The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

Motion Planning for an Omnidirectional Robot with Steering Constraints

Simon Chamberland*, Éric Beaudry*, Lionel Clavien*, Froduald Kabanza*, François Michaud*, Michel Lauria†

*Université de Sherbrooke, Sherbrooke, Québec – Canada
Email: {firstname.lastname}@USherbrooke.ca
†University of Applied Sciences Western Switzerland (HES-SO), Geneva – Switzerland
Email: michel.lauria@hesge.ch

***Abstract*—Omnidirectional mobile robots, i.e., robots that can move in any direction without changing their orientation, offer better manoeuvrability in natural environments. Modeling the kinematics of such robots is a challenging problem and different approaches have been investigated. One of the best approaches for a nonholonomic robot is to model the robot's velocity state as the motion around its instantaneous center of rotation (ICR). In this paper, we present a motion planner designed to compute efficient trajectories for such a robot in an environment with obstacles. The action space is modeled in terms of changes of the ICR and the motion around it. Our motion planner is based on a Rapidly-Exploring Random Trees (RRT) algorithm to sample the action space and find a feasible trajectory from an initial configuration to a goal configuration. To generate fluid paths, we introduce an adaptive sampling technique taking into account constraints related to the ICR-based action space.**

I. INTRODUCTION

Many real and potential applications of robots include exploration and operations in narrow environments. For such applications, omnidirectional mobile platforms provide easier manoeuvrability when compared to differential-drive or skid-steering platforms. Omnidirectional robots can move sideways or drive on a straight path without changing their orientation. Translational movement along any desired path can be combined with a rotation, so that the robot arrives to its destination with the desired heading.

Our interest lies in nonholonomic omnidirectional wheeled platforms, which are more complex to control than holonomic ones. This complexity stems from the fact that they cannot instantaneously modify their velocity state. Nevertheless, nonholonomic robots offer several advantages motivating their existence. For instance, the use of conventional steering wheels reduces their cost and results in a more reliable odometry, which is important for many applications. Our work is centered around AZIMUT-3, the third prototype of AZIMUT [1], [2], a multi-modal nonholonomic omnidirectional platform. The wheeled configuration of AZIMUT-3, depicted on Fig. 1, is equipped with four wheels constrained to steer over a 180° range, and a passive suspension mechanism.

There are different approaches to control the kinematics of a wheeled omnidirectional robot. For AZIMUT, we chose to model the velocity state by the motion around the robot's instantaneous center of rotation (ICR) [3]. The ICR is defined as the unique point in the robot's frame which is instantaneously not moving with respect to the robot. For a robot using conventional steering wheels, this corresponds to the point where the propulsion axis of each wheel intersect.

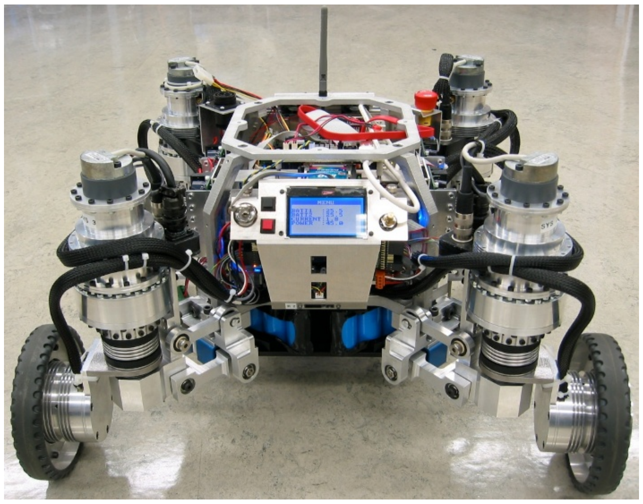


Fig. 1. The AZIMUT-3 platform in its wheeled configuration

The robot's chassis represents a physical constraint on the rotation of the wheels around their steering axis. These constraints introduce discontinuities on the steering angle of some wheels when the ICR moves continuously around the robot. In fact, a small change of the ICR position may require reorienting the wheels, such that at least one wheel has to make a full 180° rotation. This rotation takes some time, depending on the steering axis' maximum rotational speed. During such wheel reorientation, the ICR is undefined, and because the robot is controlled through its ICR, it must be stopped until the wheel reorientation is completed and the new ICR is reset.

As a solution, a motion planner for an ICR-based motion controller could return a trajectory avoiding wheel reorientations as much as possible, in order to optimize travel time and keep fluidity in the robot's movements. One way to achieve this is to use a traditional obstacle avoidance path planner ignoring the ICR-related kinematic constraints, and then heuristically smoothing the generated path to take into account the constraints that were abstracted away. This is in fact one of the approaches used to reduce intrinsically nonholonomic motion planning problems to holonomic ones [4].

However, we believe this approach is not well suited for the AZIMUT robot, as we would prefer to perform a global optimization of the trajectories, instead of optimizing a potentially ill-formed path. To this end, we chose to investigate another approach which takes directly into account the kinematic constraints related to the ICR and to the robot's velocity. The action space of the robot is modeled as the space of possible ICR changes. We adopt a Rapidly-Exploring Random Trees (RRT) planning approach to sample the action space and find a feasible trajectory from an initial configuration to a goal configuration. To generate fluid paths, we introduce an adaptive sampling technique taking into account the constraints related to the ICR-based action space.

The rest of the paper is organized as follows. Sect. II describes the velocity state of the AZIMUT-3 robot and Sect. III characterizes its state space. Sect. IV presents a RRT-based motion planner which explicitly consider the steering limitations of the robot, and Sect. V concludes the paper with simulation results.

II. Velocity state of AZIMUT

Two different approaches are often used to describe the velocity state of a robot chassis [3]. The first one is to use its twist (linear and angular velocities) and is well adapted to holonomic robots, because their velocity state can change instantly (ignoring the maximum acceleration constraint). However, this representation is not ideal when dealing with nonholonomic robots, because their instantaneously accessible velocities from a given state are limited (due to the non-negligible reorientation time of the steering wheels). In these cases, modeling the velocity state using the rotation around the current ICR is preferred.

As a 2D point in the robot frame, the ICR position can be represented using two independent parameters. One can use either Cartesian or polar coordinates to do so, but singularities arise when the robot moves in a straight line manner (the ICR thus lies at infinity). An alternative is to represent the ICR by its projection on a unit sphere tangent to the robot frame at the center of the chassis. This can be visualized by tracing a line between the ICR in the robot frame and the sphere center. Doing so produces a pair of antipodal points on the sphere's surface, as shown on Fig. 4. Using this representation, an ICR at infinity is projected onto the sphere's equator. Therefore, going from one near-infinite position to another (e.g., when going from a slight left turn to a slight right turn) simply corresponds to an ICR moving near the equator of the sphere.

In the following, we define $\mathbf{\Lambda} = \{(u;v;w) : u^2+v^2+w^2=1\}$ as the set of all possible ICR on the unit sphere (in Cartesian coordinates), and $\mu \in [-\mu_{max}(\boldsymbol{\lambda});\mu_{max}(\boldsymbol{\lambda})]$ as the motion around a particular $\boldsymbol{\lambda} \in \mathbf{\Lambda}$, with $\mu_{max}(\boldsymbol{\lambda})$ the fastest allowable motion. We also define $\mu_{max} = \max\limits_{\boldsymbol{\lambda}\in\mathbf{\Lambda}} \mu_{max}(\boldsymbol{\lambda})$ as

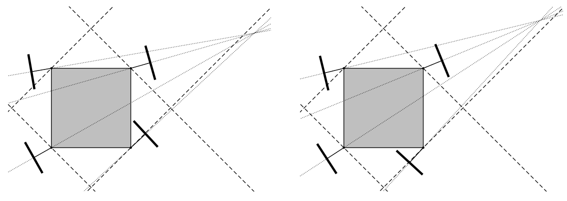

(a) ICR at $(0.8; 0.43)$ (b) ICR at $(0.8; 0.51)$

Fig. 2. ICR transition through a steering limitation. Observe the 180° rotation of the lower right wheel.

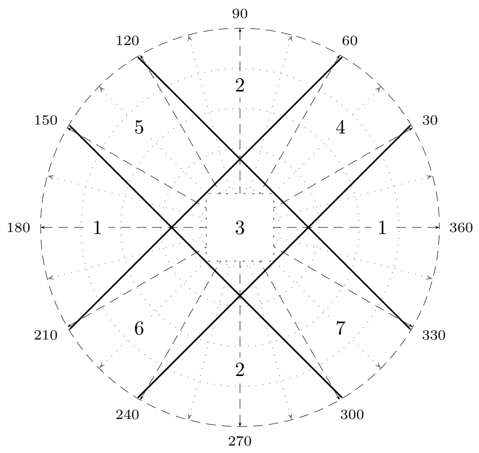


Fig. 3. Different control "zones" induced by the steering constraints. The dashed square represents the robot without its wheels.

an upper bound over all possible ICR. The whole velocity state is then specified as $\boldsymbol{\eta} = (\boldsymbol{\lambda}; \mu)$.

AZIMUT has steering limitations for its wheels. These limitations define the injectivity of the relation between ICR ($\boldsymbol{\lambda}$) and wheel configurations ($\boldsymbol{\beta}$, the set of all N steering angles). If there is no limitation, one ICR corresponds to 2^N wheel configurations, where $N \geq 3$ is the number of steering wheels. If the limitation is more than 180°, some ICR are defined by more than one wheel configuration. If the limitation is less than 180°, some ICR cannot be defined by a wheel configuration. It is only when the limitation is of 180°, as is the case with AZIMUT, that the relation is injective: for each given ICR, there exists a unique wheel configuration.

Those limitations restrict the trajectories along which the ICR can move. Indeed, each limitation creates a frontier in the ICR space. When a moving ICR needs to cross such a frontier, one of the wheel needs to be "instantly" rotated by 180°. One example of such a situation for AZIMUT-3 is shown on Fig. 2. To facilitate understanding, the ICR coordinates are given in polar form. As the ICR needs to be defined to enable motion, the robot has to be stopped for this rotation to occur. As shown on Fig. 3, the set of all limitations splits up the ICR space into several zones, and transitions between any of these zones are very inefficient. In the following, we refer to these ICR control zones as "modes".

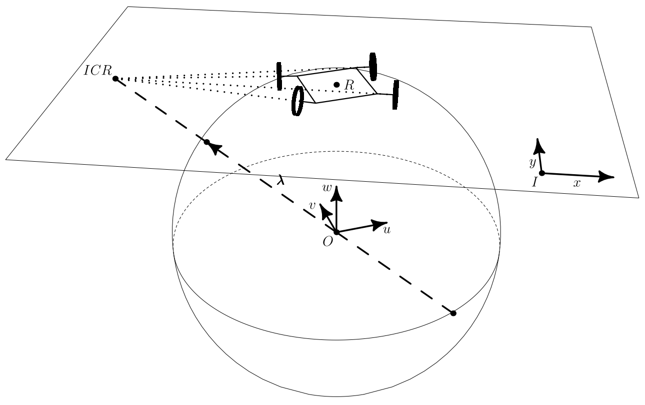


Fig. 4. State parameters. $R(0;0;1)$ is the center of the chassis.

III. Planning State Space

The representation of states highly depends on the robot to control. A state $\boldsymbol{s} \in S$ of AZIMUT-3 is expressed as $\boldsymbol{s} = (\boldsymbol{\xi};\boldsymbol{\lambda})$, where $\boldsymbol{\xi} = (x;y;\theta) \in \mathbb{SE}(2)$ represents the posture of the robot and $\boldsymbol{\lambda}$ is its current ICR, as shown on Fig. 4.

As mentioned in Sect. II, the ICR must be defined at all times during motion for the robot to move safely. Keeping track of the ICR instead of the steering axis angles therefore prevents expressing invalid wheel configurations. Since the relation $\boldsymbol{\lambda} \mapsto \boldsymbol{\beta}$ is injective, any given ICR corresponds to a unique wheel configuration, which the motion planner can disregard by controlling the ICR instead. Because AZIMUT can accelerate from zero to its maximal speed in just a few tenths of a second, the acceleration is not considered by the planner. Thus, instantaneous changes in velocity (with no drift) are assumed, which is why the current velocity of the robot is not included in the state variables.

A trajectory σ is represented as a sequence of n pairs $((\boldsymbol{u}_1,\Delta t_1),\ldots,(\boldsymbol{u}_n,\Delta t_n))$ where $\boldsymbol{u_i} \in U$ is an action vector applied for a duration Δt_i. In our case, the action vector corresponds exactly to the velocity state to transition to: $\boldsymbol{u} = \boldsymbol{\eta}_u = (\boldsymbol{\lambda}_u;\mu_u)$, respectively the new ICR to reach and the desired motion around that ICR.

It is necessary to have a method for computing the new state $\boldsymbol{s'}$ arising from the application of an action vector $\boldsymbol{u}$ for a certain duration Δt to a current state $\boldsymbol{s}$. Since the state transition equation expressing the derivatives of the state variables is non-linear and complex, we use AZIMUT's kinematical simulator as a black box to compute new states. The simulator implements the function $K : S \times U \to S = \boldsymbol{s'} \mapsto K(\boldsymbol{s},\boldsymbol{u})$ via numerical integration.

IV. Motion Planning Algorithm

Following the RRT approach, our motion planning algorithm (Alg. 1) expands a search tree of feasible trajectories until reaching the goal. The initial state of the robot $\boldsymbol{s}_{init}$ is set as the root of the search tree. At each iteration, a random state $\boldsymbol{s}_{rand}$ is generated (Line 4). Then its nearest neighboring node $\boldsymbol{s}_{near}$ is computed (Line 5), and an action is selected (Line 6) which, once applied from $\boldsymbol{s}_{near}$, produces an edge extending toward the new sample $\boldsymbol{s}_{rand}$. A local planner (Line 7) then finds the farthest collision-free state $\boldsymbol{s}_{new}$ along the trajectory generated by the application of the selected action. The problem is solved whenever a trajectory enters the goal region, C_{goal}. As the tree keeps growing, so does the probability of finding a solution. This guarantees the probabilistic completeness of the approach.

Algorithm 1 RRT-Based Motion Planning Algorithm

```
1.  RRT-PLANNER($\boldsymbol{s}_{init}, \boldsymbol{s}_{goal}$)
2.    $T.init(\boldsymbol{s_{init}})$
3.    repeat until time runs out
4.      $\boldsymbol{s}_{rand} \leftarrow GenerateRandomSample()$
5.      $\boldsymbol{s}_{near} \leftarrow SelectNodeToExpand(\boldsymbol{s}_{rand}, T)$
6.      $(\boldsymbol{u}, \Delta t) \leftarrow SelectAction(\boldsymbol{s}_{rand}, \boldsymbol{s}_{near})$
7.      $\boldsymbol{s}_{new} \leftarrow LocalPlanner(\boldsymbol{s}_{near}, \boldsymbol{u}, \Delta t)$
8.      add $\boldsymbol{s_{new}}$ to $T.Nodes$
9.      add $(\boldsymbol{s}_{near}, \boldsymbol{s}_{new}, \boldsymbol{u})$ to $T.Edges$
10.     if $C_{goal}$ is reached
11.       return EXTRACT-TRAJECTORY($\boldsymbol{s}_{new}$)
12.   return failure
```

The fundamental principle behind RRT approaches is the same as probabilistic roadmaps [5]. A naive state sampling function (e.g., uniform sampling of the state space) loses efficiency when the free space C_{free} contains narrow passages – a narrow passage is a small region in C_{free} in which the sampling density becomes very low. Some approaches exploit the geometry of obstacles in the workspace to adapt the sampling function accordingly [6], [7]. Other approaches use machine learning techniques to adapt the sampling strategy dynamically during the construction of the probabilistic roadmap [8], [9].

For the ICR-based control of AZIMUT, we are not just interested in a sampling function guaranteeing probabilistic completeness. We want a sampling function that additionally improves the travel time and motion fluidity. The fluidity of the trajectories is improved by minimizing:

- the number of mode switches;
- the number of reverse motions, i.e., when the robot goes forward, stops, then backs off. Although these reverse motions do not incur mode switches, they are obviously not desirable.

A. Goal and Metric

The objective of the motion planner is to generate fluid and time-efficient trajectories allowing the robot to reach a goal location in the environment. Given a trajectory $\sigma = ((\boldsymbol{u}_1,\Delta t_1),\ldots,(\boldsymbol{u}_n,\Delta t_n))$, we define reverse motions as consecutive action pairs $(\boldsymbol{u}_i,\boldsymbol{u}_{i+1})$ where $|\tau_i - \tau_{i+1}| \geq \frac{3\pi}{4}$, in which $\tau_i = \arctan(\boldsymbol{\lambda}_{v,i},\boldsymbol{\lambda}_{u,i}) - \mathrm{sign}(\mu_i)\frac{\pi}{2}$ represents the approximate heading of the robot. Let $(\boldsymbol{s}_0,\ldots,\boldsymbol{s}_n)$ denote the sequence of states produced by applying the actions in σ from an initial state $\boldsymbol{s}_0$. Similarly, we define mode switches as consecutive state pairs $(\boldsymbol{s}_i,\boldsymbol{s}_{i+1})$ where $\mathrm{mode}(\lambda_i) \neq \mathrm{mode}(\lambda_{i+1})$. To evaluate the quality of trajectories, we specify a metric f to be minimized as

$$q(\sigma) = t + c_1 m + c_2 r \qquad (1)$$

where $c_1, c_2 \in \mathbb{R}^+$ are weighting factors; t, m and r are, respectively, the duration, the number of mode switches and the number of reverse motions within the trajectory σ.

B. Selecting a Node to Expand

Line 4 of Alg. 1 generates a sample $\boldsymbol{s}_{rand}$ at random from a uniform distribution. As it usually allows the algorithm to find solutions faster [4], there is a small probability P_g of choosing the goal state instead.

Line 5 selects an existing node $\boldsymbol{s}_{near}$ in the tree to be extended toward the new sample $\boldsymbol{s}_{rand}$. Following a technique introduced in [10], we alternate between two different heuristics to choose this node.

Before a feasible solution is found, the tree is expanded primarily using an *exploration* heuristic. This heuristic selects for expansion the nearest neighbor of the sample $\boldsymbol{s}_{rand}$, as the trajectory between them will likely be short and therefore require few collision checks. Hence $\boldsymbol{s}_{near} = \underset{\boldsymbol{s}_i \in T.Nodes}{\arg\min}\ D_{exp}(\boldsymbol{s}_i, \boldsymbol{s}_{rand})$.

The distance metric used to compute the distance between two states $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ is specified as

$$\begin{aligned} D_{exp}(\boldsymbol{s}_1, \boldsymbol{s}_2) = & \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \\ & + \frac{1}{\pi} \left| \theta_2 - \theta_1 \right| \\ & + \frac{1}{2\pi} \sum_{j \in [1;4]} \left| \beta_{2,j} - \beta_{1,j} \right| \end{aligned} \quad (2)$$

which is the standard 2D Euclidean distance extended by the weighted sum of the orientation difference and the steering angles difference.

This heuristic is often used in the RRT approach as it allows the tree to rapidly grow toward the unexplored portions of the state space. In fact, the probability that a given node be expanded (chosen as the nearest neighbor) is proportional to the volume of its Voronoi region [11]. Nodes with few distant neighbors are therefore more likely to be expanded.

Once a solution has been found, more emphasis is given to an *optimization* heuristic which attempts to smooth out the generated trajectories. We no longer select the sample's nearest neighbor according to the distance metric D_{exp} (2). Instead, nodes are sorted by the weighted sum of their cumulative cost and their estimated cost to $\boldsymbol{s}_{rand}$. Given two samples $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, we define this new distance as:

$$D_{opt}(\boldsymbol{s}_1, \boldsymbol{s}_2) = q(\sigma_{s_1}) + c_3\, h^2(\boldsymbol{s}_1, \boldsymbol{s}_2) \quad (3)$$

where $q(\sigma_{s_1})$ is the cumulative cost of the trajectory from the root configuration to $\boldsymbol{s}_1$ (see (1)), $h(\boldsymbol{s}_1, \boldsymbol{s}_2)$ is the estimated cost-to-go from $\boldsymbol{s}_1$ to $\boldsymbol{s}_2$, and $c_3 \in \mathbb{R}^+$ is a weighting factor. We select $\boldsymbol{s}_{near}$ as the node with the lowest distance D_{opt} to $\boldsymbol{s}_{rand}$, or more formally $\boldsymbol{s}_{near} = \underset{\boldsymbol{s}_i \in T.Nodes}{\arg\min}\ D_{opt}(\boldsymbol{s}_i, \boldsymbol{s}_{rand})$.

We set $h(\boldsymbol{s}_1, \boldsymbol{s}_2)$ as a lower bound on the travel duration $\boldsymbol{s}_1$ to $\boldsymbol{s}_2$, which is found by computing the time needed to reach $\boldsymbol{s}_2$ via a straight line at maximum speed, i.e.

$$h(\boldsymbol{s}_1, \boldsymbol{s}_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} / \mu_{max} \quad (4)$$

Since $h(\boldsymbol{s}_1, \boldsymbol{s}_2)$ is a lower bound, the cumulative cost $q(\sigma_{s_1})$ and the cost-to-go $h(\boldsymbol{s}_1, \boldsymbol{s}_2)$ cannot contribute evenly to the distance $D_{opt}(\boldsymbol{s}_1, \boldsymbol{s}_2)$. If this were the case, the closest node (in the D_{opt} sense) to an arbitrary node $\boldsymbol{s}_j$ would always be the root node, as

$$h(\boldsymbol{s}_{root}, \boldsymbol{s}_j) \leq q(\sigma_{s_i}) + h(\boldsymbol{s}_i, \boldsymbol{s}_j) \quad \forall \boldsymbol{s}_i, \boldsymbol{s}_j$$

Instead, we give $h(\boldsymbol{s}_1, \boldsymbol{s}_2)$ a quadratic contribution to the total distance. This is meant to favor the selection of relatively close nodes, as the total cost rapidly increases with the distance.

When the objective is to find a feasible solution as quickly as possible, the optimization heuristic (3) is not used and the algorithm rather relies on the standard exploration heuristic (2). On the other hand, when the algorithm is allocated a fixed time window, it uses both heuristics at the same time. Indeed, prior to finding a solution, the exploration heuristic has a higher likelihood of being chosen, while the optimization heuristic is selected more often once a solution has been found. Combining both heuristics is useful, as performing optimization to improve the quality of the trajectories can be beneficial even before a solution is found. Similarly, using the exploration heuristic once a solution has been computed can sometimes help in finding a shortest path which was initially missed by the algorithm.

C. Selecting an Action

Line 6 of Alg. 1 selects an action $\boldsymbol{u}$ with a duration Δt which, once applied from the node $\boldsymbol{s}_{near}$, hopefully extends the tree toward the target configuration $\boldsymbol{s}_{rand}$. Obstacles are not considered here.

Since we know more or less precisely the trajectory followed by the robot when a certain action $\boldsymbol{u}$ is given, we can sample the action space with a strong bias toward efficient action vectors. Note that the robot's velocity μ_u should be reduced in the immediate vicinity of obstacles. For now, we disregard this constraint as we always set $\mu_u = \pm\mu_{max}(\boldsymbol{\lambda}_u)$, which instructs the robot to constantly travel at its maximum allowable velocity. Additionally, we do not require the robot's orientation θ to be tangent to the trajectory.

Let $\boldsymbol{p}_{near} = (x_{near}; y_{near})$ and $\boldsymbol{p}_{rand} = (x_{rand}; y_{rand})$ be the coordinates of the chassis position of respectively $\boldsymbol{s}_{near}$ and $\boldsymbol{s}_{rand}$. Given $\boldsymbol{p}_{near}$ and $\boldsymbol{p}_{rand}$, we can draw an infinite number of circles passing through the two points. Each circle expresses two different curved trajectories (two exclusive arcs) which can be followed by the robot to connect to the target configuration, assuming the robot's wheels are already aligned toward the circle's center. In this context, the sign of μ_u determines which arc (the smallest or the longest) the robot will travel on. All the centers of these circles lie on the bisector of the $[\boldsymbol{p}_{near};\boldsymbol{p}_{rand}]$ segment, which can be expressed in parametric form as

$$l_\lambda(k) = \frac{1}{2}(\boldsymbol{p}_{near} - \boldsymbol{p}_{rand}) + k\boldsymbol{v}$$

where $\boldsymbol{v} \cdot (\boldsymbol{p}_{near} - \boldsymbol{p}_{rand}) = 0$. This line therefore represents the set of ICR allowing a direct connection to the target configuration.

However, some of these ICR should be preferred over others, to avoid mode switches whenever possible. For this purpose, we restrain l_λ to the segment enclosing all ICR in the same mode as the source ICR, i.e., we find all k where $\text{mode}(l_\lambda(k)) = \text{mode}(\boldsymbol{\lambda}_{near})$. Doing so involves computing the intersection between l_λ and the four lines delimiting the different modes (see Fig. 3). If such a segment exists, we can sample directly a value $k_u \in \,]k_{min}; k_{max}[$ and set $\boldsymbol{\lambda}_u = l_\lambda(k_u)$, an acceptable ICR which avoids switching to another mode. However, this approach is not adequate, as all 2D points along the segment have equal likelihood of being selected. Indeed, we would prefer faraway points to have less coverage, since each of them corresponds to negligible variations of the wheel angles, and therefore negligible variations of the trajectories.

We address this problem by sampling an ICR on the unit sphere instead (depicted on Fig. 4). This is achieved by projecting the line segment on the sphere, which yields an arc of a great circle that can be parameterized by an angle $\lambda = \lambda(\phi)$, where $\phi \in [\phi_{min}; \phi_{max}]$. We then sample uniformly $\phi_u = Un(\phi_{min}, \phi_{max})$, from which we compute directly the desired ICR $\lambda_u = \lambda(\phi_u)$. By considering the relation between this angle and its corresponding point back on the global plane, one can see that the farther the point lies from the robot, the less likely it is to be selected. Hence sampling an angle along a great circle instead of a point on a line segment provides a more convenient ICR probability distribution.

Since we determined the values of λ_u and $|\mu_u|$, what remains to be decided are the sign of μ_u and Δt, the duration of the trajectory. The sign of μ_u is simply set as to always generate motions along the smallest arc of circle. We then calculate the duration of the path as the time needed to connect from $\boldsymbol{p}_{near}$ to $\boldsymbol{p}_{rand}$ along this arc of circle centered at $\boldsymbol{\lambda}_u$, given $|\mu_u| = \mu_{max}(\boldsymbol{\lambda}_u)$.

An overview of the algorithm is presented in Alg. 2. Note that besides introducing an additional probability P_l of generating straight lines, we allowed "naive" sampling to take place with a finite probability P_n, i.e., sampling an ICR without any consideration for the modes.

Algorithm 2 SelectAction Algorithm

```
1.  SELECTACTION($\boldsymbol{s}_{rand}, \boldsymbol{s}_{near}$)
2.    if $Un(0,1) < P_l$
3.      $\boldsymbol{\lambda}_u \leftarrow (u; v; 0)$ the ICR lying at infinity
4.    else
5.      find $l_\lambda(k) = k\boldsymbol{v} + \boldsymbol{m}$
6.      project $l_\lambda(k)$ on the sphere, yielding $\lambda(\theta) = (u, v, w)$
7.      if $Un(0,1) < P_n$
8.        $\boldsymbol{\lambda}_u \leftarrow \lambda(Un(0, \pi))$
9.      else
10.       find $[\theta_{min}, \theta_{max}]$ such that
              $\forall_{\theta \in [\theta_{min}, \theta_{max}]} mode(\lambda(\theta)) = mode(\boldsymbol{\lambda}_{s_{near}})$
11.       if $\nexists[\theta_{min}, \theta_{max}]$
12.         $\boldsymbol{\lambda}_u \leftarrow \lambda(Un(0, \pi))$
13.       else
14.         $\boldsymbol{\lambda}_u \leftarrow \lambda(Un(\theta_{min}, \theta_{max}))$
15.   $\mu_u \leftarrow \pm\mu_{max}(\boldsymbol{\lambda}_u)$, so as to choose the smallest arc of a circle
16.   $\Delta t \leftarrow$ time to reach $\boldsymbol{s}_{rand}$ on the arc of a circle
17.   return $(\boldsymbol{\lambda}_u; \mu_u)$ and $\Delta t$
```

TABLE I
PARAMETERS USED

$q(\sigma)$ (1)		D_{opt} (3)			
c_1	c_2	c_5	P_g	P_l	P_n
2.5	2.5	0.5	0.025	0.25	0.1

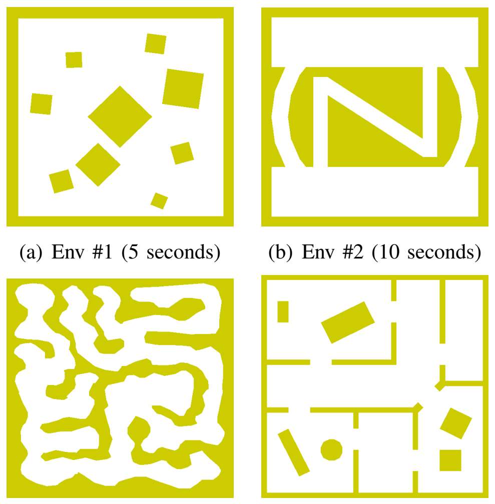

(a) Env #1 (5 seconds) (b) Env #2 (10 seconds)
(c) Env #3 (25 seconds) (d) Env #4 (20 seconds)

Fig. 5. Environments and time allocated for each query

V. RESULTS

Pending implementation on AZIMUT, experiments were performed within the OOPSMP[1] [12] library. Since the actual robot and our simulator both use the same kinematical model to compute the robot's motion, we expect the results obtained in simulation to be somewhat similar to real-world scenarios.

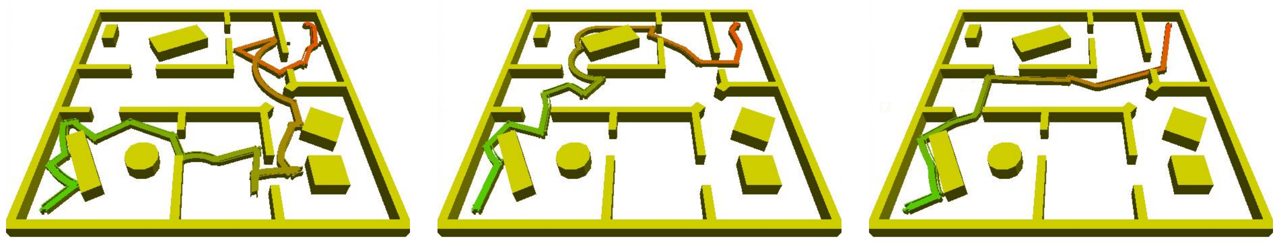
Table I summarizes the values used for the parameters described in (1), (3) and the different probabilities. The weighting factors c_1 and c_2 were both set to 2.5, which means every mode switch or reverse motion contributes an additional 2.5 seconds to the total cost. The exploration heuristic is selected 70% of the time prior to finding a solution, and 20% of the time after a solution has been found.

We compared our approach with a "naive" algorithm ignoring mode switches and reverse motions. This naive algorithm is a degenerate case of our main one, in the sense that it minimizes the metric (1) under the special condition $c_1 = c_2 = 0$ (duration only), and always selects an action naively, i.e., with $P_n = 1$. Other parameters remain the same.

An Intel Core 2 Duo 2.6 GHz with 4 GB of RAM was used for the experiments. The results are presented in Table II. Exactly 50 randomly generated queries had to be solved within each environment, with a fixed time frame allocated for each query. However, the time allocated was not the same for each environment, as some are more complicated than others (see Fig. 5) and we wanted to maximize the number of queries successfully solved.

The results show that the biased algorithm outperformed

[1]http://www.kavrakilab.rice.edu



(a) Trajectory using a completely random sampling

(b) Trajectory using a naive sampling

(c) Trajectory using our proposed sampling

Fig. 6. Comparison of trajectories created by a random, a naive, and a biased algorithms

TABLE II
COMPARISON OF A NAIVE ALGORITHM AND OUR PROPOSED SOLUTION

	Algorithm	Travel Time	Mode switches	Reverse motions	Metric evaluation (1)
Env #1	Naive	36.36	4.71	0.71	49.91
	Biased	32.41	2.65	0.46	40.19
Env #2	Naive	38.09	6.46	0.46	55.39
	Biased	33.46	3.41	0.45	43.11
Env #3	Naive	48.70	12.96	1.19	84.08
	Biased	45.18	10.48	1.92	76.18
Env #4	Naive	60.42	5.18	0.84	75.47
	Biased	51.72	3.29	0.52	61.25

the naive one on all environments. Indeed, by minimizing the number of mode switches and reverse motions, the biased algorithm not only improves the fluidity of the trajectories, but also decreases the average travel time. In heavily cluttered environments like Env #3, feasible trajectories are impaired by a large number of mode switches. To avoid these mode switches, the biased algorithm had to increase the number of reverse motions, which explains the slightly worse result for this particular element. Fig. 6 presents examples of typical trajectories generated by the different algorithms – including an algorithm selecting an ICR randomly. The first two include several steep turns corresponding to undesirable mode switches and reverse motions, whereas the last one is smoother and appears more natural. However, it is important to note that our algorithm does not always produce good-looking trajectories, as guarantees of quality are hard to obtain via nondeterministic approaches.

VI. CONCLUSION

A new RRT-based algorithm for the motion planning of nonholonomic omnidirectional robots has been presented. It has been shown that by taking explicitly into account the kinematic constraints of such robots, a motion planner could greatly improve the fluidity and efficiency of trajectories.

We plan to expand our RRT-based motion planner to constrain the orientation of the robot's chassis, and to adapt the robot's velocity according to the proximity with obstacles. We are also interested in computing a robust feedback plan so that the robot does not deviate too much from the planned trajectory, despite the inevitable real world unpredictability. For AZIMUT, this would involve the additional challenge of making sure the ICR stays as far as possible from the control zones frontiers, as to avoid undesired mode switches. Future work will integrate these additional elements.

ACKNOWLEDGMENTS

This work is funded by the Natural Sciences and Engineering Research Council of Canada, the Canada Foundation for Innovation and the Canada Research Chairs. F. Michaud holds the Canada Research Chair in Mobile Robotics and Autonomous Intelligent Systems.

REFERENCES

[1] F. Michaud, D. Létourneau, M. Arsenault, Y. Bergeron, R. Cadrin, F. Gagnon, M.-A. Legault, M. Millette, J.-F. Paré, M.-C. Tremblay, P. Lepage, Y. Morin, J. Bisson, and S. Caron, "Multi-modal locomotion robotic platform using leg-track-wheel articulations," *Autonomous Robots*, vol. 18, no. 2, pp. 137–156, 2005.

[2] M. Lauria, I. Nadeau, P. Lepage, Y. Morin, P. Giguère, F. Gagnon, D. Létourneau, and F. Michaud, "Design and control of a four steered wheeled mobile robot," in *Proc. of the 32nd Annual Conference of the IEEE Industrial Electronics*, 7-10 Nov 2006, pp. 4020–4025.

[3] G. Campion, G. Bastin, and B. d'Andréa-Novel, "Structural properties and classification of kinematic and dynamic models of wheeled mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 1, pp. 47–62, 1996.

[4] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[5] G. Sanchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," in *Proc. of the International Symposium on Robotics Research*, 2001, pp. 403–417.

[6] J. van den Berg and M. Overmars, "Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners," *International Journal on Robotics Research*, pp. 1055–1071, 2005.

[7] H. Kurniawati and D. Hsu, "Workspace importance sampling for probabilistic roadmap planning," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[8] B. Burns and O. Brock, "Sampling-based motion planning using predictive models," in *Proc. of the IEEE International Conference on Robotics and Automation*, 2005.

[9] D. Hsu, J.-C. Latombe, and H. Kurniawati, "On the probabilistic foundations of probabilistic roadmap planning," in *Proc. of the International Symposium on Robotics Research*, 2005.

[10] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *AIAA Journal on Guidance, Control on Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[11] A. Yershova, L. Jaillet, T. Simeon, and S. LaValle, "Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain," in *Proc. of the IEEE International Conference on Robotics and Automation*, 2005.

[12] E. Plaku, K. Bekris, and L. E. Kavraki, "OOPS for motion planning: An online open-source programming system," in *Proc. of the IEEE International Conference on Robotics and Automation*, 2007, pp. 3711–3716.