Noname manuscript No.
(will be inserted by the editor)

Optimal Navigation for a Differential Drive Disc Robot: A Game Against the Polygonal Environment

Rigoberto Lopez-Padilla, Rafael Murrieta-Cid, Israel Becerra, Guillermo Laguna and Steven M. LaValle

the date of receipt and acceptance should be inserted later

Abstract This paper considers the problem of globally optimal navigation with respect to minimizing Euclidean distance traveled by a disc-shaped, differential-drive robot (DDR) to reach a landmark. The robot is equipped with a gap sensor, which indicates depth discontinuities and allows the robot to move toward them. In this work we assume that a topological representation of the environment called GNT has already been built, and that the landmark has been encoded in the GNT. A motion strategy is presented that optimally navigates the robot to any landmark in the environment, without the need of using a previously known geometric map of the environment. To our knowledge this is the first time that the shortest path for a DDR (underactuated system) is found in the presence of obstacle constraints without knowing the complete geometric representation of the environment. The robot's planner or navigation strategy is modeled as a Moore Finite State Machine (FSM). This FSM includes a sensor-feedback motion policy. The motion policy is based on the paradigm of avoiding the state estimation to carry out two consecutive mappings, that is, from observation to state and then from state to control, but instead of that, there is a direct mapping from observation to control. Optimality is proved and the method is illustrated in simulation.

Keywords combinatorial filters · optimal navigation · underactuated system · environment constraints · feedback motion strategies

A preliminary version of portions of this work has been presented at the Tenth International Workshop on the Algorithmic Foundations of Robotics, WAFR 2012 [25]. This work was supported in part by NSF grants 0904501 (IIS Robotics) and 1035345 (Cyberphysical Systems), DARPA SToMP grant HR0011-05-1-0008, and MURI/ONR grant N00014-09-1-1052.

Rigoberto Lopez-Padilla
Centro de Innovación Aplicada en Tecnologías Competitivas, CIATEC, León, México
E-mail: rlopez@ciatec.mx

Rafael Murrieta-Cid and Israel Becerra
Centro de Investigación en Matemáticas (CIMAT), Guanajuato, México
E-mail: {murrieta,israelb}@cimat.mx

Guillermo Laguna
Iowa State University, Ames, IA, USA
E-mail: glaguna@iastate.edu

Steven M. LaValle
University of Illinois at Urbana-Champaign, Urbana, IL, USA
E-mail: lavalle@illinois.edu

1 Introduction

This paper considers the problem of globally optimal navigation with respect to minimizing Euclidean distance traveled by a disc-shaped, differential-drive underactuated robot to reach a landmark in an environment with obstacle constraints.

The robot observes the world mainly using a *gap sensor*, introduced in [39], which allows it to determine the directions of discontinuities in depth (distance to the boundary of the environment) and move toward any one of those directions. Under this model, but for a *point robot*, a combinatorial filter called the Gap Navigation Tree (GNT) was introduced in [38, 39], it encodes precisely the part of the shortest-path visibility graph that is needed for optimal navigation. The GNT can also be considered a topological map. A topological map can be represented as a graph, in which the vertices depict particular sensor readings and configurations and the edges refer to the controls between two different configurations. The GNT differs from previous approaches in that it is a local representation, defined for the current position of the robot, rather than a global one. The learned data structure corresponds exactly to the shortest path tree [12] from the robot's location. This enables the

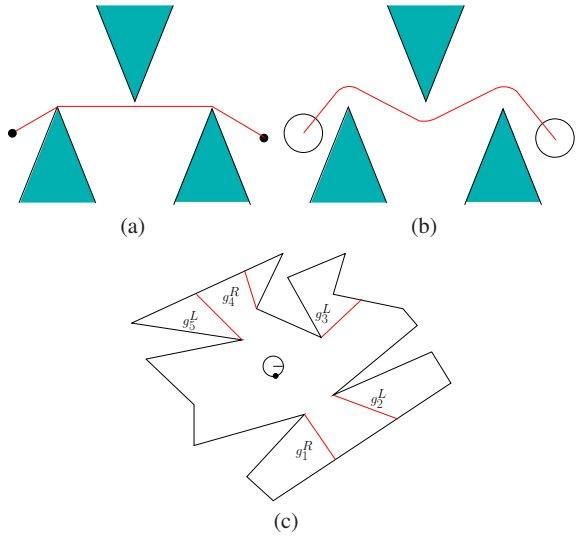


Fig. 1 a) The optimal path for a point robot, b) The optimal path for a disc robot, c) The gap sensor (attached at the small solid disc on the robot's boundary) detects the sequence of gaps $G = [g_1^R, g_2^L, g_3^L, g_4^R, g_5^L]$, in which g_1^R and g_4^R are right (near-to-far) gaps and g_2^L, g_3^L, and g_5^L are left (far-to-near) gaps.

robot to navigate to any previously seen landmark by following the shortest-distance path, even though it cannot directly measure distances. This modeling avoids building a geometric map of the environment. Furthermore, the robot does not need to localize itself with respect to a *global* reference frame. The GNT was extended and applied to exploration in [30]. The GNT was extended to point cloud models in [19]. A larger family of gap sensors is described in [21].

The case of a disc robot is important because real robots have nonzero width. It is therefore interesting to study the case of a disc robot, which could correspond, for example, to a Roomba platform. Unfortunately, the problem is considerably more challenging because without additional sensing information, the robot could accidentally strike obstacles that poke into its swept region as it moves along a bitangent. See Fig. 1 (a) and (b). In navigation, the robot must instead execute detours from the bitangent. Sensing, characterizing, and optimally navigating around these obstructions is one of the contributions of this paper. The method for generating optimal navigation motions is modeled as a Moore Finite State Machine (FSM).

Fig. 2 shows bitangents and bitangent complements, following the presentation of [39], a bitangent is identified with connected open sets $I \in \partial E$ and $J \in \partial E$, where ∂E is the boundary of the polygonal environment. A pair of disjoint connected open sets I and J identify a *bitangent* if at least one point of I is visible to one point of J and if there is a line L that partitions I and J into sets I_1, I_2 and I_3, and J_1, J_2 and J_3 respectively, such that: 1) I_1 is an open set that does not intersect L (the same for J_1); 2) I_2 is a closed subset of L (the same for J_2); and 3) I_3 is an open set that does not intersect

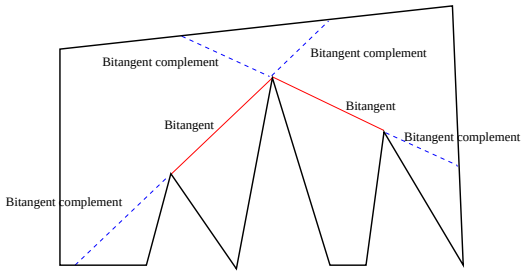


Fig. 2 bitangents and bitangent complements

L and that lies on the same side of L from I_1 (the same for J_3 and J_1).

It is important to keep in mind that the robot is placed into an environment with obstacles, but it is not given the obstacle locations or its own location and orientation. The robot *observes* this information over local reference frames. The purpose is to show how optimal navigation is surprisingly possible without a geometric map.

However, in this paper, we assume that the GNT representing the environment has been built and that a landmark is encoded in the GNT. Either the method for building the GNT with a point robot [38,39] or with a disc robot [18] can be used. The works in [38,39] presented the original GNT, a method to explore the environment and to encode a landmark in it (to come back later to the landmark) have also been presented. Since the robot was assumed to be a point robot, then any visible point in the environment was also reachable by the robot. In [18], the robot is assumed to be a disc, hence, even if the robot can see certain place within the environment, that place might not be reachable for the robot. The exploration problem addressed in [18] is more challenging than the case of a point robot because visibility information does not provide collision free paths in the configuration space. The method proposed in [18] guarantees exploring the whole environment or the largest possible region of it and the disc robot is able to find a landmark and encode it in the GNT or declare than an exploration strategy for this objective does not exist. Note that differently to [18] where the main problem is to explore the environment and construct the GNT encoding it, in this work the problem is to optimally navigate toward the landmark.

Once the GNT has been built, the approach proposed in this paper is able to determine whether or not a collision free path exists to reach the landmark and if a path exists, it proposes the motion controls to reach the landmark.

1.1 Related work and main contributions

Our work is related to the problem of planning robot's paths that avoid collision with obstacles [16,6,27], and particularly with underactuated nonholonomic robots [20,3,15]. Our problem is also related to the problem of finding optimal

paths for nonholonomic robots [2,33,40]. The study of optimal paths for non-holonomic systems has also been an active research topic. Reeds and Shepp determined the shortest paths in an environment without obstacles for a car-like robot that can move forward and backward [31]. In [33], a complete characterization of the shortest paths for a car-like robot is given for an environment without obstacles. In [2], Balkcom and Mason determined the time-optimal trajectories for a Differential Drive Robot (DDR) using Pontryagin's Maximum Principle (PMP) and geometric analysis also for an environment without obstacles. In this work, we present a motion strategy that minimizes the Euclidean distance traveled by a disc-shaped, differential-drive underractuated robot to reach a landmark, in a simply connected polygonal environment.

The approaches described above to find shortest paths with nonholonomic systems assume that the robot moves in an environment without obstacles [33,2], in this work the robot moves in an environment with obstacle constraints and the robot does not have a geometric map of the environment as in [20,15] to avoid collision with the obstacles, the robot discover the obstacles with its sensors.

In [1], the authors study the problem of finding the shortest path for a point robot, between two configurations (position and orientation) in a convex polygon. The robot path is constrained to have curvature at most 1. The authors propose an algorithm for determining whether a collision-free path exists for the point robot between two given configurations. If such a path exists, the algorithm returns a shortest one. In this work, we find the shortest path for a disc shaped differential drive robot DDR (a nonholonomic under-actuated system, but that is able to rotate in place) for any simple connected polygonal environment (convex or not) and without having the exact geometric map of the environment.

Computing shortest paths for a point robot, without nonholonomic constraints, placed into a known polygonal region is straightforward. The most common approach is to compute a visibility graph that includes only bitangent edges, which is accomplished in $O(n^2 \lg n)$ time by a radial sweeping algorithm [10] (an $O(n \lg n + m)$ algorithm also exists, in which m is the number of bitangents [13]). An alternative is the *continuous Dijkstra method*, which combinatorially propagates a wavefront through the region and determines the shortest path in $O(n \lg n)$ time. Numerous problem variations and results exist. Computing shortest paths in three-dimensional polyhedral regions is NP-hard [7]. Allowing costs to vary over regions considerably complicates the problem [29,32]. See [12,28] for surveys of shortest path algorithms. For recent efforts on curved obstacles, see [8].

Once the robot has nontrivial dimensions, the problem can be expressed in terms of configuration space obstacles. Solutions for finding shortest paths assuming that the map is known, are presented in [9,24]. Given strong sensors and good odometry, standard SLAM approaches [11,37] could be applied to obtain a geometric map.

Once a polygonal map is obtained, an alternative to find the shortest path for a disc robot is to expand the obstacles by the robot radius and reason over this extended polygonal map. But that approach requires to built first a precise polygonal map of the environment, which is a hard problem. One main advantage of the approach presented in this work is that it does not require that polygonal map. Even if the complete polygonal map is available and the obstacle expansion is done, the resulting configuration space representation is not observable or measurable directly by the robot sensors. Other main advantage of the approach proposed in this paper is that information from the workspace is obtained directly from the robot's sensors, to infer the optimal robot paths in the configuration space.

Other type of environment's representations are the topological maps in the form of graphs [36,39,17,4]. In these graphs the nodes represent environment places and the edges represent adjacency. The problem of exploring an unknown environment for searching of one or more recognizable targets is considered in [36]. In [17], the authors propose a surveillance graph to model surveillance tasks performed by teams of robots having limited sensing capabilities. A surveillance graph is automatically extracted from occupancy grid maps. The work in [17] represents an effort to close a loop between a graph-based theoretical formulation and practical scenarios. In [4], the authors study the problem of determining the minimal information required by a robot to reconstruct the visibility graph of an initially unknown polygon. The authors allow a robot to collect sensory input while it is located at a vertex. Vertices are not identified globally, the vertex can be distinguished only in a local sense by a relative position. The GNT has been used in several other works (e.g. [38,39],[14], [30] and [18]). In the presented work, we still use the GNT to establish connectivity in the workspace between the robot and the landmark. However, note that we present new contributions that are independent of the GNT (see below).

Considering the problem of globally optimal navigation for a disc robot, various objective functions are possible, for instance time or energy spent by the robot. However, we choose to minimize the distance traveled by the center of the robot because it turns out that such a criterion establishes the existence of a collision free path for the worst scenario, that is, the environments with the most narrow passage, such that, this passage is wider than the robot's diameter. Hence, the problem addressed in this paper can be seen as a *game* [23] against the polygonal environment. The rationale behind this result is as following: our motion strategy moves the center of the robot as less as possible, in other words the volume of the space that the robot sweeps as it moves,

is as small as possible, hence the proposed motion strategy requires the smallest free space region to move the robot.

The main contributions of this work are theoretical and have not been presented in our related previous works [38, 39, 18]; we consider that they are the following:

1. The original GNT guarantees that a point robot will travel the shortest Euclidian distance path to a landmark. In this work, we extend the work to find the shortest path for a disc robot, between any initial robot configuration and the landmark. The GNT is used for establishing connectivity between a point robot and the landmark, but note that a disc robot could strike obstacles that poke into its swept region as it moves along a bitangent. In navigation, the disc robot must execute detours from the bitangent (see Figure 1). In this work, we present a Finite State Machine that commands the robot to execute those detours directly maping observations into controls and yielding the shortest path for a disc shaped robot.
2. We consider a differential drive robot (DDR) nonholonomic under actuated system. In the original GNT, nonholonomic constrains over the robotic platform have not been considered.
3. We have shown, that the criterion that we propose to optimize, that is minimizing the Euclidean distance traveled by the center of the robot, gives as a consequence the existence of a geometric solution for the navigation problem of finding a path towards the landmark. Hence if the proposed motion strategy does not find a collision free path to the landmark then such path does not exist.

The content of the paper is organized as follows: Section 2 formally presents the problem statement, and the robot's model including motion and sensing capabilities. Section 3 presents an observation vector that is used to decide the robot action. Section 4 describes the FSM that is used for generating optimal robot navigation. Section 5 presents a feedback motion policy that maps observations to robot's commands. Section 6 argues the optimality of the motion strategy. Section 7 presents an implementation in simulation, and Section 8 concludes the paper.

2 Problem statement

The robot is a differential drive (underactuated) system having a defined forward heading. The robot has the shape of a disc with radius r moving in a planar and polygonal environment which could be any compact set $E \subset \mathbb{R}^2$ for which the interior of E is simply connected. The boundary ∂E of E is the image of a piecewise-analytic closed curve. However, it is assumed that the collision-free subset of the robot's configuration space is simply connected or it might have several connected components. C-space obstacle projected in the plane corresponds to that of a translating disc, that is, the extended boundary of E which is due to the robot radius [1].

Let Λ be a static landmark located in the environment having the shape of a disc with the same radius as the robot. Let assume that Λ is painted on the ground, hence it does not have volume and it does not produce distance discontinuities (gaps). This assumption is made since the robot has as its goal to park on the landmark. In this work we assume that a GNT representing the environment has already been built, and that the landmark has been encoded in the GNT.

One objective of this paper is to determine whether or not a collision free path toward the landmark Λ exists, if a path exists then a second objective is to determine commands to travel the shortest path toward the landmark in finite time.

2.1 The GNT

The GNT is an efficient data structure that dynamically changes according to some critical events until the whole environment has been discovered.

The GNT can be constructed incrementally as the robot moves along a path τ. Initially, the GNT consists of a root node that is connected to one leaf node for every gap in $G(\tau(0))$. Each time t at which a change in $G(\tau(t))$ occurs corresponds to a critical event. This requires updating the GNT. There are four different kinds of critical events:

- A new gap g *appears*: A node g is added as a child of the root, while preserving the cyclic ordering from the gap sensor (see Fig. 3 (a)). For a description of the gap sensor see Section 2.2.
- A gap g *disappears*: The node g, which must be a leaf, is removed (see Fig. 3 (b)).
- Gaps g_1 and g_2 *merge* into g: Nodes g_1 and g_2 become children of a new node, g, which is added as a child of the root and preserving the ordering of gaps (see Fig. 3 (c)).
- Gap g_1 *splits* into g_2 and g_3: If g_1 is a leaf node, then g_2 and g_3 become new nodes;otherwise, they already exist as children of g_1. Both g_2 and g_3 are connected to the root, preserving the ordering of gaps and removing g_1 (see Fig. 3 (d)).

If any leaf vertex has the potential to split, then the GNT is incomplete because it could expand, some gaps split and other gaps simply disappear. The gaps that disappear are called primitive (their corresponding nodes in the GNT are also called primitive). If all the leaf nodes of the GNT are primitive, then the GNT is said to be complete. Indeed, the following Lemma in [39] guarantees the termination of the GNT's construction.

[1] Note that this is the configuration space for a translating disc rather than for a rigid body because of rotational symmetry.

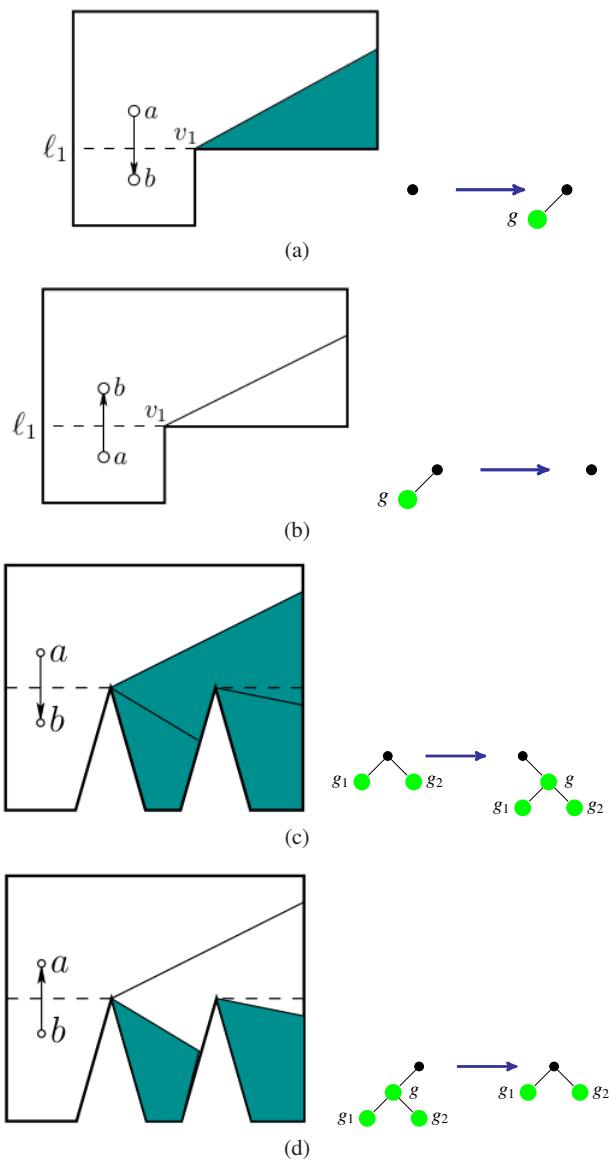


Fig. 3 Critical events: (a) Gap appears, (b) Gap disappears, (c) Gap merge, (d) Gap splits.

Lemma 1 *The procedure of iteratively chasing non-primitive leaves terminates with a resulting complete GNT.*

For the proof please refer to [39].

The GNT can be extended to encode landmarks in it. If a landmark disappears behind a gap g, then it is added to the GNT in the node corresponding to g (see Fig. 4). The robot can return to any previously visible landmark by traveling to the node g until the landmark is visible.

The next theorem states that once completely constructed, the extended GNT can be used for navigation from the current position of the robot to any landmark in the environment.

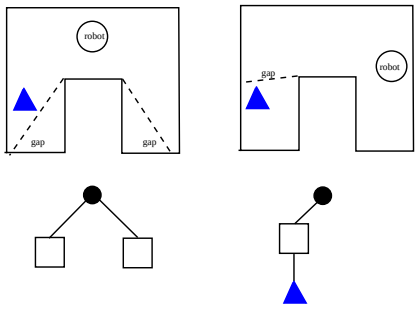


Fig. 4 A landmark encoded in a node of the GNT.

Theorem 1 *The extended GNT encodes a path to any object or landmark in the environment from the current position of the robot.*

The proof is presented in [39].

2.2 Sensing model

The robot has an omnidirectional sensor, which is used to sense the environment. The omnidirectional sensor is also able to detect and track discontinuities in depth information (gaps). Hence, over the omnidirectional sensor, it is possible to build a gap detector, further referred as the gap sensor. The robot uses a side sensor that is a laser pointer use to measure distance in a particular direction. The robot is also equipped with a contact sensor that is able to tell the robot when it is in contact with the environment in particular points over the robot boundary. Finally the robot has a landmark detector sensor that tells the robot that it has reached the landmark.

The sensor model is minimalist in the sense that the navigation strategy described in Section 4 makes use of information provided by these four sensors and if any component is taken away then the robot will fail to solve its task. On the other hand, it is interesting to notice that navigation strategy only needs the information provided by those sensors, so any component more powerful is unnecessary.

Furthermore, the sensing components are standard in the literature and can be implemented using existing technology. For instance using laser range finders [39], laser pointers [26], tactile bumpers [5] and for detecting the landmark a color [5] or line detectors [35]. Below we describe in more detail the sensing requirements.

1) Gap sensor: The omnidirectional sensor [21,39] is able to detect and track two types of discontinuities in depth information (considering a counterclockwise direction along ∂E): discontinuities from far to near and discontinuities from near to far (see Fig. 1(c)). Let $G = [g_1^t, ..., g_k^t]$ denote the circular sequence of gaps observed by the sensor. Using this notation, t represents the discontinuity type, in which $t = R$ refers to a *right gap* where the hidden environment's portion

is to the right (a discontinuity from near to far), and $t = L$ refers to a *left gap* where the hidden environment's portion is to the left (a discontinuity from far to near). For example, the gap sensor in Fig. 1(c) detects gaps of different types: $G = [g_1^R, g_2^L, g_3^L, g_4^R, g_5^L]$.

We place the gap sensor on the robot boundary and define motion primitives that *during navigation* send the robot on collision-free trajectories that possibly contact the obstacles (moving along the boundary of the free subset of the configuration space, that is semi-free trajectories, is necessary for optimal paths). These motion primitives, described in detail in Section 2.4, allow the robot to rotate itself so that it is aligned with a desired gap, to move forward while chasing a gap, and to follow ∂E while the sensor is aligned to a gap.

It is assumed that the gap sensor can be moved to two different fixed positions on the robot's boundary: the extremal left and right sides with respect to the forward wheel direction. One way to implement this is with a turret that allows the robot to move the gap sensor from its right side to its left side and vice-versa. Fig. 6(c) shows the sensor aligned to a right gap in which the gap sensor is on the right side of the robot. To align the sensor to a left gap, the robot moves the gap sensor to the left side of the robot. The omnidirectional sensor is able to measure distance and angles to the vertices that generate gaps. Let d_u be the distance between the omnidirectional sensor and the vertex u_i that originated the gap g_i (in Fig. 5(d) $g_i = g_0^R$).

2) Side sensors: To detect obstacles that obstruct the robot, our method needs to measure distances between the extremal left and right side robot's points along the direction of the robot heading (forward) and the obstacles. Let those particular robot points be left side point lp and right side point rp. The particular forward direction tangent to the robot's boundary at rp is called rt. The particular forward direction tangent to the robot boundary at lp is called lt (see Fig. 5(a)). Note that based on distance the discontinuities can be detected. Let d_R be the distance between rp and the obstacles at the particular direction rt, and d_L be the distance between lp and the obstacles at the particular direction lt (see Fig. 5(b)).

If the particular direction, either rt or lt, is pointing to a reflex vertex[2] (a gap is aligned with this direction), then a discontinuity in the sensor reading at this direction occurs. Let $d_R{}^t$ denote the distance from rp to the closer point along the discontinuity direction in the boundary of environment ∂E. Similarly, $d_L{}^t$ denotes the distance from lp. See Fig. 5(c).

Sensor errors are not considered in this work. However, note that our motion strategies will require only comparisons of distances to determine which is larger, rather than needing precise distance measurements.

[2] A reflex vertex is a polygon vertex of an internal angle greater than π.

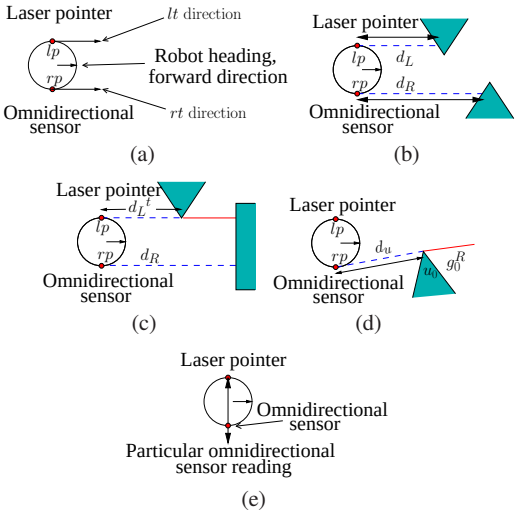


Fig. 5 Side Sensors: (a) Points rp and lp, and directions rt and lt, (b) d_L and d_R, (c) $d_L{}^t$, (d) d_u, (e) Omnidirectional sensor readings for contact detection.

3) Contact sensors: Our approach requires detecting whether the robot is contacting ∂E at rp or lp.

4) Landmark reached sensor: Our approach also requires a sensor that detects that the landmark has been reached.

Distance measurements between the obstacles and rp and lp in directions rt and lt (forward), and the information of whether the robot is touching ∂E at rp or lp, can be obtained with different sensor configurations. For example, it is possible to use two laser pointers and two contact sensors, each of them located at rp and lp. However, to use a smaller number of sensors and facilitate the instrumentation of the robotic system, it is possible to emulate both the contact sensors and one of the laser pointers, using the omnidirectional sensor. The omnidirectional sensor reading in the particular forward robot heading direction emulates the laser pointer reading. The sensor readings at directions perpendicular to the robot heading are used in this case. If the robot is touching ∂E at the point at which the omnidirectional sensor is located, then the sensor reading is zero. If robot is touching ∂E at the point diametrically opposed to the omnidirectional sensor, then the sensor reading will correspond to the robot diameter (see Fig. 5(e)). Thus, one option is to have the robot equipped with an omnidirectional sensor and a laser pointer; they will be located at lp and rp. Recall that a turret can be used to swap the locations of the laser pointer and the omnidirectional sensor to avoid unnecessary robot rotations in place. The turret moves on the boundary of the robot. The gaps or landmarks are always chased with the omnidirectional sensor; the laser pointer is used to help to detect obstacles (see below). Also to detect that the landmark has been reached several different hardware implementation might be used, for instance using the omnidirectional sensor.

An observation vector is obtained with the robot's sensors, this vector is used to decide the robot action. In Section 3 this observation vector is described in detail.

2.3 Landmark encoding

In [18], the landmark is said to be recognized if Λ is visible at least partially from the location of the omnidirectional sensor. In [18], if the landmark is totally visible (fully contained in the visibility polygon $V(q)$) from the omnidirectional sensor location then the landmark is encoded in the GNT as a node child of the root. If the landmark is totally or partially occluded from the omnidirectional sensor location then the landmark is encoded as a node child of the node representing a gap in the GNT [39].

Let rp_Λ be an extremal point on the landmark such that whenever rt is aligned to rp_Λ the body of the landmark is to left of particular direction rt. There is an analogous definition for point lp_Λ.

If a reflex vertex occludes point rp_Λ then the Λ is encoded with the gap generated by the vertex. Similarly, if a reflex vertex occludes point lp_Λ then it is also encoded with the gap generated by the vertex. This is the encoding used in [18], hence the landmark Λ can be encoded at most with two gaps. The navigation strategy proposed in this paper is able to find the path that minimizes the Euclidian distance to reach the landmark.

2.4 Motion model

The robot navigates using a sequence of motion primitives that are generated by an automaton for which state transitions are induced by sensor feedback alone. To navigate, a gap (or equivalently the vertex that generates it) or a landmark is given to the robot as goal. There are five motion primitives (see Fig. 6). Let the angular velocity of the right and left wheels be w_r and w_l, respectively, with $w_r, w_l \in \{-1, 0, 1\}$. Thus, the motion primitives are generated by the following controls:

- Clockwise rotation in place: $w_r = -1, w_l = 1$.
- Counterclockwise rotation in place: $w_r = 1, w_l = -1$.
- Clockwise rotation w.r.t. to point rp: $w_r = 0, w_l = 1$.
- Counterclockwise rotation w.r.t. to point lp: $w_r = 1, w_l = 0$
- Forward straight line motion: $w_r = 1, w_l = 1$.

In this work, we propose the simple controls described above, which produce the three needed motion primitives (straight line motion, rotation in place and rotation w.r.t. point rp or lp) that yield the shortest path in terms of the Euclidian distance. The rotation primitives are used to align rt or lt to a specific gap (or landmark). Once rt or lt is aligned to a gap, the robot moves in a straight line to chase the gap. If the path to the chosen gap is blocked, then the robot executes a detour by choosing a new vertex as a subgoal. More details are given in Section 4.

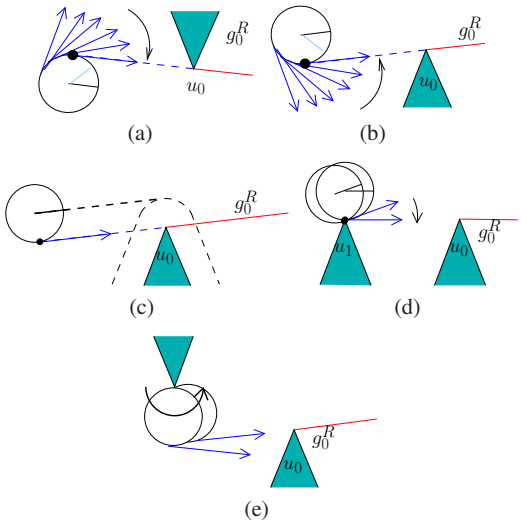


Fig. 6 The motion primitives: (a) Clockwise rotation in place, (b) Counterclockwise rotation in place, (c) Straight line motion, (d) Clockwise rotation w.r.t. to point rp, (e) Counterclockwise rotation w.r.t. to point lp.

It would be interesting to consider some suitable properties of the robot controls. Since the optimization criterion is the Euclidian distance and not time, then to preserve optimality is not mandatory to travel at saturated maximal speed. This gives flexibility in how to execute the motion primitives and concatenate them. We leave for future work to include suitable properties over the robot controls. For instance, to include smooth transitions between controls or to propose control laws able to deal with noise.

3 Observation vector

The GNT encodes the shortest path to any place in the environment for a point robot, it also encodes the robot's goal. However, for a disc robot this information is not enough for optimal navigation. The observation vector provides the information that the robot needs to make decisions about the actions yielding optimal motion. First, some useful definition are presented.

Definition 1 The gap that encodes the path toward the landmark in the GNT is called a goal gap g_{goal}. It is encoded in a node child of the root in the GNT.

Definition 2 A goal vertex u_{goal} is the vertex that generates the goal gap. A goal vertex is visible to the omnidirectional sensor.

Definition 3 A sub-goal vertex is the next vertex to be visited in the optimal path for a disc robot, when a detour to the goal vertex is required.

Definition 4 A vertex that generates a right gap, is called a right vertex.

Definition 5 A vertex that generates a left gap, is called a left vertex.

We will also refer to a candidate vertex, which is a vertex that might become a sub-goal vertex. The precise conditions that define a candidate vertex are given in definition 6.

The observation vector yn_i has 14 binary observations, in the list below a description of each one is presented. Some of them are more complex and they are described with more detail.

1. FI: the landmark Λ is reached (1) or not (0).
2. LV: the landmark is totally visible from the omnidirectional sensor location (1) or not (0).
3. FR: the robot is aligned by the first time to a sub-goal vertex (1) otherwise (0).
4. RP: the robot is touching ∂E with point rp (1) or not (0).
5. LP: the robot is touching ∂E with point lp (1) or not (0).
6. VR: the vertices generating right gaps have been located over a local reference frame (1) or not (0).
7. VL: the vertices generating left gaps have been located over a local reference frame (1) or not (0).
8. AL: the robot is aligned to a given vertex or to a landmark (1) or not (0).
9. BL: there is a blockage toward a given vertex or a landmark (1) otherwise (0).
10. UN: the goal gap has merged (1) or not (0).
11. GT: the type of goal gap, right (1) or left (0).
12. CT: the type of the candidate vertex, right (1) or left (0).
13. O1: sensor location (two bits are needed to establish the sensor location).
14. O2: sensor location.

VL and VR: To find a candidate vertex, it is necessary to locate the vertices in *local reference frames*. There are four types of local reference frames. Two of them are just the symmetric cases of the other two. We describe the two basic local reference frames in Appendices A and B.

VR = 1 when the locations of right vertices has been computed, otherwise VR=0. VL = 1 when the locations of left vertices has been computed, otherwise VL=0.

Robot alignment AL: The robot might be aligned to a given vertex or to a landmark. The robot aligns direction rt to a right vertex. Symmetrically, the robot aligns direction lt to a left vertex. To chase a landmark, the robot aligns lt with lp_Λ or rt with rp_Λ.

Blockage BL: This bit is (1) if there is a blockage toward a given vertex or a landmark and (0) otherwise. A given vertex or landmark is blocked if the robot cannot travel in straight line toward it. A formal definition for blockage conditions is given in Section 4.

There are four ways to detect a blockage:

1. The first one takes places when the robot is aligned to a vertex. To detect the blockage, distances d_L, d_R, ${d_L}^t$ and ${d_R}^t$ are used. If direction rt is aligned to a right vertex and ${d_R}^t > d_L$ then a straight line robot path toward this vertex is blocked. Likewise, if direction lt is aligned to a left vertex and ${d_L}^t > d_R$ then a straight line robot path toward that vertex is blocked.
2. The second way to detect a blockage is used when the robot is not aligned to a given vertex. It might happen that the robot cannot align either rt or lt to the goal vertex, because this motion will produce an unnecessary robot translation, and the global optimality would be lost. In Fig. 7, if the robot rotates counterclockwise with respect to rp then the center of the robot will move backward and optimality would be lost. Whenever the robot is touching a vertex with point rp to align rt with the goal vertex u_{goal} or landmark, the maximal angle measured in clockwise sense from rt to the goal vertex can never be larger than π, thus, if this angle is larger than π a blockage is detected, as a robot collision necessarily occurs (robot touches ∂E with a point different to rp).

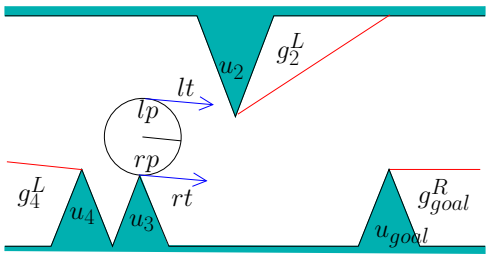


Fig. 7 Blockage type 2

There is a symmetric blockage detection whenever the robot is touching a vertex with point lp.

3. The third way to detect a blockage requires several measurements, this detection type is equivalent to fulfill the blockage condition defined in Section 4.
4. For the fourth way to detect a blockage, the robot is aligned to a landmark and the landmark is totally visible from the omnidirectional sensor location. The distance from the omnidirectional sensor to the landmark cannot be sensed. Consequently, the path to reach the landmark is declared as blocked, since a turret motion must be executed to establish whether or not the landmark can be reached without visiting first a vertex. Thus, a Landmark

blockage is detected using visibility. If the landmark is totally visible from the omnidirectional sensor located at both points rp and lp then the path toward a landmark is not blocked. Case 2-R and Case II-RP (see subsection 4.2) present the procedures for alignment with a landmark.

Sensor Location O1 and O2: We use two bits to establish the omnidirectional sensor location. O1=0 and O2=0 establishes that the omnidirectional sensor is at lp, O1=0 and O2=1 establishes that the omnidirectional sensor moves from lp to rp, O1=1 and O2=0 establishes that the omnidirectional sensor is at rp, and O1=1 and O2=1 establishes that the omnidirectional sensor moves from rp to lp.

4 Finite State Machine for Optimal Navigation

The robot navigates toward a landmark by moving toward gaps based on the GNT. The GNT in [39] was designed for a point robot, for a disc robot, the robot must execute detours from the bitangent lines between vertices. At the end, the navigation consists in visiting goal vertices. However, the straight line path to a goal vertex might be blocked, in such a case a sub-goal vertex is found. The vertex that generates the goal gap, which is encoded in the GNT, is always visited. For this reason, we call the modified path a detour. A detour starts whenever a goal-vertex is blocked and it ends when the sub-goal vertex is reached.

In this section a Finite State Machine (FSM) for optimal navigation is provided. A main objective of this FSM is to find the sub-goal vertex. Once the sub-goal vertex is found, the robot rotates to get aligned with that vertex and then it moves in straight line to reach it. There are two types of rotation: rotation in place or rotation with respect to a point (rp or lp). To find a sub-goal vertex, the robot determines candidate vertices. The selection of a sub-goal vertex from a set of candidate vertices is slightly different for a rotation in place and for a rotation with respect to point rp or lp which is further described in the next subsections.

The definition of candidate vertices is given below. This definition is recursive and makes use of the local frames $\mathscr{F}$, search domains, and θ angles defined in Appendix A (rotations in place) and Appendix B (rotations with respect to an extremal point), which are used to verify traversability toward a vertex u. The definition slightly changes depending on whether $\mathscr{F}$ is over rp or lp, and if the vertex u, which is used to build $\mathscr{F}$, is right or left vertex, producing two different scenarios. Scenario 1 corresponds to $\mathscr{F}$ centered on rp (or lp) and u being right (or left respectively), which appears in both rotations in place and rotations with respect to an extremal point, and scenario 2 corresponds to $\mathscr{F}$ centered on rp (or lp) and u being left (or right respectively), which only appears in rotations with respect to an extremal point. To adequate the correct definition for each scenario, just make the proper substitutions in the definition according to Table 1.

Table 1 Labels for Definition 6

	Scenario 1	Scenario 2
$q_p = rp$ (or $q_p = lp$)	u is right (or u left respectively)	u is left (or u right respectively)
Ξ means	larger	smaller
Υ means	largest	smallest
Used local frame $\mathscr{F}$ built according to	Appendix A or Appendix B	Appendix B

Conditions B (blockage conditions): Consider a local frame $\mathscr{F}$ properly built (see Table 1) over q_p (rp or lp). An opposite type vertex o (left if u is right, or right if u is left) blocks u in $\mathscr{F}$, if

(a) o is within the search domain of u with respect to $\mathscr{F}$
(b) o has a θ_o angle (θ_L if o is left, or θ_R if o is right) that is Ξ than the θ_u angle related to u (θ_R if o is left, or θ_L if o is right), both measured according to $\mathscr{F}$, and
(c) o has a distance d_o^t (d_L^t if o is left, or d_R^t if o is right) smaller than distance d_u^t (d_R^t if o is left, or d_L^t if o is right) related to u.

Definition 6 For a given goal vertex u_{goal}, a candidate vertex c is a reflex vertex such that:

(a) c is an opposite type vertex to u_{goal}, where c fulfills the blockage conditions for u_{goal}, and c has the Υ θ angle properly generating the local frame $\mathscr{F}$ (see Table 1), or
(b) c is an opposite type vertex to another candidate vertex c', where c fulfills the blockage conditions for c', and c has the Υ θ angle properly generating the local frame $\mathscr{F}$ (see Table 1).

The complete FSM is presented in subsection 4.1. Later, we describe the procedures to find a sub-goal vertex and to align the robot to that vertex. The procedures described in subsections 4.2 and 4.4 are part of the FSM presented in subsection 4.1.

4.1 Complete FSM for Optimal Navigation

Fig. 8 shows the FSM M for navigation. For modularity and simplicity M is organized in 3 procedures and 5 states. Notice that the procedures themselves are computed by a set of states. The procedures are ALIGN, RPALING and LPALIGN. These procedures have as a main objective to find a sub-goal vertex. The cases in procedures ALIGN, RPALIGN are described in detail in Appendix C, the procedure LPALIGN is just the symmetric case of RPALIGN.

The Finite State Machine modeling allows one to organize the several cases and to consider symmetry on them.

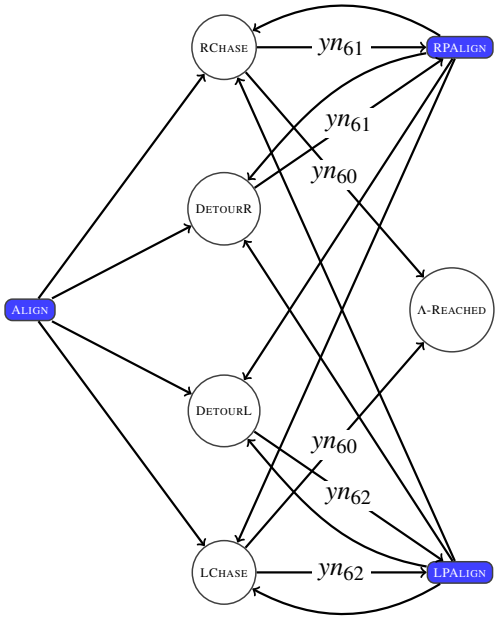


Fig. 8 Complete FSM for Optimal Navigation

Once that the FSM is designed, it can be used to navigate in any simple connected environment by directly mapping observations into controls.

The robot rotates to get aligned with a sub-goal vertex. In ALIGN, the robot rotates in place either in clockwise or counterclockwise sense. In RPALIGN the robot rotates in clockwise sense with respect to point rp and in LPALIGN the robot rotates in counterclockwise sense with respect to point lp.

The states in the machine M are RCHASE, LCHASE, DETOURL, DETOURR and Λ-REACHED. RCHASE makes the robot to move in straight line to a right goal vertex or to the landmark (when the robot moves toward the landmark direction rt is aligned with point rp_Λ). The omnidirectional sensor is placed at rp. LCHASE makes the robot to move in straight line to a left goal vertex or to the landmark (when the robot moves toward the landmark direction lt is aligned with point lp_Λ). The omnidirectional sensor is placed at lp. DETOURR makes the robot to move in straight line to a right sub-goal vertex. The omnidirectional sensor is placed at rp. DETOURL makes the robot to move in straight line to a left sub-goal vertex. The omnidirectional sensor is placed at lp. The state Λ-REACHED indicates that the robot has reached the landmark and the navigation task is over.

The observation vectors for the states in machine M are shown in Table 2 (refer to Fig. 8). For simplifying the presentation of Fig. 8, the observations yielding transitions between procedures and states are not labeled, because there might several ones that generate the transition. However, inside the procedures at the level of states all the transitions are labeled with the corresponding observation.

4.2 Rotation in Place: ALIGN

ALIGN is a procedure that is organized in cases. Each case corresponds to a set of states in the FSM.

In procedure ALIGN a rotation in place is executed to (1) align the robot to a no blocked goal vertex, (2) align it to a landmark, or (3) align it to a sub-goal vertex; an alignment to a sub-goal vertex might require one or more alignments to candidate vertices.

Fig. 9 shows the design of the part of the FSM dealing with the three cases mentioned above, which in turn each of them is instantiated in a left case and a right case. Thus, this part of the FSM is organized in three right cases and three left cases. Note that in the machine the cases are related, that means that while the procedure to find the sub-goal vertex is carried out the current case might change.

The state START1 is the initial state of the whole FSM, it decides whether the robot must get align with a left goal vertex, a right goal vertex, to lp_Λ or to rp_Λ. The observation vectors that make this state to transit to other state are shown in Table 3. In all the further tables 1 denotes true, 0 false and x any value.

In Appendix C three of the six cases are described in detail. The three cases correspond to an alignment to a right goal vertex, to rp_Λ, and to a left candidate vertex. The other three cases are just symmetric.

4.3 Example of the execution of ALIGN

Fig. 10 shows an example of the execution of Case 1-R, Case 3-L (left candidate vertex) and Case 3-R (right candidate vertex) in ALIGN to find a sub-goal vertex. These cases occur sequentially one after the other.

Fig. 10(a) shows the alignment of rt to the goal vertex u_0 corresponding to the execution Case 1-R. First, the goal vertex u_0 is a right vertex, hence the FSM transits to state RV1. Second, the FSM transits to state RCCW1 to align rt to the right goal vertex u_0. Third, once the robot is aligned, the FSM transits to T1B1. This state decides whether or not the right goal vertex u_0 is blocked. Since this goal vertex is blocked the FSM transits to state TCCW2, the first state in Case 3-L.

Fig. 10(b) shows the alignment of lt to the left candidate vertex u_2 corresponding to the execution of Case 3-L (left candidate vertex). First, in state TCCW2 the robot moves the omnidirectional sensor to point lp. Once the omnidirectional sensor is at lp the robot locates the left vertices (in the reference frame $\mathcal{F}$ defined in Appendix A) and selects u_2 as the current candidate vertex. Second, since the lt is not aligned to the left candidate vertex u_2 the FSM transits to RCW1. In this state the robot rotates to align lt to candidate vertex u_2 (see Fig. 10(b)). Third, once the robot is aligned, the FSM transits to T1B2. This state decides whether or

Table 2 Observation vectors for the states in machine M, Fig. 8

	FI	LV	FR	RP	LP	VR	VL	AL	BL	UN	GT	CT	O1	O2
yn_{60}	1	1	1	0	0	x	x	x	x	x	x	x	x	x
yn_{61}	x	x	1	1	0	x	x	0	x	x	x	x	x	x
yn_{62}	x	x	1	0	1	x	x	0	x	x	x	x	x	x

Table 3 Observation vectors for state START1

	FI	LV	FR	RP	LP	VR	VL	AL	BL	UN	GT	CT	O1	O2
yn_0	x	0	x	x	x	x	x	x	x	x	0	x	x	x
yn_1	x	0	x	x	x	x	x	x	x	x	1	x	x	x
yn_2	x	1	x	x	x	x	x	x	x	x	0	x	0	x
yn_3	x	1	x	x	x	x	x	x	x	x	0	x	1	x

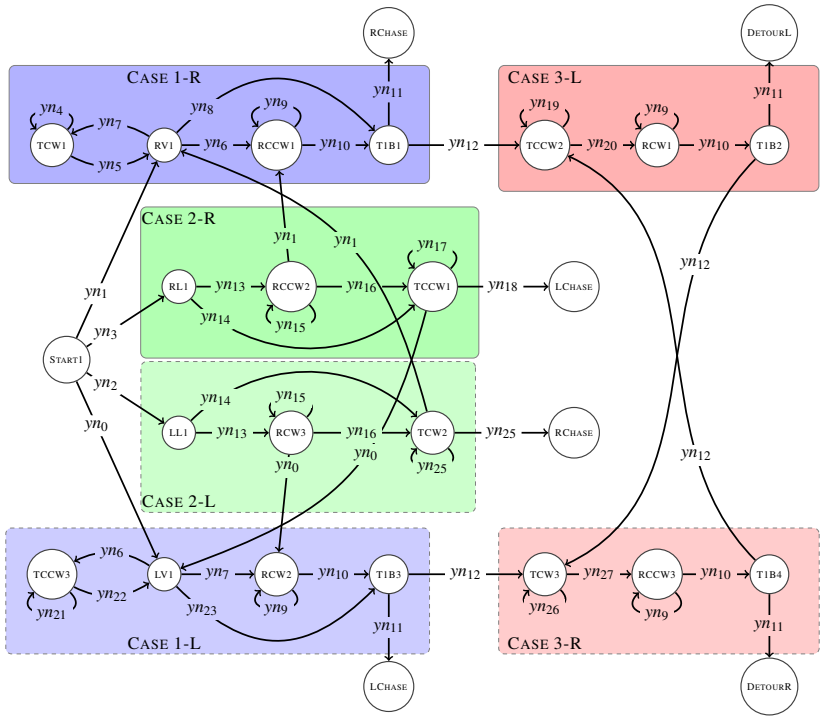


Fig. 9 Procedure ALIGN

not the left candidate vertex u_2 is blocked. Given that the left candidate vertex u_2 is blocked, the FSM transits to state TCW3, the first state in Case 3-R (right candidate vertex).

Fig. 10(c) shows the alignment of rt to the right candidate vertex u_1 corresponding to the execution of Case 3-R (right candidate vertex). In this case, while in TCW3, the FSM selects u_1 as a candidate vertex then the FSM transits to RCCW3 and later to T1B4. State T1B4 decides that the right candidate vertex u_1 is not blocked. Therefore, the right candidate vertex u_1 becomes the right sub-goal vertex.

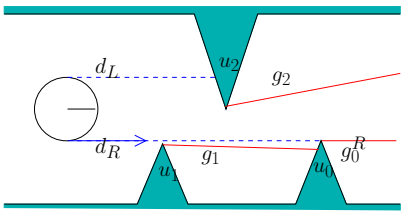
4.4 Rotation with respect to point rp: RPALIGN

In this subsection, we present a procedure that makes the robot rotate w.r.t. a point. Again, the main objective is to find the sub-goal vertex. RPALIGN is also organized in cases. Each case corresponds to a set of states in the FSM.

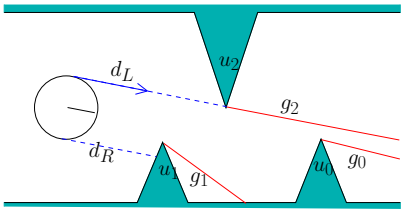
A rotation w.r.t. point rp is executed to (I) align the robot to a right goal vertex, (II) align it to a landmark, (III) align it to a left goal vertex or (IV) align it to a sub-goal vertex.

In Appendix C, we present in detail the cases of the procedure RPALIGN that makes the robot to rotate with respect to point rp. There is an equivalent procedure called LPALIGN which is symmetric to RPALIGN and makes the robot to rotate with respect to point lp.

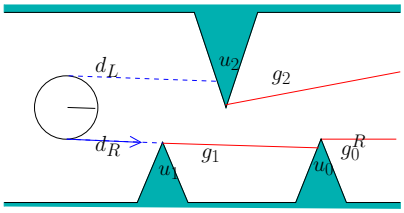
In a rotation in place, the center of the robot does not translate, and the robot can align rt to any right vertex or lt to any left vertex without the risk of loosing optimality in terms of the distance traveled by the center of the robot. In contrast, in a rotation with respect to point rp the center of the robot translates. Consequently, the robot cannot align lt to each left vertex, or rt to each right vertex exhaustively one by one to check whether that vertex is blocked or not. That is because the center of the robot might translate unnecessarily and optimality might be lost. To deal with this problem Al-



(a) Case 1-R



(b) Case 3-L (left candidate vertex)



(c) Case 3-R (right candidate vertex)

Fig. 10 An example of the execution by the robot of Case 1-R, Case 3-L (left candidate vertex) and Case 3-R (right candidate vertex) in ALIGN

gorithm 1 is used. Thus, in Case IV-RP, a sub-goal vertex is found with Algorithm 1. This algorithm guarantees that the robot is able to rotate to get aligned with the sub-goal vertex without losing path optimality.

It might happen that the gap generated by a sub-goal vertex merges with other gap, while the robot rotates to get aligned with a sub-goal vertex. If this occurs then the sub-goal vertex is re-calculated by executing again Algorithm 1.

There is another issue, there are vertices that from the omnidirectional sensor location do not generate a gap when Algorithm 1 is invoked. Such a vertex might block the path toward the goal vertex or the landmark. We call this other issue a hidden vertex (see cases III-RP and IV-RP in Appendix C and Lemma 5 for more details about a hidden vertex). In Lemma 5 is shown that a hidden vertex always will eventually generate a gap. Once a hidden vertex is detected Algorithm 1 is invoked again and the sub-goal vertex is re-calculated.

Fig. 11 shows the design of the part of the FSM dealing with the four cases mentioned above. Again, the cases are related, while the procedure to find the sub-goal vertex is carried out the current case might change. The symmetric cases correspond to LPALIGN.

4.5 Algorithm for finding a sub-goal vertex

This algorithm is only used in RPALIGN Case IV-RP (presented in Appendix C) to find a sub-goal vertex (or in the symmetric left Case IV-LP in LPALING). We call u_p to a left sub-goal vertex and u_n to a right sub-goal vertex. To find vertices u_p or u_n all vertices are located in the local reference frame $\mathscr{F}$ presented in Appendix B. Using the vertices' locations, the distances and angles of alignments ${d_R}^t$, ${d_L}^t$, θ_R and θ_L are computed.

Algorithm 1 uses two orders. The first order is established w.r.t. distances. This order includes distances ${d_R}^t$ for the right vertices and distances ${d_L}^t$ for the left vertices. The order is defined from smaller to larger distances. The second order is an angular order also from smaller to larger; vertices are ordered by angle including both θ_R and θ_L, angle θ_R is used to consider right vertices and θ_L is used to consider left vertices. The vertices that Algorithm 1 must consider to find a sub-goal vertex are located in the search domain presented in Appendix B.

Algorithm 1 Find a sub-goal vertex in RPALIGN or LPALING

1) The algorithm starts from the goal vertex. The goal vertex is declared a candidate vertex.
2) If the candidate vertex is a left vertex, then go to Step 6.
3) Detect left vertices that block the path toward the right candidate vertex u_c^R. To block the path toward u_c^R, left vertices must have an angle θ_L larger than the angle θ_R related to u_c^R, and a distance ${d_L}^t$ smaller than distance ${d_R}^t$ related to u_c^R.
4) If no vertex blocks the path toward the candidate vertex u_c^R, then go to Step 9.
5) Selection of a left candidate vertex u_c^L. The left vertex with largest θ_L, the last in the angular order is selected as u_c^L.
6) Detect right vertices that block the path toward the left candidate vertex u_c^L. To block the path toward u_c^L, right vertices must have an angle θ_R smaller than the angle θ_L related to u_c^L, and a distance ${d_R}^t$ smaller than distance ${d_L}^t$ related to u_c^L.
7) If no vertex blocks the path toward the candidate vertex u_c^L, then go to Step 9.
8) Selection of a right candidate vertex u_c^R. The right vertex with smallest angle θ_R, the first in the angular order is selected as a new candidate vertex u_c^R. Go to Step 3.
9) The right vertex u_c^R is selected as sub-goal vertex u_n or the left vertex u_c^L is selected as sub-goal vertex u_p.

Table 4 shows an example of the execution of Algorithm 1 and the selection of a u_p vertex, this example is shown in Fig. 12. Each row in the botom part of Table 4 is an iteration of Algorithm 1. In Table 4, $\uparrow$ indicates the current candidate vertex to which the blockage conditions are verified, $\times$ indicates the vertices whose distance to each of them is smaller than the distance to the candidate vertex $\uparrow$, $\otimes$ indicates the vertex selected as the next candidate at each iteration, $-$ indicates that the distance to this vertex is smaller than the distance to the candidate vertex $\uparrow$, $+$ indicates that the distance

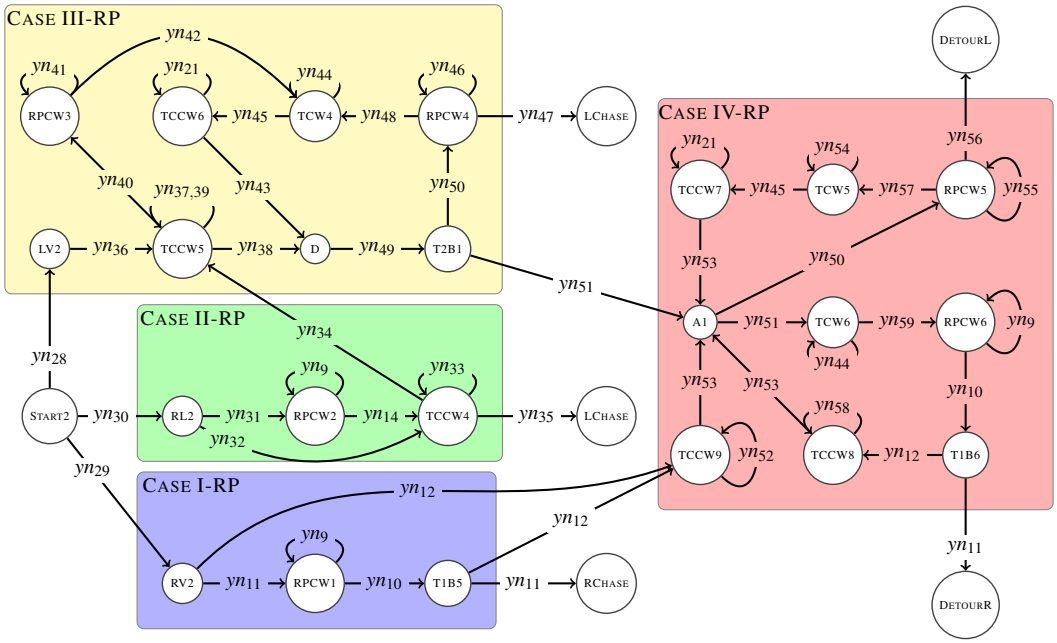


Fig. 11 Procedure RPALIGN

Table 4 Example of orders for selecting a u_p vertex (refer to Fig. 12).

Angular order								Distance order							
Index	1	2	3	4	5	6	7	Index	1	2	3	4	5	6	7
Direction	*rt*	*rt*	*lt*	*rt*	*lt*	*rt*	*lt*	Direction	*rt*	*lt*	*lt*	*rt*	*lt*	*rt*	*rt*
Type	*R*	*R*	*L*	*R*	*L*	*R*	*L*	Type	*R*	*L*	*L*	*R*	*L*	*R*	*R*
Vertex	u_{goal}	u_1	u_5	u_3	u_4	u_6	u_2	Vertex	u_6	u_5	u_4	u_3	u_2	u_1	u_{goal}
$\rightarrow$	$\uparrow$		$\times$		$\times$		$\otimes$			-	-		-		$\uparrow$
$\leftarrow$				$\otimes$		$\times$	$\uparrow$		-			-	$\uparrow$	+	+
$\rightarrow$			$\times$	$\uparrow$	$\otimes$					-	-	$\uparrow$	+		
$\leftarrow$					$\uparrow$	$\times$			-		$\uparrow$	+		+	+

Table 5 Orders for selecting a u_n vertex (refer to Fig. 13).

Angular order					Distance order				
Index	1	2	3	4	Index	1	2	3	4
Direction	*rt*	*lt*	*rt*	*lt*	Direction	*lt*	*rt*	*lt*	*rt*
Type	*R*	*L*	*R*	*L*	Type	*L*	*R*	*L*	*R*
Vertex	u_{goal}	u_3	u_2	u_4	Vertex	u_3	u_2	u_4	u_{goal}
$\rightarrow$	$\uparrow$	$\times$		$\otimes$		-		-	$\uparrow$
$\leftarrow$			$\otimes$	$\uparrow$			-	$\uparrow$	+
$\rightarrow$		$\times$	$\uparrow$			-	$\uparrow$	+	

to this vertex is larger than the distance to the candidate vertex $\uparrow$, $\rightarrow$ indicates that for a left vertex, the vertex that must be selected as the next candidate is the last in the angular order, and $\leftarrow$ indicates that for a right vertex, the vertex that must be selected as the next candidate is the first in the angular order. The algorithm determines that u_4 is a u_p vertex.

Table 5 shows another example of the execution of Algorithm 1 and the selection of a u_n vertex, u_2 is a u_n. The corresponding example is shown in Fig. 13.

4.6 Example of the execution of RPALIGN

Fig. 14 shows an example of the execution by the robot of Case I-RP (right goal vertex) and Case IV-RP (left goal vertex) in RPALIGN. These cases occur sequentially one after the other.

Fig. 14(a) shows the execution of Case I-RP. First, the goal vertex is a right vertex, hence the FSM transits to state RV2. Second, the FSM transits to state RPCW1 to align *rt* to the right goal vertex u_0. Third, once the robot is aligned (see Fig. 14(b)), the FSM transits to T1B5. This state decides whether or not the right goal vertex is blocked (detec-

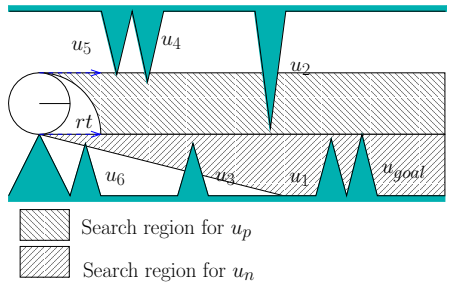


Fig. 12 A u_p vertex, u_4 in this example.

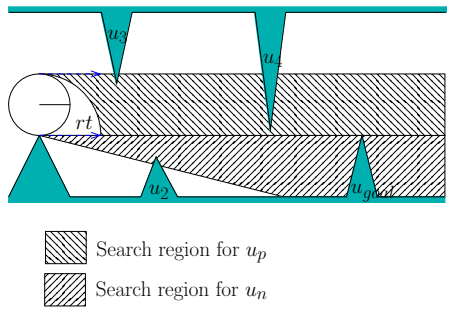


Fig. 13 A u_n vertex, u_2 in this example.

tion of type 1). Since this goal vertex is blocked the FSM transits to state TCCW9, the first state in Case IV-RP.

In state TCCW9, first, the robot locates right vertices in the local reference frame $\mathscr{F}$ presented in Appendix B, second the robot moves the omnidirectional sensor to point lp. Third, once the omnidirectional sensor is at lp, left vertices are located in the reference frame presented in Appendix B. The FSM transits to state A1. In this state the left sub-goal vertex u_2 is selected using Algorithm 1. Once the left sub-goal vertex u_2 is selected the FSM transits to state RPCW5. In this state the robot rotates w.r.t point rp to try to align lt to u_2 (see Fig. 14(b)). During the rotation the gap generated by vertex u_2 merges with the gap generated by vertex u_3. Refer to Fig. 14(c), the omnidirectional sensor crosses the bitangent line between vertex u_2 and u_3. When the merge between the gaps occurs the FSM transits to state TCW5. In state TCW5 the robot stops, and the turret places the omnidirectional sensor at rp. The right vertices are located in the reference frame $\mathscr{F}$ defined in Appendix B. Once the right vertices are located in the local reference frame, the FSM transits to state TCCW7. In state TCCW7 the omnidirectional sensor is placed at lp, and left vertices are located in the reference frame. The FSM transits to state A1. In this state the u_3 is selected as sub-goal vertex using Algorithm 1. Once the left sub-goal vertex is selected the FSM transits to state RPCW5. In this state the robot rotates w.r.t point rp to align lt to u_3 (see Fig. 14(d)). Since u_3 is not blocked no re-invocation of Algorithm 1 is required and u_3 remains as the sub-goal vertex.

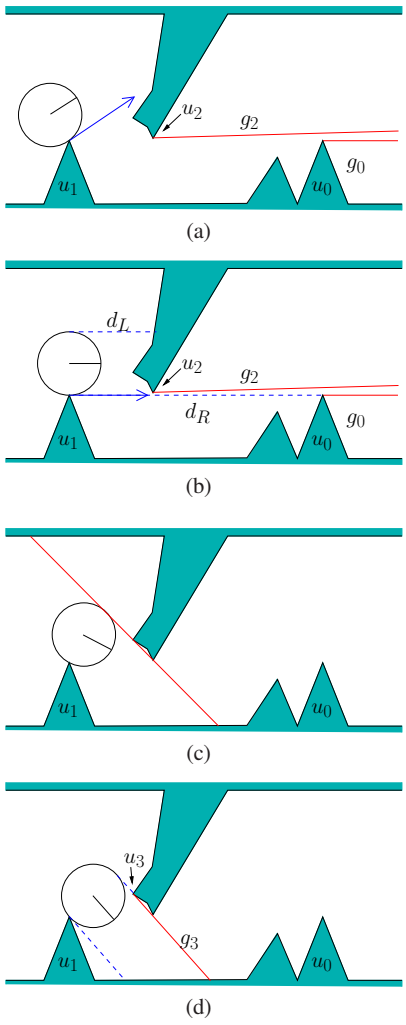


Fig. 14 Case I-PR (right goal vertex u_0) and Case IV-RP (left sub-goal vertices u_2 and u_3) in RPALIGN

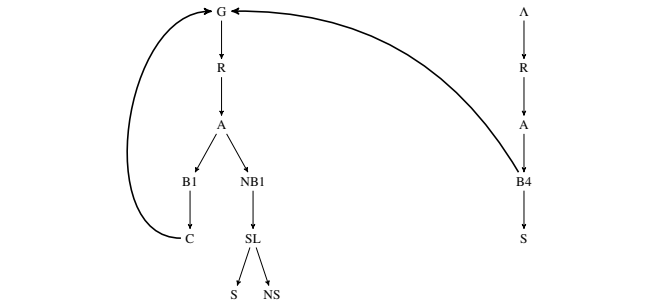
4.7 Considering all possibilities for the FSM

The graphs shown in Figs. 15(a) and 15(b) have as objective to show that the design of the FSM is exhaustive, all possibilities are considered.

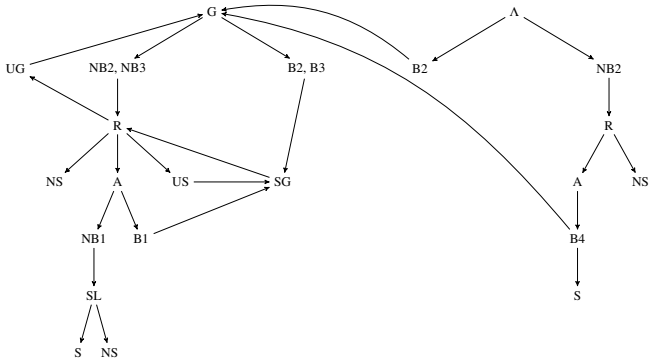
For the case of rotation in place (see Fig 15(a)) the possibilities are as follows. (G) The robot's objective is to get aligned with a gap, or, (Λ) the robot's objective is to get aligned with the landmark. First, consider the possibilities for node (G). Indeed, there is only one possibility as the alignment with the gap is always possible by executing a rotation in place, therefore, (R) the robot rotates in place, and (A) it gets aligned with the gap. Once the robot is aligned with the gap, there are only two possibilities. (B1) There exists a blockage (detection type 1) or not (NB1). If (B1) happens there is only one possibility, namely, (C) the robot finds a candidate vertex. Since the candidate vertex generates a gap the only possibility is to return to node (G) for the

robot to get aligned with this gap. If (NB1) there is no blockage then (SL) the robot moves in straight line. A straight line motion can only yield two possibilities: (S) there is an optimal path toward the gap (the robot touches the vertex with point rp or lp), or, (NS) the robot collides (it touches ∂E with a point different to rp or lp and hence there is no solution path).

Now, consider that (Λ) the robot's objective is to get aligned with the landmark. There is only one possibility, (R) the robot rotates in place, and (A) gets aligned with the landmark. Once the robot is aligned to the landmark, (B4) the path toward the landmark is always assumed as blocked (blockage detection of type 4). From (B4) there are two possibilities: (S) there is an optimal path toward the landmark, or, the path toward the landmark is blocked by a vertex, since this vertex generates a gap, (G) the robot must get aligned with a gap. This analysis lists all possibilities for rotation in place.



(a) Rotation in place



(b) Rotation w.r.t. point rp or lp

Fig. 15 Graphs that present all the possibilities in the FSM

Let us consider a rotation w.r.t point rp or lp. (G) The robot's objective is to get aligned with a gap, or, (Λ) the robot's objective is to get aligned with the landmark. First, consider the possibilities for node (G). There are two possibilities: (NB2,NB3) the path toward the vertex generating the gap is not blocked, or, (B2,B3) it is blocked (blockage detection of type 2 or type 3). If (B2,B3) occurs there is only one possibility, (SG) a sub-goal vertex is found using Algorithm 1. Once the sub-goal vertex has been found, (R) the robot rotates. If (NB2,NB3) occurs the only possibility is (R). If (R) happens there are four possibilities: (NS) the robot collides (the robot touches ∂E with a point different to rp or lp), (UG) the gap generated by the goal vertex merges with other gap, (US) the gap generated by the sub-goal vertex merges with other gap, or (A) the robot is able to achieve the alignment with the vertex generating the gap. If (UG) then there is a new goal vertex and it is needed to verify a blockage for this vertex; the process starts again in (G). If (US) occurs then the sub-goal must be re-calculated using Algorithm 1 in (SG). If (A), once the alignment has been done, there are two possibilities: (NB1) the path toward the vertex is not blocked, or, (B1) it is blocked (blockage detection of type 1). If (B1) then a sub-goal vertex is calculated using Algorithm 1 in (SG). If (NB1), no blockage, then (SL) the robot moves in straight line. A straight line motion can only yield two possibilities: (S) there is a an optimal path toward the gap (the robot touches the vertex with point rp or lp), or, (NS) the robot collides (it touches ∂E with a point different to rp or lp).

Finally, consider that (Λ) the robot's objective is to get aligned with the landmark. There is only one possibility, (B2) the path toward the landmark is blocked (blockage detection of type 2), or, (NB2) the path is not blocked. If (B2) then the path toward the landmark is blocked by a vertex, since this vertex generates a gap, (G) the robot must get aligned with such gap. If (NB2) then (R) robot rotates. Once the robot rotates there are two possibilities: (NS) robot collides, there is no solution, or, (A) the robot is able to achieve the alignment with the landmark. If (A) then, once the robot is aligned to the landmark, (B4) the path toward the landmark is always assumed as blocked (blockage detection of type 4). From (B4) there are two possibilities: (S) there is an optimal path toward the landmark, or, the path toward the landmark is blocked by a vertex, since this vertex generates a gap, (G) the robot must get aligned with a gap. This analysis lists all possibilities for rotation with respect to point rp or lp.

5 The feedback motion strategy for navigation

In this section two motion policies are syntethized from the FSM, which maps observations to actions are presented. In the first one, the motion of a turret that changes the omnidirectional sensor position is included, and in the second one it is implicit. The motion policy is based on the paradigm of avoiding the state estimation to carry out two consecutive mappings: $y \rightarrow x \rightarrow u$, that is from observation y to state x and then to control u, but instead of that there is a direct mapping $y \rightarrow u$. The observation vector yn for navigation including the turret's motion has 14 binary sensor observations (see Section 3 for the description of each binary obser-

vation). It is interesting to note that the observation vector ym for the feedback motion strategy without turret's motion is a subset of yn, that is $ym \subset yn$. This means that some of the observation elements of yn are not relevant for the feedback motion strategy that makes implicit the turret's motion and they can be any value, consequently they are not included.

5.1 Feedback motion strategy including turret's motion

In this feedback motion strategy the turret's motion is included. Whenever the robot translates the turret is static and whenever the turret moves the robot is motionless. Depending on the observation, one of the five different motion primitives will be executed for the robot and two for the turret: (1) straight line motion; (2) clockwise rotation in place; (3) counterclockwise rotation in place; (4) clockwise rotation with respect to point rp and (5) counterclockwise rotation with respect to point lp. In the next two motion primitives the robot is motionless and the turret moves: (6) the robot moves the turret on its boundary in clockwise sense and (7) the robot moves the turret on its boundary in counterclockwise sense.

The feedback motion strategy can be established by: $\gamma : \{0,1\}^{14} \rightarrow \{-1,0,1\}^3$. The feedback motion strategy is given by $\gamma(yn_i) = (w_r, w_l, v_t)$, where w_r and w_l are the angular velocities of the right and left wheels, and v_t is the velocity displacement of the turret on the robot boundary. The set of all 16384 possible observation vectors can be grouped by letting x denote "any value" . This can be done using standard boolean algebra and reduction techniques, which might be implemented using software. Doing so we obtain:

$yn_{63} = (0,x,x,x,x,x,1,1,0,0,x,x,x,0)$,
$yn_{64} = (0,x,x,x,x,x,x,0,1,0,0,x,x,x,0)$,
$yn_{65} = (0,x,0,x,x,0,0,0,0,x,0,0,0,0,0)$,
$yn_{66} = (0,x,0,x,x,x,1,1,0,0,0,x,0,0,0)$,
$yn_{67} = (0,0,0,x,x,0,0,0,0,0,0,1,0,1,0)$,
$yn_{68} = (0,x,0,x,x,x,0,0,0,1,0,x,0,1,0)$,
$yn_{69} = (0,x,0,x,x,x,1,1,0,0,0,x,1,1,0)$,
$yn_{70} = (0,x,1,1,0,1,1,0,0,0,x,0,0,0,0)$,
$yn_{71} = (0,x,1,1,0,1,1,0,0,0,x,1,1,0,0)$,
$yn_{72} = (0,0,1,1,0,x,0,0,0,0,1,0,1,0,0)$,
$yn_{73} = (0,1,1,1,0,1,0,0,0,0,0,0,1,0,0)$,
$yn_{74} = (0,0,1,1,0,x,0,0,0,1,1,0,0,0,0)$,
$yn_{75} = (0,0,1,1,0,0,0,0,0,0,0,0,0,0,0)$,
$yn_{76} = (0,x,1,0,1,1,1,0,0,0,x,1,1,0,0)$,
$yn_{77} = (0,x,1,0,1,1,1,0,0,0,x,0,0,0,0)$,
$yn_{78} = (0,x,1,0,1,0,x,0,0,0,0,0,0,0,0)$,
$yn_{79} = (0,0,1,0,1,0,x,0,0,1,0,0,1,0,0)$,
$yn_{80} = (0,0,1,0,1,0,0,0,0,0,0,1,0,1,0)$,
$yn_{81} = (x,x,x,x,x,x,x,x,x,x,x,0,1)$,
$yn_{82} = (0,x,x,x,x,x,x,1,1,0,x,x,0,0)$,
$yn_{83} = (0,0,0,x,x,0,0,0,0,0,1,0,0,0)$,
$yn_{84} = (0,0,1,1,0,0,0,0,0,0,1,0,0,0,0)$,
$yn_{85} = (0,x,1,1,0,1,1,0,0,0,x,1,0,0)$,
$yn_{86} = (0,x,1,1,0,1,1,0,0,1,x,0,0,0)$,
$yn_{87} = (0,0,1,0,1,0,1,0,0,x,1,0,0,0)$,
$yn_{88} = (0,x,1,0,1,0,1,0,0,1,x,0,0,0)$,
$yn_{89} = (0,x,1,0,1,0,x,0,1,0,0,0,0,0)$,
$yn_{90} = (x,x,x,x,x,x,x,x,x,x,x,x,1,1)$,
$yn_{91} = (0,x,x,x,x,x,x,x,1,1,0,x,x,1,0)$,
$yn_{92} = (0,0,x,x,x,x,0,0,0,0,0,0,1,0)$,
$yn_{93} = (0,x,1,1,0,1,0,0,0,1,x,0,1,0)$,
$yn_{94} = (0,x,1,1,0,x,0,0,1,0,x,0,1,0)$,
$yn_{95} = (0,0,1,0,1,x,x,0,0,1,1,x,1,0)$,
$yn_{96} = (0,x,1,0,1,1,1,0,0,0,0,x,0,1,0)$,
$yn_{97} = (0,x,1,0,1,1,1,0,0,1,0,1,1,0)$.

The strategy γ can be encoded as

1 $\gamma(yn_{63} \vee yn_{64}) = (1,1,0)$;
2 $\gamma(yn_{65} \vee yn_{66}) = (-1,1,0)$;
3 $\gamma(yn_{67} \vee yn_{68} \vee yn_{69}) = (1,-1,0)$;
4 $\gamma(yn_{70} \vee yn_{71} \vee yn_{72} \vee yn_{73} \vee yn_{74} \vee yn_{75}) = (0,1,0)$;
5 $\gamma(yn_{76} \vee yn_{77} \vee yn_{78} \vee yn_{79} \vee yn_{80}) = (1,0,0)$;
6 $\gamma(yn_{81} \vee yn_{82} \vee yn_{83} \vee yn_{84} \vee yn_{85} \vee yn_{86} \vee yn_{87} \vee yn_{88} \vee yn_{89}) = (0,0,1)$;
7 $\gamma(yn_{90} \vee yn_{91} \vee yn_{92} \vee yn_{93} \vee yn_{94} \vee yn_{95} \vee yn_{96} \vee yn_{97}) = (0,0,-1)$.

in which $\vee$ means "or".

5.2 Feedback motion strategy with turret motion made implicit

Only six binary sensor observations affect the control of the wheels motors. The used observation vector is $ym_i =$ (FR, RP, LP, AL, BL, O1). Refer to Section 3 for the meaning of each element of the observation vector. Depending on the observation, one of the five different motion primitives will be executed: (1) straight line motion; (2) clockwise rotation in place; (3) counterclockwise rotation in place; (4) clockwise rotation with respect to point rp and (5) counterclockwise rotation with respect to point lp. Recall that

the angular velocities of the differential-drive wheels yield one of these motion primitives. Hence, the feedback motion strategy can be established by: $\gamma : \{0,1\}^6 \rightarrow \{-1,0,1\}^2$ to obtain $\gamma(ym_i) = (w_r, w_l)$. The set of all 64 possible observation vectors can be grouped by letting x denote "any value" to obtain: $ym_1 = (x,x,x,1,0,x)$, $ym_2 = (0,x,x,0,x,0)$, $ym_3 = (0,x,x,0,x,1)$, $ym_4 = (1,0,1,0,x,x)$, $ym_5 = (1,1,0,0,x,x)$.

The strategy γ can be encoded as

$(1)\gamma(ym_1) = (1,1)$; $(4)\gamma(ym_2) = (-1,1)$;
$(2)\gamma(ym_3) = (1,-1)$; $(5)\gamma(ym_4) = (0,1)$;
$(3)\gamma(ym_5) = (1,0)$.

6 Proof of optimal navigation

In this section, we prove that the center of the disc nonholonomic robot travels the shortest distance to reach the landmark. Our methodology is not based on numerical optimization techniques but on a geometrical and topological reasoning. To our knowledge this is the first time that the shortest path for a DDR (system) is found in the presence of obstacles (environment constraints) without knowing the complete geometric representation of the environment. Note that our result is not an approximation but the exact solution. The robot path is found in the continuous. However, recall that the robot follows only three motions primitives (straight line motion, rotation in place and rotation with respect to point rp or lp), the finite state machine triggers one of these motion primitives according to critical changes in the sensor readings. Hence, we think that the approach is related to hybrid control in which continuous and discrete modeling is combined.

In this section, we also stress the fact that minimizing the Euclidean distance traveled by the center of the robot, gives as a consequence the existence of a geometric solution for the navigation problem of reaching a landmark. Hence if the proposed motion strategy does not find a collision free path to the landmark then such path does not exist.

The methodology presented here might be used to solve other related problems, e.g., finding the shortest path for other nonholonomic underactuated systems.

Subsection 6.1 considers the case when the GNT encoded paths are non-blocked and Subsection 6.2 considers the case of blocked paths.

6.1 Non-blocked GNT-encoded paths

The shortest path to Λ is encoded as a sequence of gaps in the GNT. Let $U = (u_n, u_{n-1}, ..., u_0)$ be the sequence of connected intervals $u_i \subset \partial E$ that the robot contacts when the gap sensor (fixed to the robot boundary) moves from its initial position to its final position in Λ. In this section we establish that the robot executes an Euclidean, distance-optimal path in the absence of blockages, namely, no detours between intervals u_{i+1} and u_i are done.

Let $H = (g_n, g_{n-1}, ..., g_0)$ denote the corresponding sequence of gaps that are chased, in which $g_i \in H$ is the gap that is being chased on the path to u_i or while traversing u_i.

Now consider the problem in terms of the configuration space of the robot. The obstacle region in the configuration space is obtained by growing the environment obstacles by the robot's radius. Let C denote the projection of the obstacle region into the plane, thereby ignoring rotation. Let $V = (v_n, v_{n-1}, ..., v_0)$ be the sequence of intervals $v_i \subset \partial C$ obtained by transforming the interval sequence U from ∂E to ∂C, element by element. The following lemma uses the definition of a generalized bitangent from [39].

Lemma 2 *Chasing the sequence H of gaps produces the shortest path if and only if: 1) there is a straight collision-free path from the center of the robot to v_n, 2) there is a (generalized) bitangent line between v_{i+1} and v_i, 3) there is a straight collision-free path from v_0 to the landmark center, and 4) C is connected.*

Proof: We first proof the left to right direction of the if and only if statement. The gap sensor is located over points rp or lp. Chasing the sequence H of gaps makes the robot to touch the intervals $u_i \subset \partial E$ with points rp or lp, as a consequence, the robot's center visits intervals $v_i \subset \partial C$. If chasing the gaps in H produces the shortest path and the robot's center visits each interval $v_i \in V$, then a solution global path must exist (C is connected), and there must exist local straight collision free paths connecting the initial location of the robot's center with the sequence V, connecting each pair v_{i+1} and v_i in V, and the sequence V with Λ, which in conjunction are the four conditions in the Lemma statement.

To proof the other direction of the statement, we proceed by contradiction. Let's suppose that the three first conditions are fulfilled and that there exists a solution path, hence, the fourth condition is also fulfilled. Now, let's suppose that the shortest path is not the one depicted by the three first conditions. If such shortest path exists, then there must be a blockage in the straight paths of either the initial position of the robot's center and V, between intervals v_{i+1} and v_i in V, or between V and Λ, therefore, at least one of the three conditions is not fulfilled, which is a contradiction. Furthermore, as the path depicted by the three conditions visits the intervals $v_i \in V$, then it is traversable chasing the gaps in H. The result follows. ■

6.2 Blocked GNT-encoded paths

We now consider the cases for which either of the first three conditions of Lemma 2 is violated, meaning that the robot

would become blocked when applying the GNT in the usual way. For these cases, various forms of "detours" are required. The GNT-encoded path is based on the bitangent lines between intervals in ∂E. However, in C, some bitangent lines disappear. Bitangent lines in the workspace that remain in C are displaced by a distance r or are rotated by some fixed angle.

The GNT-encoded path cannot be executed by the robot when there is a blockage while chasing $g_i \in H$ (or Λ). If this happens it means that: 1) the robot is in a zone in which it cannot detect the crossing of a bitangent line in C, that means that because of the width of the disc robot, there is a bitangent line between the sensor location and the vertex u_n, but there is not a bitangent line between the center of the robot and v_n, 2) there is no bitangent line between v_{i+1} and v_i in C, 3) there is no straight line path to chase Λ when the robot sees Λ, or 4) C is disconnected. These are the conditions of Lemma 2. In Theorem 2, we state that our navigation strategy is able to deal with the first three cases presented above. Therefore, it is always possible to detect an optimal collision free path if one exists. The disconnection of C, the case in which there is no path to Λ will produce a robot collision when commanded by the FSM M. In Theorem 3, we prove that if our motion strategy does not find a collision free path to reach the landmark, then there is no path to reach it.

If the robot detects a blocked path, then it performs a detour to avoid the obstacles that block the GNT-encoded path. We cannot re-plan the entire path to Λ because the path depends on the gap $g_i \in H$ (or Λ) that is in the gap sensor field of view. For this reason the detour to avoid obstacles is done when the robot detects a blocked path while chasing g_i or Λ.

In the remaining of this section, without loss of generality, when a rotation w.r.t. a point is referred it is a rotation w.r.t. rp. To prove some lemmas and theorems below, we introduce the following concepts. Let us consider a given candidate vertex u_c. If u_c is blocked by any other vertex then the candidate vertex u_j that deforms the most the straight line robot path toward u_c is said to *lie on the boundary of the restrictions*. For a $u_c = u_c^R$, the u_j that lies on the boundary of the restriction corresponds to the u_j^L that blocks the path toward u_c^R, and has the largest θ_L angle, the last in the angular order, see Fig. 16. Symmetrically, for a $u_c = u_c^L$, the u_j that lies on the boundary of the restriction corresponds to the u_j^R that blocks the path toward u_c^L, and has the smallest θ_R angle, the first in the angular order.

Lemma 3 *Algorithm 1 finds a sub-goal vertex among the vertices generating gaps at the robot configuration when the algorithm is invoked. The candidates vertices, including the sub-goal vertex, lie on the boundary of the restrictions of semi-collision free motion imposed by the vertices blocking the path toward the goal vertex.*

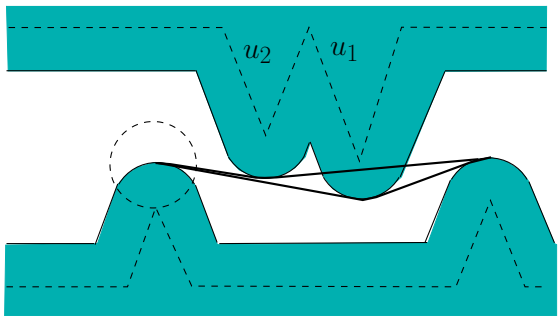


Fig. 16 vertex u_1 lies on the boundary of the restriction

Proof: By Construction Algorithm 1 detects left vertices that block the path toward a right candidate vertex or right vertices that block the path toward a left candidate vertex. For a blocked right candidate vertex, Algorithm 1 selects as a new candidate vertex the left vertex with largest θ_L, the last in the angular order, which lies on the bordary of the restriction. For a blocked left candidate vertex, Algorithm 1 selects as a new candidate vertex the right vertex with smallest θ_L, the first in the angular order, which lies on the bordary of the restriction. This procedure repeats until the candidate vertex is not blocked selecting this last one as the sub-goal vertex. Hence, the result follows. ■

A path toward a landmark might be blocked by a hidden vertex. Whether or not a vertex is hidden depends on the relative positions between the omnidirectional sensor and the vertex. A hidden vertex is important because if it blocks the robot it must be considered to find the optimal path, which is done using Algorithm 1, and hence if there was a hidden vertex when Algorithm 1 was invoked then this algorithm must be invoked again when the vertex generates a gap.

Exhaustively, there are four possible cases analyzing if (EG) the hidden vertex eventually generates a gap or (NG) it never generates a gap, and if (S) there exists a collision free path toward the landmark or not (NS), while the robot moves commanded by the FSM. Below we show, through the four cases, that either, if there is a solution (an optimal path) then the FSM finds it, or if there is no solution then the robot collides.

The case 1 is the following: 1) in the path that the robot follows the hidden vertex eventually generates a gap (EG) and 2) there is not an optimal path (NS). See Fig. 17(a) and element (EG,NS) in Table 6. In this case the FSM returns a sub-goal vertex but robot collides since there is no solution path.

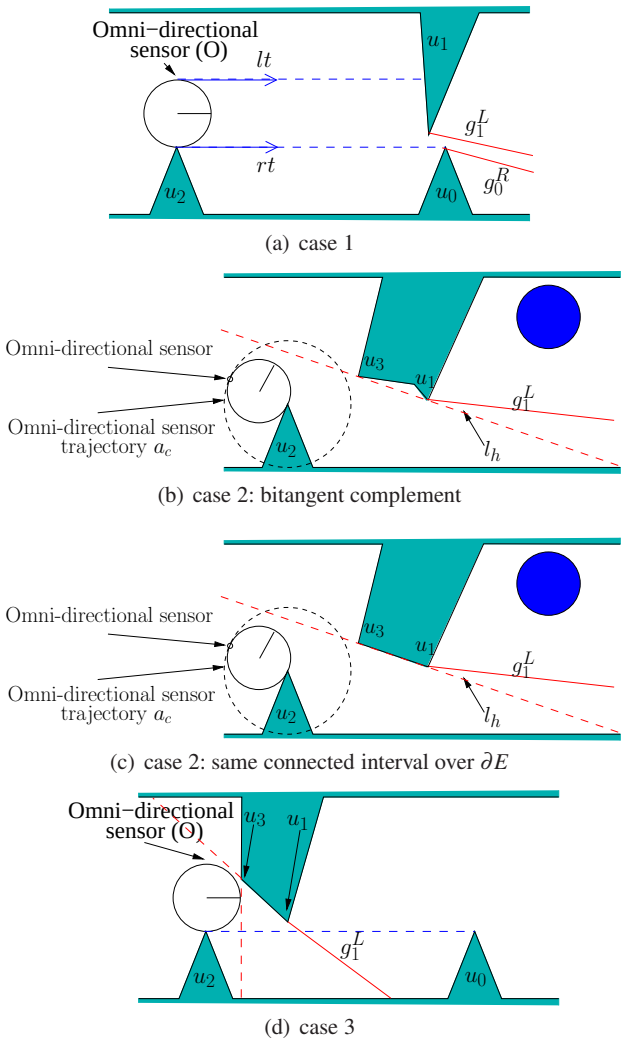
The case 2 is: 1) the hidden vertex eventually generates a gap (EG) while the robot moves in the optimal path and 2) there is an optimal path (S). See Figs. 17(b) and 17(c), and element (EG, S) in Table 6. Lemma 4 proves that if there exists a solution path then each hidden vertex always eventually generates a gap while the robot is commanded by the FSM.

Table 6 4 cases relating the existence of the optimal path and a hidden vertex

	NS	S
EG	Yes	Yes
NG	Yes	No

The case 3 is: 1) the hidden vertex never generates a gap while the robot moves commanded by the FSM (NG) and 2) there is not an optimal path (NS). See Fig. 17(d) and element (NG, NS) in Table 6. In this case the FSM will never take into account the hidden vertex but the no solution will be detected because the robot collides.

The case 4 is: 1) the hidden vertex never generates a gap while the robot moves commanded by the FSM (NG) and 2) there is an optimal path (S). Element (NG, S) in Table 6. Note that given that there exists a solution, this case is the complement of case 2 and by Lemma 4 case 2 always occurs, hence case 4 *does not exist*.



(a) case 1

(b) case 2: bitangent complement

(c) case 2: same connected interval over ∂E

(d) case 3

Fig. 17 Cases relating the existence of the optimal path and a hidden vertex

Lemma 4 *Let us assume that a sub-goal vertex u_s is obtained by Algorithm 1. If there is a collision free path to the landmark Λ, then a hidden vertex u_h that blocks to u_s must generate a gap while the robot rotates with respect to rp to align lt to a left sub-goal vertex u_s.*

Proof: Refer to Figs. 17(b) and 17(c). The robot rotates w.r.t. point rp to align lt to a left u_s. While the robot rotates the omnidirectional sensor is moving in an arc of circle path a_c. Consider a line l_h that contains u_s and u_h. For blocking the path toward u_s by u_h, the line l_h must intersect a_c, otherwise the robot is always able to align lt to u_s. Furthermore the line l_h must be crossed by the omnidirectional sensor in order to align lt to the u_s vertex, as lt is tangent to the robot boundary, l_h contains u_s and l_h is a secant line to the circle a_c. For this robot rotation, two cases exist according to the gap events triggered by crossing l_h. The first one occurs when vertex u_h generates a gap and this gap merges with the gap generated by the u_s, see Fig. 17(b). Since two gaps merged there must exist a bitangent complement between u_h and u_s (as defined in [39]) over l_h. Originally the vertex u_h does not generate a gap, but due to the omnidirectional sensor motion the bitangent complement is crossed (the gaps generated by u_s and u_h merge), hence a gap originated by vertex u_h must appear. The second case occurs when the gap generated by u_s changes of vertex generating it, and the gap is generated by vertex u_h, see Fig. 17(c). In this second case the segment joining vertices u_s and u_h lies over line l_h. Since the line segment joining u_h and u_s is the same connected interval over ∂E, then the gap generated by u_s must change the vertex generating it. The gap must be generated by vertex u_h at the moment that line l_h is crossed by the gap sensor while the robot is rotating. Hence the vertex u_h must eventually generate a gap. The result follows. ■

There is an equivalent lemma for the case when the robot rotates w.r.t. point lp to align rt to a right u_s.

Lemma 5 *Consider a procedure ζ, which invokes Algorithm 1 at each time that a hidden vertex is detected. Procedure ζ terminates and finds the sub-goal vertex. The path toward the sub-goal vertex is optimal in the sense of Euclidean distance traveled by the center of the robot.*

Proof: By Lemma 3 Algorithm 1 delivers a sub-goal vertex. By Lemma 4 a hidden vertex is always detected. If the sub-goal vertex is not blocked by a hidden vertex then the sub-goal vertex is not re-calculated. Otherwise, Algorithm 1 is invoked again. Therefore, procedure ζ finds the sub-goal vertex u_s and as there is a finite number of vertices the procedure ζ terminates. Furthermore, by Lemma 3 a sub-goal vertex u_s lies on the boundary of the restriction and it is not blocked then the path toward the sub-goal vertex u_s is optimal. ■

Now, we present the theorem that ensure globally optimal navigation when using M.

Theorem 2 *The path that the robot center follows when commanded by the automaton M, using the information encoded in the GNT and making detours when the straight line path to chase $g_i \in H$ is blocked or when the straight line path to chase Λ is blocked, is globally optimal in the sense of Euclidean distance.*

Proof: Let assume that there is a collision free path toward the landmark, that is, C is connected. The GNT-encoded path is the shortest path for a point in the workspace and it is in the same homotopy class that the shortest path in C because E and C are simply connected. The sequence of connected intervals V in ∂C that the robot traverses is only changed when either of the first 3 conditions of Lemma 2 are not satisfied. However, even if any of the first 3 conditions of Lemma 2 are not satisfied then the original sequence of intervals in V does not change the order, as the homotopy class of the shortest path in C remains the same, but new sub-goals vertices are added corresponding to new intervals of ∂C. These sub-goals produce detours between original consecutive intervals of v_i and v_{i-1}, and they are locally optimal (as proved in Lemma 5) as all of them belong to the boundary of the restriction (as proved in Lemma 3). Hence, the resulting global path is optimal. ■

Theorem 3 is one of our main results. In short it indicates that if our motion strategy does not find a solution, then there is no solution. This result also increases the interest of the proposed optimization criterion, i.e. to move the center of the circular robot as less as possible, since from a point of view of the geometric existence of a solution, it establishes a solution for the worst scenario -the most restricted polygonal environment, that is, the environment with the most narrow passage, such that, this passage is wider than the robot's diameter. So that, the problem addressed in this paper can be considered a *game* [23] against the polygonal environment.

Theorem 3 *If robot is commanded by the automaton M yielding the optimal path in the sense of Euclidean distance and the robot touches ∂E with a point different to lp or rp (robot collides) then there does not exist a path to reach the landmark.*

Proof: By Lemma 5 sub-goal vertices are found. The restriction of collision free path is imposed by all vertices blocking the straight line path toward the goal vertex. The sub-goal vertices are located on the boundary of the restriction of semi-collision free motion and must be touched by point rp or lp. Hence, if robot collides (that is the robot touches ∂E with a point different to lp or rp) then there is no solution. ■

7 Implementation

The whole method has been implemented and simulations' results are included. All our simulation experiments were run on a PC with a quad-core processor, equipped with 4 GB of RAM, running Linux, and were programmed in C++ using the computational geometry library LEDA. Our software implementation exactly emulates the FSM.

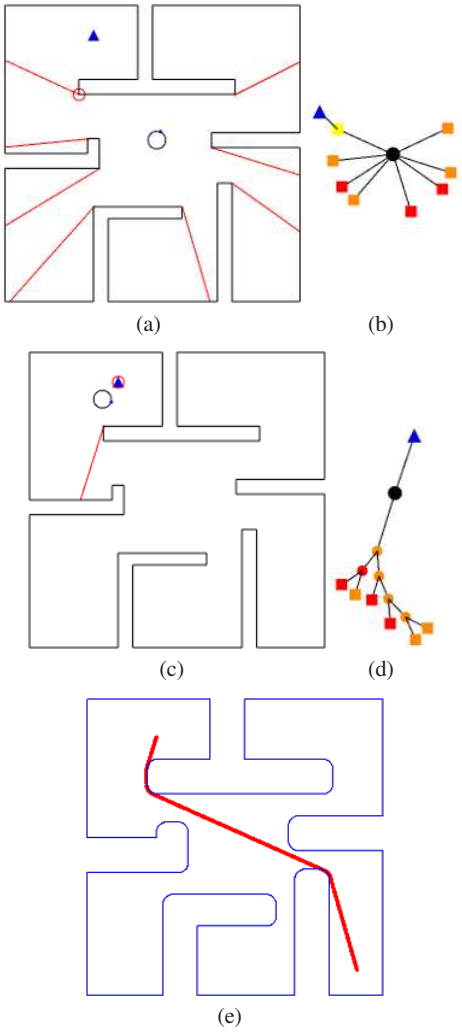


Fig. 18 A simulation of optimal gap navigation for a disc robot. (e) shows the path in the projected configuration space that the robot traverses to go to the landmark.

The planning time for obtaining each one of the five simulation experiment presented in this section, is always less than a second. These planning times do not include the execution time of the navigation itself.

We have implemented the method with several objectives: (1) to illustrate the execution of the FSM, (2) to graphically show the path in C, and (3) to display the evolution of the GNT as the navigation task is executed.

Fig. 18 shows a simulation of the optimal gap navigation for a disc robot. In this first example, the vertices that the robot visits in its shortest path toward the landmark are the same that the ones visited by a point robot. Figs. 18(a) and 18(c) show the robot at different times while following

a sequence of gaps to reach the landmark. To the right of each figure is shown the complete GNT with the representation used in [39]. The landmark to be chased is marked as a blue triangle in the workspace and in the GNT as a leaf triangle node. Fig. 18(a) shows the robot chasing the gap in the path toward the landmark. Finally, Fig. 18(c) shows the robot chasing the landmark. Fig. 18(e) shows the shortest path in the configuration space that the robot traverses to navigate to the landmark. This path was computed based on the information obtained by the robot sensors and using the automaton M from Section 4.

Fig. 19 shows a second navigation example to reach a landmark for the case where blockages of the encoded path in the GNT occur. Fig. 19(a) shows the robot aligned to a right vertex. Fig. 19(b) shows the case in which the first sub-goal vertex when the robot is touching ∂E with rp is a left sub-goal vertex -u_p-, vertex u_4 in the figure. Algorithm 1 is used to find this sub-goal vertex u_p. Figs. 19(c) and 19(d) show some snapshots of the navigation task. Fig. 19(e) shows the shortest distance path in the projected configuration space C that the center of the robot travels to reach the landmark. Note that the vertices u_5 and u_6 are not touched by the robot. Fig. 19(f) shows the GNT when the robot chases the landmark.

Fig. 20 shows an example in which in the shortest path to reach the landmark the robot moves in contact with segments of the polygonal environment (in contact with ∂E). Figs. 20(b) and 20(c) show this case. Fig. 20(e) shows the shortest distance path in the projected configuration space C that the center of the robot travels to reach the landmark. Fig. 20(f) shows the GNT when the robot chases the landmark.

Fig. 21 shows an example of a hidden vertex (vertex u_1). Fig. 21(a) shows the beginning of the robot's path. Fig. 21(b) shows that at the moment when the robot is touching vertex u_3 with rp having the omnidirectional sensor placed at rp, the vertex u_1 does not generate a gap (it is a hidden vertex). To find a sub-goal vertex Algorithm 1 is executed without considering vertex u_1. Algorithm 1 determines that the goal vertex u_{goal} is not blocked. When the robot rotates to align lt to the left goal gap, it merges with the gap generated by u_1, see Fig. 21(c). When the gaps generated by vertices u_{goal} and u_1 merges Algorithm 1 is invoked again. The algorithm determines that u_2 is the sub-goal vertex. Since u_2 is not blocked by a hidden vertex Algorithm 1 is not invoked again.

Fig. 21(d) show rt aligned with u_2. Fig. 21(e) shows the shortest distance path in the configuration space C. Fig. 21(f) shows the GNT when the robot chases the landmark.

In the multimedia material of the paper *3 videos are available*, showing from the second to the fourth simulation experiments in the paper.

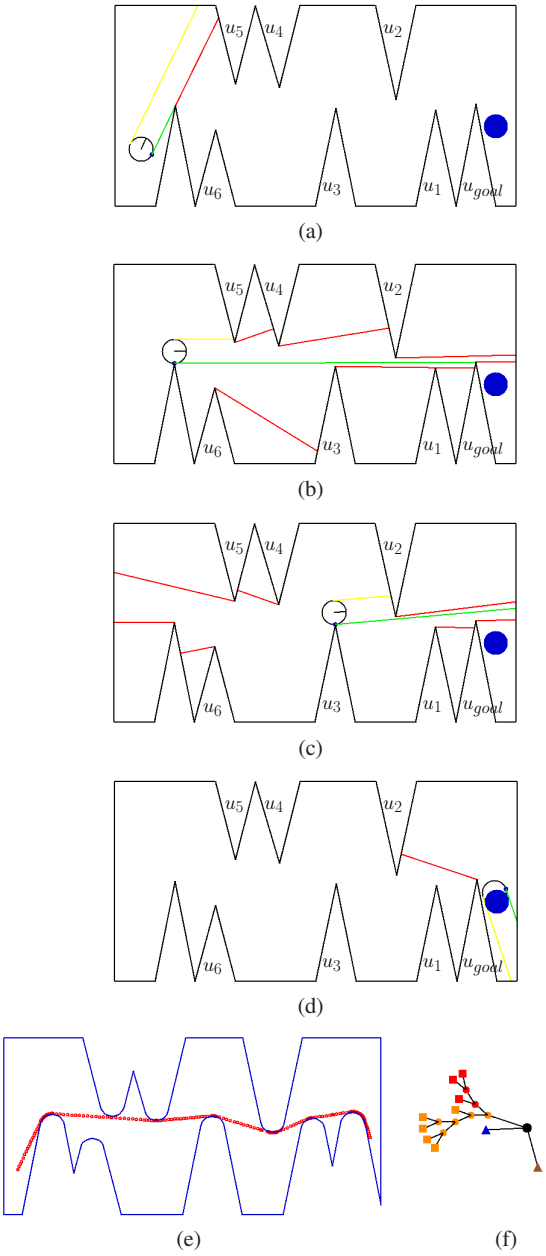


Fig. 19 Example in which the first sub-goal when the robot is touching ∂E with rp is an u_p vertex

8 Conclusions

In this paper we have presented an approach for distance optimal navigation of a disc-shaped differential-drive robot. The robot is equipped with sensors and it does not need to build precise geometric maps or localize itself in a global Euclidean frame. Critical information from the workspace is obtained from the robot's sensors, to infer the optimal robot paths in the configuration space. To solve this problem we developed a motion strategy based on sensor feedback and then proved that the motion strategy yields globally optimal

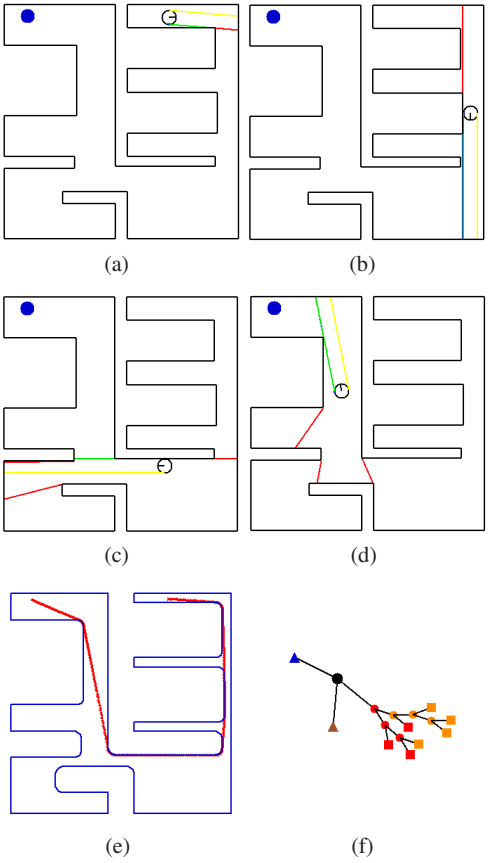


Fig. 20 The robot moves in contact with segments of the polygonal environment

motions in the sense of Euclidean distance by characterizing all possible trajectories in terms of sequences of states visited in a finite state machine.

To our knowledge this is the fist time that the shortest path for a DDR (underactuated system) is found in the presence of obstacles (environment constraints) without knowing the complete geometric representation of the environment. Note that our result is not an approximation but the exact solution. The methodology presented here might be used to solve other related problems, e.g., finding the shortest path for other underactuated systems.

This work has shown that a crucial requirement for planning navigation strategies is to define the required information to accomplish the task. Once that this information is established, it is possible to describe the motion strategy as an automaton. This automaton is built prior to navigation and determines an action to be taken by the robot for any possible observation. Based on this automaton it is possible to obtain a motion policy that maps directly observation to action without estimating the state of the robot in the configuration space.

It is also interesting to note that the criterion of minimizing the Euclidean distance traveled by the center of the

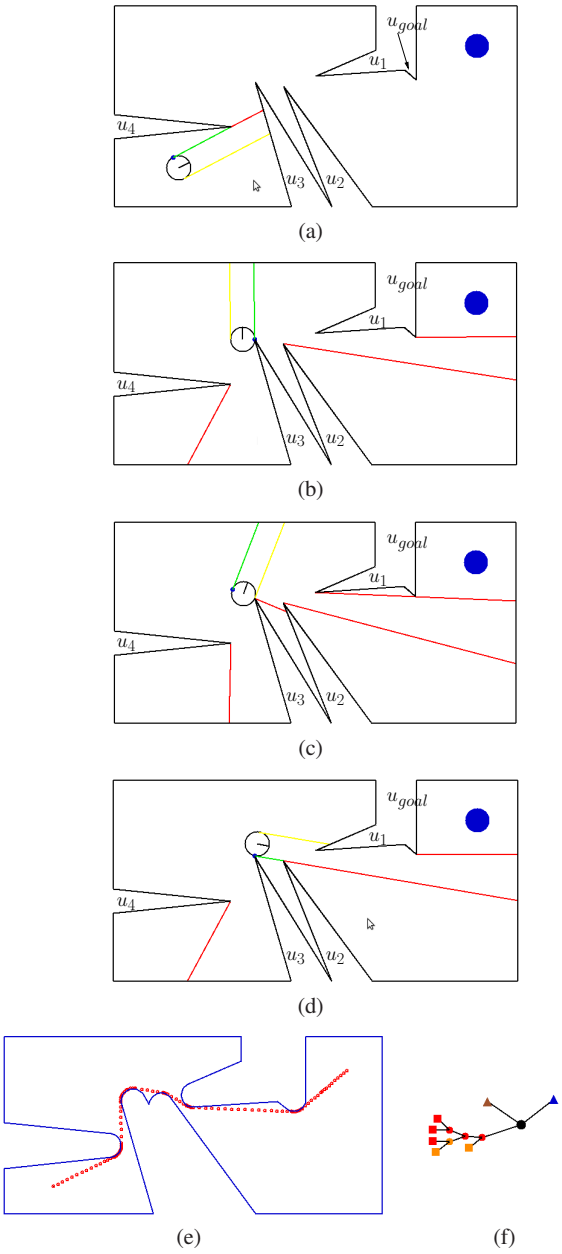


Fig. 21 An example of a hidden vertex, u_1 is the fugure

robot, gives as a consequence the existence of a geometric solution for the navigation problem of reaching a landmark.

Important directions for future work include multiple connected environments (environments with holes) and bounds with respect to optimality for the cases in which all sensing conditions are not met. We also propose as future work to include suitable properties over the robot controls. For instance, to include smooth transitions between controls or to propose control laws able to deal with noise. We want to develop a feedback-based control law to follow the environment boundary, which is useful for both exploration and

navigation, and we would like to implement the exploration strategy presented in [18] in a real robot.

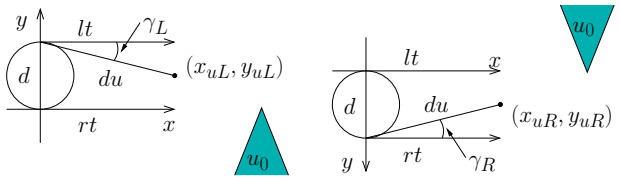
References

1. P. K. Agarwal, T. Biedl, S. Lazard, S. Robbins, S. Suri and S. Whitesides. Curvature-Constrained Shortest Paths in a Convex Polygon. *SIAM J. Comput.*, 31(6):1814-1851, 2012.
2. D.J. Balkcom and M.T. Mason. Time optimal trajectories for bounded velocity differential drive vehicles. *Int. J. of Robotics Research*, 21(3):199–217, 2002.
3. A. Bicchi, G. Casalino, and C. Santilli. Planning shortest bounded-curvature paths for a class of nonholonomic vehicles among obstacles. *J. of Intelligent Robots Systems*, 16(4):387–405, 1996.
4. D. Bilò, Y. Disser, S. Suri M. Mihalák, E. Vicari, and P. Widmayer. Reconstructing visibility graphs with simple robots. *Theoretical Computer Science*, 444:52–59, 2012.
5. L. Bobadilla, F. Martinez, E. Gobst, K. Gossman, and S. M. LaValle. Controlling wild mobile robots using virtual gates and discrete transitions. In *Proc. of American Control Conference*, pages 743–749, 2012.
6. J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1179–1187, 1989.
7. J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *Proc. of IEEE Symposium on Foundations of Computer Science*, pages 49–60, 1987.
8. D. Z. Chen and H. Wang. Paths among curved obstacles in the plane. In *Proc. of Computing Research Repository*, 2011.
9. L. P. Chew. Planning the shortest path for a disc in $O(n^2 logn)$ time. In *Proc. ACM Symposium on Computational Geometry*, 1985.
10. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications, 2nd Ed.* Springer-Verlag, 2000.
11. H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: Part I. *IEEE Robotics and Automation Magazine*, 13(2):99–110, 2006.
12. S. K. Ghosh. *Visibility Algorithms in the Plane*. Cambridge University Press, 2007.
13. S. K. Ghosh and D. M. Mount. An output sensitive algorithm for computing visibility graphs. In *Proc. of IEEE Symposium on Foundations of Computer Science*, pages 11–19, 1987.
14. L. Guilamo, B. Tovar, S. M. LaValle: Gap Navigation Trees: Minimal Representation for Visibility-based Tasks. In M. Erdmann et al., editor, *Proc. of the Tenth Workshop on the Algorithmic Foundations of Robotics: Springer Tracts in Advanced Robotics*, pages 425–440. 2004.
15. J.-B.. Hayet, H. Carlos, C. Esteves, and R. Murrieta-Cid. Motion planning for maintaining landmarks visibility with a differential drive robot. *Robotics and Autonomous Systems*, 4(62):456–473, 2014.
16. O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.
17. A. Kolling and S. Carpin. Extracting surveillance graphs from robot maps. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 11–19, 2008.
18. G. Laguna, R. Murrieta-Cid, H.M. Becerra, R. Lopez-Padilla, and S.M. LaValle. Exploration of an unknown environment with a differential drive disc robot. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 2527–2533, 2014.
19. Y. Landa and R. Tsai. Visibility of point clouds and exploratory path planning in unknown environments. *Communications in Mathematical Sciences*, 6(4):881–913, 2008.
20. J.-P. Laumond, P.E. Jacobs, M. Taïx, and R.M. Murray. A motion planner for nonholonomic mobile robots. *IEEE Trans. on Robotics and Automation*, 10(5):577–593, 1994.
21. S. M. LaValle. Sensing and filtering: A fresh perspective based on preimages and information spaces. In *Foundations and Trends in Robotics Series*. Now Publishers, Delft, The Netherlands, 2012.
22. S.M. LaValle, *Planning Algorithms*, Cambridge University Press; 2006.
23. S.M. LaValle. Robot motion planning: A game-theoretic foundation. *Algorithmica*, 26(3-4):430–465, 2000.
24. Y. Liu and S. Arimoto. Finding the shortest path of a disc among polygonal obstacles using a radius-independent graph. *IEEE Transactions on Robotics*, 11(5):682–691, 1995.
25. R. Lopez-Padilla, R. Murrieta-Cid, and S.M. LaValle. Optimal gap navigation for a disc robot. In E. Frazzoli et al., editor, *Proc. of the Tenth Workshop on the Algorithmic Foundations of Robotics: Springer Tracts in Advanced Robotics*, pages 123–138. 2013.
26. M. Mikawa, Y. Morimoto, and K. Tanaka. Guidance method using laser pointer and gestures for librarian robot. In *Proc. of IEEE Int. Conf. on Robot and Human Interactive Communication*, pages 416–419, 2011.
27. J. Minguez and L. Montano. Nearness diagram (nd) navigation: collision avoidance in troublesome scenarios. *IEEE Transactions on Robotics and Automation*, 20(1):45–59, 2004.
28. J. S. B. Mitchell. Shortest paths and networks. In J. E. Goodman and editors J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 607–641. 2004.
29. J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem. *Journal of the ACM*, 38:18–73, 1991.
30. L. Murphy and P. Newman. Using incomplete online metric maps for topological exploration with the gap navigation tree. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 2792–2797, 2008.
31. J.A. Reeds and L.A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific J. of Mathematics*, 145(2):367–393, 1990.
32. J. H. Reif and Z. Sun. An efficient approximation algorithm for weighted region shortest path problem. In B. R. Donald, K. M. Lynch, and D. Rus editors., editors, *Algorithmic and Computational Robotics: New Directions*, pages 191–203. 2001.
33. P. Souères and J.-P. Laumond. Shortest paths synthesis for a car-like robot. *IEEE Trans. on Automatic Control*, 41(5):672–688, 1996.
34. J. Stillwell. *The Four Pillars of Geometry*. Springer, 2005.
35. H. Ta, D Kim, and S. Lee. A novel laser line detection algorithm for robot application. In *Proc. of Int. Conf. on Control, Automation and Systems*, pages 361–365, 2011.
36. C.J. Taylor and D. Kriegman. Vision-based motion planning and exploration algorithms for mobile robots. *IEEE Transactions on Robotics and Automation*, 14(3):417–426, 1998.
37. S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.
38. B. Tovar and S. M. LaValle and R. Murrieta Optimal Navigation and Object Finding without Geometric Maps or Localization. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 464–470, 2003.
39. B. Tovar, R. Murrieta-Cid, and S. M. LaValle. Distance-optimal navigation in an unknown environment without sensing distances. *IEEE Transactions on Robotics*, 23(3):506–518, 2007.
40. H. Wang, Y. Chen, and P. Soueres. A geometric algorithm to compute time-optimal trajectories for a bidirectional steered robot. *IEEE Transactions on Robotics*, 25(2):399–413, 2009.

A Local Reference Frames for Rotation in Place

For a rotation in place, two local reference frames are defined. One reference frame is defined by point rp and a right vertex. The other ref-

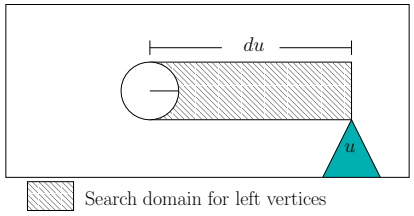
erence frame is defined by point lp and a left vertex. The type of a gap is relative to the location of the omnidirectional sensor. Therefore, the omnidirectional sensor must be located at rp to observe right gaps and at lp to observe left gaps. Consequently, the locations of right vertices w.r.t. a local reference frame are computed with the omnidirectional sensor placed at rp and the locations of left vertices are computed with the omnidirectional sensor placed at lp.



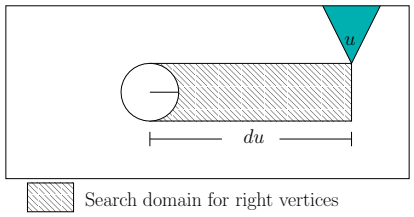
(a) Location of a left vertex (x_{uL}, y_{uL}) on a local reference frame $\mathscr{F}$ defined by rp and u_0.

(b) Location of right vertex (x_{uR}, y_{uR}) on a local reference frame $\mathscr{F}$ defined by lp and u_0.

Fig. 22 Locations of vertices over local reference frames.

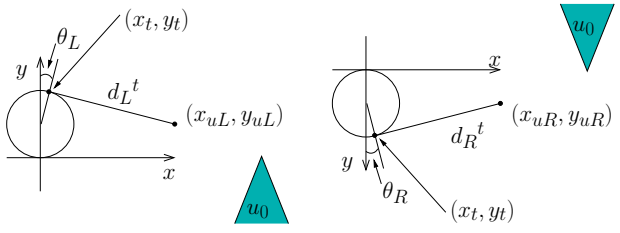


(a) Search domain in a local reference frame $\mathscr{F}$ defined by point rp and a right vertex



(b) Search domain in a local reference frame $\mathscr{F}$ defined by point lp and a left vertex

Fig. 23 Regions in the local reference frames (called search domains) for a rotation in place



(a) $d_L{}^t$ and θ_L are depicted in the figure.

(b) $d_R{}^t$ and θ_R are depicted in the figure.

Fig. 24 Robot rotation in place: Distances and angles of alignment.

A.1 Locations of vertices

Fig. 22(a) shows the location of a left vertex (x_{uL}, y_{uL}) on a local reference frame $\mathscr{F}$ defined by point rp and a right vertex u_0. Fig. 22(b) shows the location of a right vertex (x_{uR}, y_{uR}) on a local reference frame $\mathscr{F}$ defined by point lp and a left vertex u_0.

Equation 1 indicates the coordinates of left vertices in a reference frame $\mathscr{F}$ defined by the point rp and the right vertex u_0 based on distance d_u and angle γ_L, see Fig. 22(a). Equation 2 indicates the coordinates in the same reference frame $\mathscr{F}$ of a right vertex to which the robot is aligned.

$$\begin{aligned} x_{uL} &= d_u \cos\gamma_L \\ y_{uL} &= d - d_u \sin\gamma_L \end{aligned} \tag{1}$$

$$\begin{aligned} x_{uR} &= {d_R}^t \\ y_{uR} &= 0 \end{aligned} \tag{2}$$

where d denotes the robot diameter and ${d_R}^t$ is the distance from rp to vertex u_0 (rt is aligned with u_0). When the reference frame $\mathscr{F}$ is defined by lp and a left vertex (see Fig. 22(b)), there are other similar equations to compute the coordinates of right vertices in these reference frames. In this case the robot is aligned to a left vertex.

A.2 Search domains

Fig. 23 shows the search domains for a rotation in place. The objective of a search domain is to define a region in the local reference frame $\mathscr{F}$ where a candidate vertex must lie. Fig. 23(a) shows the search domain in a local reference $\mathscr{F}$ defined by point rp and a right vertex. It is in this search domain, where the left vertices that might block a right vertex can be located. This domain is delimited by directions rt and lt assuming that the robot is aligned with the right vertex. Fig. 23(b) shows the search domain in a local reference defined by point lp and a left vertex. It is in this region where the right vertices that might block a left vertex can be located.

A.3 Distances and angles of alignment

Here we provide equations to calculate distance ${d_L}^t$ and angle θ_L in a local frame $\mathscr{F}$ defined by rp and a right vertex.

Fig. 24(a) illustrates the angle θ_L that the robot must rotate in clockwise sense to align lt to a left vertex with coordinates (x_{uL}, y_{uL}). θ_L is measured in clockwise sense with respect to the y axis in the local reference frame $\mathscr{F}$ defined by rp and a right vertex. Equation 3 calculates θ_L.

$$\theta_L = \arctan\left(\frac{x_t}{y_t - r}\right) \tag{3}$$

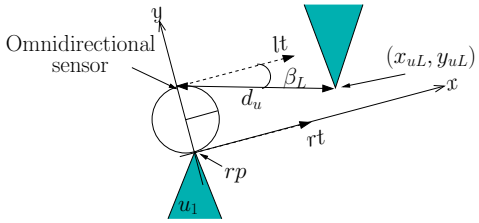
where r denotes the robot radius. The points x_t and y_t are obtained using the Thales's Theorem [34].

Fig. 24(a) shows distance ${d_L}^t$, which is computed assuming that particular direction lt is pointing to a left vertex with coordinates (x_{uL}, y_{uL}). Equation 4 calculates distance ${d_L}^t$.

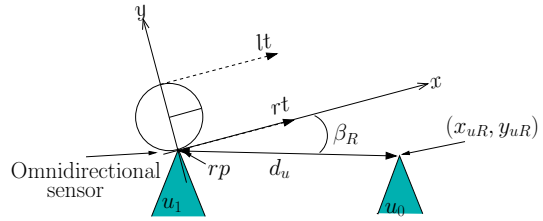
$${d_L}^t = \sqrt{(x_{uL} - x_t)^2 + (y_{uL} - y_t)^2} \tag{4}$$

When the reference frame $\mathscr{F}$ is defined by lp and a left vertex (see Fig. 24(b)). There are similar equations to compute the angle θ_R and distance ${d_R}^t$. Angle θ_R is calculated assuming that the angle increases from y to x (in clockwise sense). Distance ${d_R}^t$ is computed assuming that particular direction rt is pointing to a right vertex.

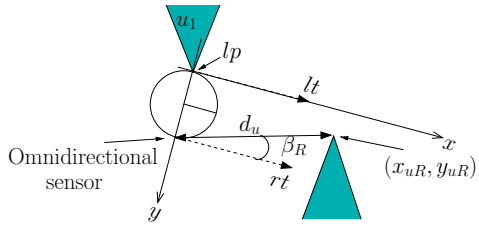
B Local Reference Frames for Robot Rotating w.r.t. Point rp or lp



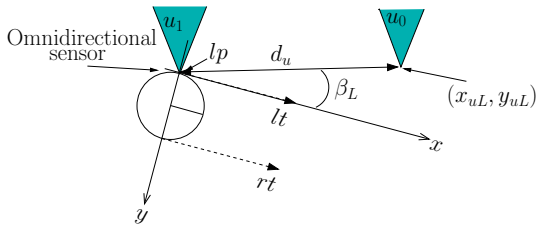
(a) Location of a left vertex w.r.t. a local reference frame $\mathscr{F}$ defined by point rp and direction rt.



(b) Location of a right vertex over a local reference frame $\mathscr{F}$ defined by point rp and direction rt.

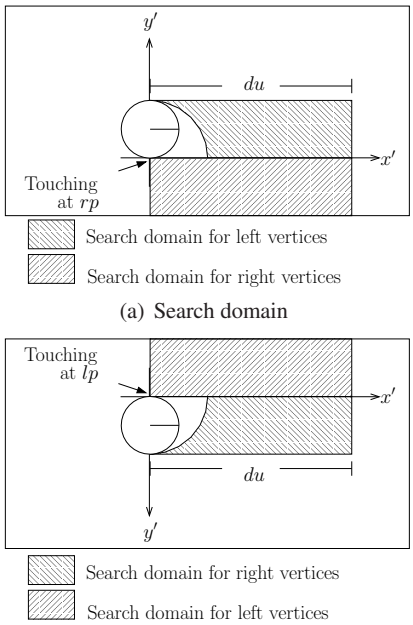


(c) Location of a right vertex over a local reference frame $\mathscr{F}$ defined by point lp and direction lt.



(d) Location of a left vertex over a local reference frame $\mathscr{F}$ defined by point lp and direction lt

Fig. 25 Location of right and left vertices over local reference frames for robot touching ∂E.



(a) Search domain

(b) Search domain

Fig. 26 Regions in the local reference frames (called search domains)

For a rotation w.r.t. point rp or lp, two local reference frames are also defined. One reference frame is defined by point rp and direction rt. The other reference frame is defined by point lp and direction lt.

B.1 Locations of vertices

Fig. 25(a) shows, the location of left vertex, coordinates (x_{uL}, y_{uL}), over a local reference frame $\mathscr{F}$ defined by point rp and direction rt. The location of the vertex is computed based on distance d_u, and an angle β_L measured in clockwise sense starting from particular direction lt.

Equation 5 indicates the coordinates of left vertices in a reference frame $\mathscr{F}$ defined by the point rp and direction rt. See Fig. 25(a).

$$\begin{aligned} x_{uL} &= d_u \cos(\beta_L) \\ y_{uL} &= 2r - d_u \sin(\beta_L) \end{aligned} \tag{5}$$

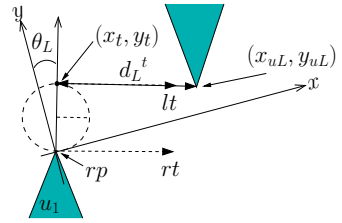
The coordinates of right vertices are given by Equation 6 (see Fig. 25(b)).

$$\begin{aligned} x_{uR} &= d_u \cos(\beta_R) \\ y_{uR} &= -d_u \sin(\beta_R) \end{aligned} \tag{6}$$

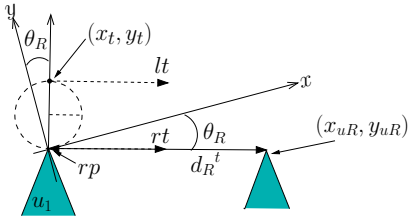
Angle β_L is measured in clockwise sense starting from particular direction lt. The omnidirectional sensor is located at point lp to calculate the locations of left vertices and it is located at point rp to calculate the locations of right vertices. This is done to correctly *measure* angles β_R and β_L. There are equivalent equations to indicate the coordinates of right and left vertices over a reference frame defined by point lp and direction lt, see Figs. 25(c) and 25(d).

B.2 Search domains

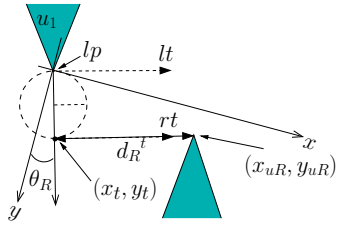
Fig. 26 shows the search domains for a rotation w.r.t. point rp or lp. The objective of a search domain is to define a region in the local reference frame where a candidate vertex must lie. Fig. 26(a) shows the



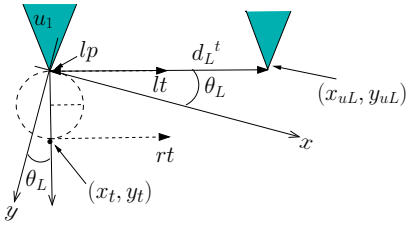
(a) Robot rotation angle θ_L (this is the rotation angle to align lt with a left vertex) and distance $d_L{}^t$.



(b) Robot rotation angle θ_R (this is the rotation angle to align rt with a right vertex) and distance $d_R{}^t$.



(c) Robot rotation angle θ_R (this is the rotation angle to align rt with a right vertex) and distance $d_R{}^t$.



(d) Robot rotation angle θ_L (this is the rotation angle to align lt with a left vertex) and distance $d_L{}^t$.

Fig. 27 Robot rotation touching ∂E at point rp or lp: Distances and angles of alignment used to find a candidate vertex.

search domain for the case in which the robot is touching ∂E at point rp. Fig. 26(b) shows the search domain when the robot is touching ∂E at point lp. Note that in Fig. 25(a) direction rt defining the local reference frame $\mathscr{F}$ (see Subsection B.1) is not aligned with the goal vertex. Hence, a rotation of the reference frame defined by point rp and direction rt is needed. This is done to locate the vertices in a new reference frame $\mathscr{F}'$ defined by x' and y', in which the x' axis is aligned with the goal vertex. Note that, the robot is actually oriented as according to reference frame $\mathscr{F}$ as shown in Fig. 25, but a virtual rotation over $\mathscr{F}$ is done to obtain $\mathscr{F}'$. This is to establish the search domain as if the robot would be aligned to the goal vertex. Indeed, the search domain is drawn over the reference frame $\mathscr{F}'$ defined by x' and y'. All this is done to determine the vertices that might block the goal vertex.

B.3 Distances and angles of alignment

Equation 7 calculates the robot rotation angle with respect to the y axis of the local reference frame $\mathscr{F}$ equivalent to the angle needed to align particular direction lt to a left vertex, by executing a robot clockwise rotation w.r.t. point rp (see Fig. 27(a)).

$$\theta_L = \arctan\left(\frac{x_t}{y_t}\right) \tag{7}$$

Again, the points x_t and y_t are obtained based on the Thales's Theorem [34].

Equation 8 calculates distance $d_L{}^t$. Fig. 27(a) shows distance $d_L{}^t$. To compute this distance, it is assumed that direction lt points to a left vertex associated with $d_L{}^t$.

$$d_L{}^t = \sqrt{(x_{uL} - x_t))^2 + (y_{uL} - y_t)^2} \tag{8}$$

$d_R{}^t$ is measured directly by the omnidirectional sensor (see Fig. 27(b)). θ_R is also measured directly by the omnidirectional sensor (see Fig. 27(b)). There are totally analogous equations to compute angles θ_R θ_L, $d_R{}^t$ and $d_L{}^t$ over a local reference frame defined by point lp and direction lt. Refer to Figs. 27(c) and 27(d).

C Cases in the Finite State Machine

The cases in the FSM are devided in two, the cases in procedure ALIGN and the cases in procedure RPALIGN.

C.1 Cases in procedure ALIGN

Case 1-R: alignment to a right goal vertex

The robot executes a rotation in place to get aligned with the goal vertex, if the path is not blocked then the robot decides that an optimal path toward the goal vertex is a straight line. Else, a detour is needed and the machine transits to Case 3-L.

This case is shown in blue in Fig. 9. In this case there are 4 states:

1. RV1 indicates that the task of the robot is to align rt to a right goal vertex.
2. TCW1 places the omnidirectional sensor at point rp.
3. RCCW1 makes the robot to rotate to align rt to a right goal vertex.
4. T1B1 decides whether or not the right goal vertex is blocked (blockage detection of type 1).

The input observation vectors that trigger a transition between two states in Case 1-R are shown in Table 7.

Case 2-R: alignment to rp_Λ

The robot executes a rotation in place to get aligned with the landmark. If during the rotation in place motion the landmark is at least partially occluded by a vertex then the machine transits to Case 1-R. Else, once the robot is aligned with the landmark, the turret moves the omnidirectional sensor to the opposite side of the robot. If during the turret's motion the landmark is at least partially occluded by a vertex then the machine transits to Case 1-L. Otherwise the landmark is reachable by a straight line motion.

Case 2-R is shown in green in Fig. 9. There are 3 states in this case:

1. RL1 indicates that the task given to the robot is to align rt to rp_Λ.
2. RCCW2 makes the robot to rotate to align rt to rp_Λ.
3. TCCW1 places the omnidirectional sensor at point lp.

Table 7 Observation vectors Case 1-R

	FI	LV	FR	RP	LP	VR	VL	AL	BL	UN	GT	CT	O1	O2
yn_4	x	x	x	x	x	x	x	x	x	x	x	x	x	1
yn_5	x	0	x	x	x	x	x	x	x	x	x	x	1	0
yn_6	x	x	x	x	x	x	x	0	x	x	x	x	1	x
yn_7	x	x	x	x	x	x	x	0	x	x	x	x	0	x
yn_8	x	x	x	x	x	x	x	1	x	x	x	x	1	x
yn_9	x	x	x	x	x	x	x	0	x	x	x	x	x	x
yn_{10}	x	x	x	x	x	x	x	1	x	x	x	x	x	x
yn_{11}	x	x	x	x	x	x	x	x	0	x	x	x	x	x
yn_{12}	x	x	x	x	x	x	x	x	1	x	x	x	x	x

Table 8 Observation vectors Case 2-R

	FI	LV	FR	RP	LP	VR	VL	AL	BL	UN	GT	CT	O1	O2
yn_0	x	0	x	x	x	x	x	x	x	x	0	x	x	x
yn_1	x	0	x	x	x	x	x	x	x	x	1	x	x	x
yn_{13}	x	x	x	x	x	x	x	0	1	x	x	x	x	x
yn_{14}	x	x	x	x	x	x	x	1	1	x	x	x	x	x
yn_{15}	x	1	x	x	x	x	x	0	x	x	x	x	x	x
yn_{16}	x	1	x	x	x	x	x	1	x	x	x	x	x	x
yn_{17}	x	1	x	x	x	x	x	x	x	x	x	x	1	1
yn_{18}	x	1	x	x	x	x	x	1	x	x	x	x	0	0

Table 9 Observation vectors Case 3-L

	FI	LV	FR	RP	LP	VR	VL	AL	BL	UN	GT	CT	O1	O2
yn_{19}	x	x	x	x	x	1	0	0	0	x	x	0	1	1
yn_{20}	x	x	x	x	x	x	1	x	x	x	x	0	0	0
yn_9	x	x	x	x	x	x	x	0	x	x	x	x	x	x
yn_{10}	x	x	x	x	x	x	x	1	x	x	x	x	x	x
yn_{11}	x	x	x	x	x	x	x	x	0	x	x	x	x	x
yn_{12}	x	x	x	x	x	x	x	x	1	x	x	x	x	x

The inputs observation vectors in Case 2-R are shown in Table 8.

Case 3-L: alignment to a left candidate vertex

The main objective of this case is to find a left candidate vertex and to decide whether or not that vertex is a sub-goal vertex. The locations of the vertices depends on a given local reference frame. For a robot rotation in place, the vertices are located in the local reference frame $\mathscr{F}$ defined in Appendix A. Based on the locations of the vertices on $\mathscr{F}$, the distance ${d_L}^t$ is computed. ${d_L}^t$ is obtained assuming that the particular direction lt is pointing to a left vertex. The angle θ_L that the robot needs to rotate (clockwise) to align lt to a left vertex is also computed. Based on distances ${d_L}^t$ and angles θ_L to each left vertex on the region (search domain) presented in Appendix A, a candidate vertex is found using definition 6.

In this Case 3-L, first, a candidate vertex is found and the robot rotates in place to get align with this candidate vertex. Once the robot is aligned to this vertex, if the vertex is blocked then the robot searches a new candidate vertex (Case 3-R). The past two steps are repeated until the candidate vertex is not blocked, this no blocked candidate vertex becomes the sub-goal vertex.

Case 3-L is shown in red in Fig. 9. There are 3 states in this case:

1. TCCW2 places the omnidirectional sensor at point lp, locates left vertices and find a candidate vertex.
2. RCW1 makes the robot to rotate to align lt to the left candidate vertex.
3. T1B2 decides whether or not the left candidate vertex is blocked (blockage detection of type 1)).

The input observation vectors in Case 3-L are shown in Table 9.

C.2 Cases in procedure RPALIGN

The state START2 is the initial state of the procedure RPALIGN, it decides whether the robot must get aligned with a left goal vertex, a right goal vertex or to rp_A. The observation vectors that make this state to transit to other state are shown in Table 10.

Case I-RP: align the robot to a right goal vertex

The robot decides whether or not it can get aligned with a right goal vertex without losing path optimality. This decision is made using the angle between the particular direction rt and the goal vertex. This angle is measured from rt to the direction of the goal vertex in clockwise sense. If that angle is smaller than π then the alignment is possible. If the alignment is possible the robot rotates to get aligned with the goal vertex. Once, the robot is aligned with the goal vertex, the robot decides whether or not the path to the goal vertex is blocked. This decision is made using distances ${d_R}^t$ and d_L. If ${d_R}^t \leq d_L$ then the path is not blocked. If the alignment is not possible (blockage detection of type 2) or ${d_R}^t > d_L$ (blockage detection of type 1) the FSM transits to case IV, else the goal vertex is not blocked and is reachable by a straight line robot motion.

This case is shown in blue in Fig. 11. In this case there are 3 states:

1. RV2 indicates that the task of the robot is to align rt to a right goal vertex, and decides whether or not there is a blockage (detection of type 2).
2. RPCW1 makes the robot rotate w.r.t point rp to align rt to the right goal vertex.
3. T1B5 decides whether or not the right goal vertex is blocked (detection of type 1).

The input observation vectors that triggers a transition between two states in Case I-RP are shown in Table 11.

Case II-RP: align the robot to a landmark

The robot decides whether or not it can get aligned with the landmark without losing path optimality. This decision is made using the angle between the particular direction rt and the point rp_Λ on the landmark. The angle is measured from rt to the direction of point rp_Λ in clockwise sense. If that angle is smaller than π then the alignment is possible, otherwise there is a blockage (detection of type 2). If the alignment is not possible the FSM transits to Case III-RP. Else, once the robot is aligned with the landmark, the turret moves the omnidirectional sensor to the opposite side of the robot. If during the turret's

Table 10 Observation vectors for state START2

	FI	LV	FR	RP	LP	VR	VL	AL	BL	UN	GT	CT	O1	O2
yn_{28}	x	0	x	x	x	0	0	x	0	x	0	0	x	x
yn_{29}	x	0	x	x	x	0	0	x	x	x	1	0	x	x
yn_{30}	x	1	x	x	x	0	0	x	x	x	0	0	x	x

Table 11 Observation vectors Case I-RP

	FI	LV	FR	RP	LP	VR	VL	AL	BL	UN	GT	CT	O1	O2
yn_{11}	x	x	x	x	x	x	x	x	0	x	x	x	x	x
yn_{12}	x	x	x	x	x	x	x	x	1	x	x	x	x	x
yn_{9}	x	x	x	x	x	x	x	0	x	x	x	x	x	x
yn_{10}	x	x	x	x	x	x	x	1	x	x	x	x	x	x

Table 12 Observation vectors Case II-RP

	FI	LV	FR	RP	LP	VR	VL	AL	BL	UN	GT	CT	O1	O2
yn_{31}	x	x	x	x	x	1	x	x	0	x	x	x	x	x
yn_{32}	x	x	x	x	x	1	x	x	1	x	x	x	x	x
yn_{9}	x	x	x	x	x	x	x	0	x	x	x	x	x	x
yn_{14}	x	x	x	x	x	x	x	1	1	x	x	x	x	x
yn_{33}	x	x	x	x	x	x	x	x	0	x	x	x	1	1
yn_{34}	x	0	x	x	x	x	x	x	0	x	0	x	x	x
yn_{35}	x	1	x	x	x	x	x	1	0	x	x	x	0	0

motion the landmark is at least partially occluded by a vertex (detection of type 4) then the machine transits to Case III-RP. Otherwise the landmark is reachable by a straight line motion, there is not blockage.

Case II-RP is shown in green in Fig. 11. There are 3 states in this case:

1. RL2 indicates that the task given to the robot is to align rt to rp_A, and decides whether or not there is a blockage (detection of type 2).
2. RPCW2 makes the robot to rotate w.r.t. point rp to align rt to rp_A.
3. TCCW4 places the omnidirectional sensor at point lp, the motion is done to detect a blockage (detection of type 4).

The inputs observation vectors in Case II-RP are presented in Table 12.

Case III-RP: align the robot to a left goal vertex

The robot decides whether or not the left goal vertex is blocked (detection of type 3). To make this decision the vertices are located in the local reference frame $\mathscr{F}$ presented in Appendix B. Using the vertices' locations, the distances and angles of alignments ${d_R}^t$, ${d_L}^t$, θ_R and θ_L are computed. Based on those distances and angles the decision is made.

A right vertex blocks the left goal vertex if it has a θ_R angle smaller than the angle θ_L related to the left goal vertex, and a distance ${d_R}^t$ smaller than distance ${d_L}^t$ related to that vertex. If the vertex is not blocked the robot rotates w.r.t. rp to get aligned with the vertex. Else, the FSM transits to Case IV-RP.

This case also considers the next complication. It might happen that during the turret motion (state TCCW5), the left goal gap merges with a right gap, this union yields a right gap. See Fig. 28. The vertex that generates this right gap is the vertex that the robot is touching. To consider this issue the robot rotates w.r.t rp until the right gap splits.

Case III-RP is shown in yellow in Fig. 11. There are 8 states in this case:

1. LV2 indicates that the task of the robot is to align lt to left goal vertex, and in this state the right vertices are located in the local reference frame $\mathscr{F}$ presented in Appendix B.
2. TCCW5 places the omnidirectional sensor at point lp.
3. RPCW3 makes the robot to rotate w.r.t. point rp until the right gap splits.
4. D in this state the left vertices are located in the local reference frame $\mathscr{F}$ presented in Appendix B.
5. T2B1 this state decides whether or not there is a blockage (detection of type 3) to the goal vertex.

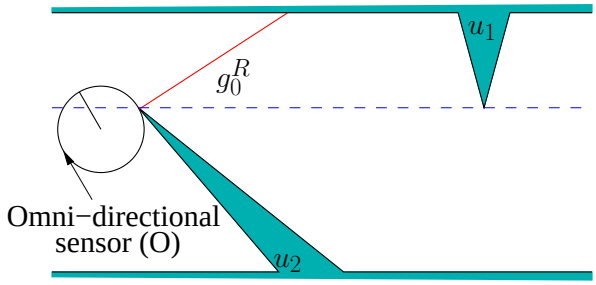


Fig. 28 A left goal gap merges with a right gap

6. RPCW4 makes the robot rotate w.r.t. point rp to align lt to left goal vertex.
7. TCW4 places the omnidirectional sensor at point rp, and in this state the right vertices are located in the local reference frame $\mathscr{F}$ presented in Appendix B.
8. TCCW6 places the omnidirectional sensor at point lp.

The input observation vectors in Case III-RP are presented in Table 13.

Case IV-RP: align the robot to a sub-goal vertex

In this case the path toward a goal vertex or landmark is blocked. To solve this case, first the robot finds a sub-goal vertex, it rotates to get aligned with this vertex and then it travels in a straight line toward a sub-goal vertex.

First, the locations of right and left vertices are computed, using the local reference frame $\mathscr{F}$ presented in Appendix B. Algorithm 1 is used to find a sub-goal vertex.

If the sub-goal vertex is a left one then the robot rotates w.r.t. rp to align lt to the left sub-goal vertex. If the gap generated by that vertex merges with other gap (generated by a hidden vertex) then the sub-goal vertex is re-calculated using Algorithm 1.

If the sub-goal vertex is a right one then the robot rotates w.r.t. rp to align rt to the right sub-goal vertex. The robot is always able to align rt with the right sub-goal vertex, without losing global optimality. Once, the robot is aligned with the sub-goal vertex, the robot decides whether or not the path to the sub-goal vertex is blocked. This decision is made using distances ${d_R}^t$ and d_L. If ${d_R}^t < d_L$ then the path is not blocked. It is important to note that the only manner that Algorithm 1 does not detect that a left vertex blocks the path toward the right goal

Table 13 Observation vectors Case III-RP

	FI	LV	FR	RP	LP	VR	VL	AL	BL	UN	GT	CT	O1	O2
yn_{36}	x	x	x	x	x	1	x	x	x	x	x	x	x	x
yn_{37}	x	x	x	x	x	x	x	x	x	x	0	x	1	1
yn_{38}	x	x	x	x	x	x	x	x	x	x	0	x	0	0
yn_{39}	x	x	x	x	x	x	x	x	x	1	1	x	1	1
yn_{40}	x	x	x	x	x	x	x	x	x	1	1	x	0	0
yn_{41}	x	x	x	x	x	0	0	x	x	x	1	0	x	x
yn_{42}	x	x	x	x	x	0	0	x	x	x	0	0	x	x
yn_{21}	x	x	x	x	x	x	x	x	x	x	x	x	1	1
yn_{43}	x	x	x	x	x	x	x	x	x	x	x	x	0	0
yn_{44}	x	x	x	x	x	x	x	x	x	x	x	x	0	1
yn_{45}	x	x	x	x	x	1	x	x	x	x	x	x	1	0
yn_{46}	x	x	x	x	x	0	0	0	x	0	x	0	x	x
yn_{47}	x	x	x	x	x	0	0	1	x	0	x	0	x	x
yn_{48}	x	x	x	x	x	0	0	0	x	1	x	0	x	x
yn_{49}	x	x	x	x	x	x	1	x	x	x	x	x	x	x
yn_{50}	x	x	x	x	x	x	x	x	0	0	x	0	x	x
yn_{51}	x	x	x	x	x	x	x	x	0	0	x	1	x	x

Table 14 Observation vectors Case IV-RP

	FI	LV	FR	RP	LP	VR	VL	AL	BL	UN	GT	CT	O1	O2
yn_{50}	x	x	x	x	x	x	x	x	0	0	x	0	x	x
yn_{51}	x	x	x	x	x	x	x	x	0	0	x	1	x	x
yn_{52}	x	x	x	x	x	1	x	0	x	x	x	x	1	1
yn_{53}	x	x	x	x	x	x	1	x	x	x	x	x	0	0
yn_{21}	x	x	x	x	x	x	x	x	x	x	x	x	1	1
yn_{54}	x	x	x	x	x	0	0	x	x	x	x	0	0	1
yn_{45}	x	x	x	x	x	1	x	x	x	x	x	x	1	0
yn_{55}	x	x	x	x	x	x	x	0	x	0	x	x	x	x
yn_{56}	x	x	x	x	x	x	x	1	x	0	x	x	x	x
yn_{57}	x	x	x	x	x	x	x	0	x	1	x	x	x	x
yn_{58}	x	x	x	x	x	1	0	0	x	x	x	0	1	1
yn_{44}	x	x	x	x	x	x	x	x	x	x	x	x	0	1
yn_{59}	x	x	x	x	x	x	x	x	x	x	x	x	1	0
yn_{9}	x	x	x	x	x	x	x	0	x	x	x	x	x	x
yn_{10}	x	x	x	x	x	x	x	1	x	x	x	x	x	x
yn_{11}	x	x	x	x	x	x	x	x	0	x	x	x	x	x
yn_{12}	x	x	x	x	x	x	x	x	1	x	x	x	x	x

vertex is that the left vertex does not generate a gap when Algorithm 1 was invoked (that left vertex is a hidden vertex). However, once the robot is aligned with the right sub-goal vertex a left vertex that blocks the path toward the sub-goal vertex must generate a left gap (see subsection 6.2, Lemma 4), so if the path toward the right sub-goal vertex is blocked then the sub-goal vertex is re-calculated using Algorithm 1. If the path toward the right sub-goal vertex is blocked then the robot is always able to keep rotating in the same sense to get aligned with the vertex generating the blockage.

Case IV-RP is shown in red in Fig. 11. There are 9 states in this case:

1. TCCW9 in this state, first, right vertices are located in the local reference frame $\mathscr{F}$ presented in Appendix B. Second, the omnidirectional sensor is placed at point lp, and finally the left vertices are located in the same local reference frame presented $\mathscr{F}$ in Appendix B.
2. A1 this state executes Algorithm 1 and selects a sub-goal vertex.
3. RPCW5 makes the robot to rotate w.r.t to point rp to align lt to left sub-goal vertex.
4. TCW5 places the omnidirectional sensor at point rp, and in this state the right vertices are located in the local reference frame $\mathscr{F}$ presented in Appendix B.
5. TCCW7 places the omnidirectional sensor at point lp, and in this state the left vertices are located in the local reference frame $\mathscr{F}$ presented in Appendix B.
6. TCW6 places the omnidirectional sensor at point rp.
7. RPCW6 makes the robot to rotate w.r.t. point rp to align rt to a right sub-goal vertex.
8. T1B6 This state decides whether or not there is a blockage (detection of type 1) to the right sub-goal vertex.
9. TCCW8 in this state, first, right vertices are located in the local reference frame $\mathscr{F}$ presented in Appendix B. Second, the omnidirectional sensor is placed at point lp, and finally the left vertices are located in the same local reference frame $\mathscr{F}$ presented in Appendix B.

The input observation vectors in Case IV-RP are presented in Table 14.