectively the *crowd A* and the *crowd B* scenarios. **Red** crosses represent the robot position, the goal is displayed by **green** circles. In all the figures, the edges of the Voronoi diagram are in **dark grey** and example paths generated by our approach are displayed with **black** edges.

The normalized cumulative gain nCG_k is often used to measure the goodness of the ranking results returned by a web search engine algorithm. It computes how far is the candidate ranking set from the ideal ranking set. It is based on the definition of relevance (rel) of a single path. In our case the relevance is defined as the inverse of the path cost.

$$CG_k = \sum_{k}^{K} rel_k$$

$$nCG_k = \frac{CG_k}{\max(CG_k)}$$

To paths with smaller costs correspond higher values of cumulative gain. nCG_k is normalized by the maximum cumulative gain of k best paths generated by Yen's.

D. Parameters

In the algorithm only one parameter need to be set: the number of homotopy classes to find. For Yen and RHCF algorithms we find the first 5 paths or homotopy classes (K=5). The parameters of the cost function after several informal validations have been set to (see Fig.3):

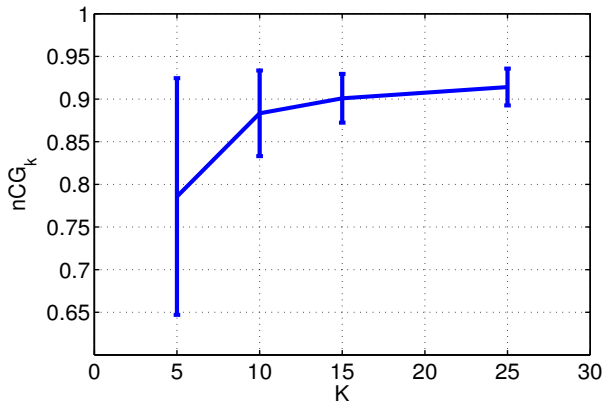


Fig. 5. Normalized Cumulative Gain $\mathbf{nCG}_k$ obtained by varying K in the *crowd A* scenario. As K increases, our approach converges to the optimal value of the normalized cumulative gain $\mathbf{nCG}_k$.

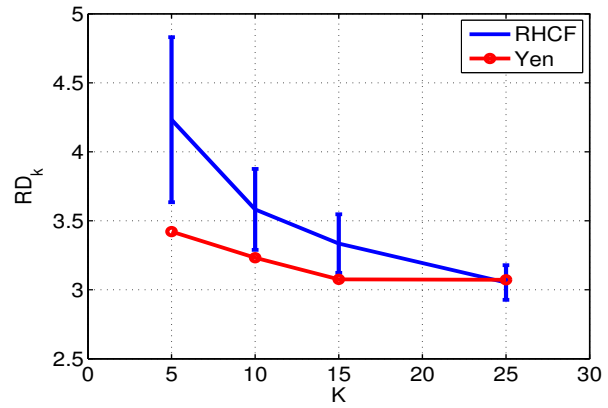


Fig. 6. Robust diversity $\mathbf{RD}_k$ obtained by varying K in the *crowd A* scenario. The paths produced by our approach are more diverse than the one generated by Yen's for small values of K.

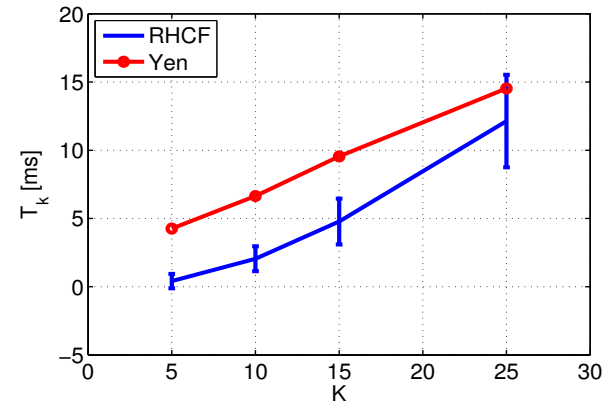


Fig. 7. Planning time $\mathbf{T}_k$ obtained with different values for K in the *crowd A* scenario. For small values of K, our approach is faster than Yen's algorithm in very complex scenarios (with hundreds of different homotopy classes).

$\mathbf{RD}_k$		
Scenarios	RHCF	Yen
Crowd A	**3.76813**	2.79823
Crowd B	**4.23284**	3.4217
Wall of People	3.7924	**5.19881**
Surrounded	**3.76813**	2.79823

TABLE I
ROBUST DIVERSITY RESULTS

$\mathbf{nCG_k}$	
Scenarios	RHCF
Crowd A	0.7857
Crowd B	0.7142
Wall of People	0.771
Surrounded	0.7461

TABLE II
CUMULATIVE GAIN RESULTS

magnitude of social force to pedestrians a_j to 2, range of social force to pedestrians b_j to 1, strength of the anisotropic factor λ to 0.1, r_{ij} equal to 0.4 m.

IV. RESULTS AND DISCUSSION

Table I, Table II and Table III collect the results generated for all the described scenarios, considering the number of homotopy classes to find equal to five ($K = 5$). Table III shows the planning time results: our approach is at least five times faster than Yen's approach. Table II details the results related to the cumulative gain, *RHCF* while being faster than Yen's it also finds homotopy classes with a gain close the optimal one (i.e 1): the costs of the solutions generated by the two approaches are only slightly different, the solutions' quality of the two approaches are very similar. Table I details the diversity of the paths generated by both approaches: our method outperforms Yen's in three environments of the four. Only in one scenario, *wall of people*, Yen's finds more diverse paths.

Fig.5-7 show the metrics trends for different values of K in the *crowd A* scenario (the same trends are visible in the other scenarios too). For small values of K, our approach is faster than Yen's algorithm in very complex scenarios (with hundreds of different homotopy classes), as it is showed in Fig.7. In average, when K is greater than one fourth of the total possible homotopy classes in the scenario, the algorithm is slower than Yen's. Our approach has a better robust diversity RD_k, considering K up to 25 see Fig.6: the paths produced are more diverse than the one generated by Yen's for small values of K. With a higher value for K, see Fig.5, our approach converges to the optimal value of the normalized cumulative gain nCG_k values, the one associated to Yen's rankings.

V. CONCLUSIONS

In this paper, we introduce the Randomized Homotopy Classes Finder, that finds homotopy classes in

$\mathbf{T_K}$ $[ms]$		
Scenarios	RHCF	Yen
Crowd A	**0.41**	4.26
Crowd B	**1.53**	4.42
Wall of People	**0.035**	2.07
Surrounded	**1.05**	5.61

TABLE III
PLANNING TIME RESULTS

an undirected weighted graph built from a Voronoi diagram. We use the algorithm to find a set of k distinct socially-aware paths from which the robot chooses the best one to follow in terms of a social cost function. Our experimental evaluation shows that our approach is faster than Yen's approach. Moreover, as the cumulative gains results show, the paths produced by our approach are not far from the ones generated by Yen's that finds the true k best paths. A key property is that our approach computes a set of more diverse paths respect to the baseline: usually different paths share few edges which make them robust to invalidation due to unexpected obstacles.

In future work, we intend to further improve the time performance of the *RHCF* by introducing a discounting factor that biases the search towards a not frequently visited subset of the state space, therefore increasing the probability to generate paths not yet found. Moreover we are interested to couple our approach with an informed (weighted) Voronoi diagram that implicitly encodes the social context of the scene. Finally, we plan to integrate our algorithm in a hierarchical framework where an optimal sampling-based motion planner generates (locally) optimal kinodynamic trajectories in the best homotopy class found by *RHCF*.

ACKNOWLEDGEMENTS

The authors thank Markus Kuderer and Christoph Sprunk for valuable discussions and feedback. This work has partly been supported by the European Commission under contract number FP7-ICT-600877 (SPENCER)

REFERENCES

[1] R. Triebel, K. Arras, R. Alami, L. Beyer, S. Breuers, R. Chatila, M. Chetouani, D. Cremers, V. Evers, M. Fiore, H. Hung, O. A. I. Ramírez, M. Joosse, H. Khambhaita, T. Kucner, B. Leibe, A. J. Lilienthal, T. Linder, M. Lohse, M. Magnusson, B. Okal, L. Palmieri, U. Rafi, M. van Rooij, and L. Zhang, "Spencer: A socially aware service robot for passenger guidance and help in busy airports," in *Proc. Field and Service Robotics (FSR)*, 2015.

[2] N. J. Nilsson, "Problem-solving methods in artificial intelligence," 1971.

[3] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," in *Int. Conf. on Robotics and Automation (ICRA)*, Detroit, USA, 1999.

[4] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.

[5] S. Koenig and M. Likhachev, "D* lite." in *AAAI/IAAI*, 2002, pp. 476–483.

[6] D. Demyen and M. Buro, "Efficient triangulation-based pathfinding," in *Proc. of the AAAI Conf. on Artificial Intelligence (AAAI)*, 2006.

[7] S. Eriksson-Bique, J. Hershberger, V. Polishchuk, B. Speckmann, S. Suri, T. Talvitie, K. Verbeek, and H. Yıldız, "Geometric k shortest paths," in *26th Symposium on Discrete Algorithms (SODA)*. SIAM, 2015.

[8] J. Y. Yen, "Finding the k shortest loopless paths in a network," *management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[9] S. Bhattacharya, V. Kumar, and M. Likhachev, "Search-based path planning with homotopy class constraints," in *Third Annual Symposium on Combinatorial Search*, 2010.

[10] M. Kuderer, C. Sprunk, H. Kretzschmar, and W. Burgard, "Online generation of homotopically distinct navigation paths," in *Int. Conf. on Robotics and Automation (ICRA)*, 2014.

[11] N. Katoh, T. Ibaraki, and H. Mine, "An efficient algorithm for k shortest simple paths," *Networks*, vol. 12, no. 4, pp. 411–427, 1982.

[12] P. Vela, A. Vela, and G. Ogunmakin, "Topologically based decision support tools for aircraft routing," in *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, 2010.

[13] C. Voss, M. Moll, and L. E. Kavraki, "A heuristic approach to finding diverse short paths," in *Int. Conf. on Robotics and Automation (ICRA)*, 2015.

[14] D. Eppstein, "Finding the k shortest paths," *SIAM J. Computing*, vol. 28, no. 2, 1998.

[15] E. Hernandez, M. Carreras, and P. Ridao, "A comparison of homotopic path planning algorithms for robotic applications," *Robotics and Autonomous Systems*, vol. 64, no. 0, 2015.

[16] K. D. Jenkins, "The shortest path problem in the plane with obstacles: A graph modeling approach to producing finite search lists of homotopy classes, master's thesis," in *Monterey, California, Naval Postgraduate School*, 1991.

[17] A. W. Brander and M. C. Sinclair, "A comparative study of k-shortest path algorithms," *Proc. of 11th UK Performance Engineering Workshop*, 1996.

[18] T. Linder and K. O. Arras, "Multi-model hypothesis tracking of groups of people in RGB-D data," in *IEEE Int. Conf. on Information Fusion (FUSION'14)*, Salamanca, Spain, 2014.

[19] M. Luber, G. D. Tipaldi, and K. O. Arras, "Place-dependent people tracking," *International Journal of Robotics Research*, vol. 30, no. 3, March 2011.

[20] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical review E*, vol. 51, no. 5, p. 4282, 1995.

[21] D. Vasquez, B. Okal, and K. O. Arras, "Inverse reinforcement learning algorithms and features for robot navigation in crowds: an experimental comparison," 2014.

[22] T. Kruse, A. Kirsch, H. Khambhaita, and R. Alami, "Evaluating directional cost models in navigation," in *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction*. ACM, 2014, pp. 350–357.

[23] E. Q. Martins and M. M. Pascoal, "A new implementation of Yen's ranking loopless paths algorithm," *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, vol. 1, no. 2, pp. 121–133, 2003.