Optimal Gap Navigation for a Disc Robot

Rigoberto Lopez-Padilla, Rafael Murrieta-Cid and Steven M. LaValle

Abstract This paper considers the problem of globally optimal navigation with respect to Euclidean distance for disc-shaped, differential-drive robot placed into an unknown, simply connected polygonal region. The robot is unable to build precise geometric maps of the environment. Most of the robot's information comes from a gap sensor, which indicates depth discontinuities and allows the robot to move toward them. A motion strategy is presented that optimally navigates the robot to any landmark in the region. Optimality is proved and the method is illustrated in simulation.

1 Introduction

If a point robot is placed into a given polygonal region, then computing shortest paths 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 [4] (an $(n \lg n + m)$ algorithm also exists, in which m is the number of bitangents [7]). 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 exist. Computing shortest paths in three-dimensional polyhedral regions is NP-hard [1]. Allowing costs to vary over regions considerably complicates the problem [14, 16]. See [6, 13]

Rigoberto Lopez-Padilla
Center for Mathematical Research (CIMAT), Guanajuato, Gto. 36240, México e-mail: rigolpz@cimat.mx

Rafael Murrieta-Cid
Center for Mathematical Research (CIMAT), Guanajuato, Gto. 36240, México e-mail: murrieta@cimat.mx

Steven M. LaValle
University of Illinois, Urbana, IL 61801, USA e-mail: lavalle@uiuc.edu

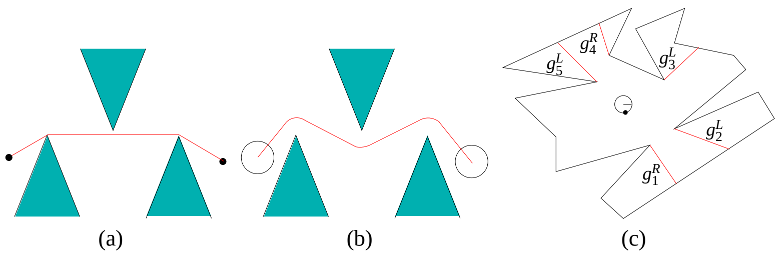


Fig. 1 a) The optimal path for a point robot, b) The optimal path for a disc robot, c) The gap sensor (attached at the small solid disc on the robot boundary) detects the sequence of gaps $G = [g_1^R, g_2^L, g_3^L, g_4^R, g_5^L]$, in which g_1^R and g_4^R are near-to-far gaps and g_2^L, g_3^L, and g_5^L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

for surveys of shortest path algorithms. For recent efforts on curved obstacles, see [2].

The approaches described thus far address a point robot, which is unrealistic in most practical settings. It is therefore interesting to study the case of a disc robot, which could correspond, for example, to a Roomba platform. Various objective functions are possible; we choose to optimize the distance traveled by the center of the robot. Once the robot has nontrivial dimensions, the problem can be expressed in terms of configuration space obstacles. Solutions are presented in [3, 11].

Now suppose that the map of the environment is not given to the robot. It must use its sensors to explore and map the environment to develop navigation strategies. Given strong sensors and good odometry, standard SLAM approaches [5, 17] could be applied to obtain a map that can be used as input to the previously mentioned methods. However, we do not allow the robot to localize itself with respect to a *global* reference frame or to build a geometric map. Instead, it observes the world using mainly a *gap sensor*, introduced in [18], which allows it to determine the directions of discontinuities in depth (distance to the boundary) 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 that encodes precisely the part of the shortest-path visibility graph that is needed for optimal navigation [18]. The learned data structure corresponds exactly to the shortest path tree [6] from the robot's location. This enables the robot to navigate to any previously seen landmark by following the distance-optimal path, even though it cannot directly measure distances. The GNT was extended and applied to exploration in [15]. The GNT was extended to point cloud models in [8]. A larger family of gap sensors is described in [10].

The case of a disc robot is important because real robots have nonzero width. 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 Figure 1 (a) and (b). The robot must instead execute detours from the bitangent. Sensing, characterizing, and optimally navigating around these obstructions is the main difficulty of this paper.

Before proceeding to the detailed model and motion strategy, several points are important to keep in mind:

1. The robot is placed into an environment, but it is not given the obstacle locations or its own location and orientation. Robot *observes* this information over local reference frames. The purpose is to show how optimal navigation is surprisingly possible without ordinary SLAM.
2. The robot first learns the GNT by executing a learning phase, which is described in [9, 18], and needs just minor changes to be used by disc-shaped robots. The process involves iteratively chasing "unknown" gaps, causing each to split or disappear. Eventually, only primitive gaps, which were formed by appearances of gaps (due to inflection ray crossings), and gaps formed by merging primitives remain. This corresponds to learning the entire shortest-path graph.
3. A simple navigation strategy is provided that guides the robot to any landmark placed in the environment by using the learned GNT. We give precise conditions under which the motions are optimal and prove this statement.
4. We believe that even when the optimality conditions are not met, the strategy itself is close to optimal. Therefore, it may be useful in many practical settings to efficiently navigate robots with simple sensor feedback.

Section 2 formally describes the robot model and the sensor-based motion primitives. Section 3 introduces an automaton that characterizes all possible sequences of motion primitives that could occur when executing optimal motions to a landmark. Section 4 describes how obstacle blockages are detected and handled when the robot chases a gap. This includes detours (that is, the modification of the path encoded in the GNT for a point robot) needed to achieve optimal navigation. 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 modeled as a disc with radius r moving in an unknown environment, which could be any compact set $E \subset \mathbb{R}^2$ for which the interior of E is simply connected and the boundary, ∂E, of E is a polygon. Furthermore, it assumed that the collision-free subset of the robot's configuration space C is connected. C-space obstacle corresponds to that of a translating disc, that is, the extended boundary of E which is due to the robot radius [1].

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

2.1 Sensing capabilities

1) Gap sensor: The robot has an omnidirectional gap sensor [10, 18], which is able to detect and track two types of discontinuities in depth information: discontinuities from far to near and discontinuities from near to far (in the counterclockwise direction along ∂E). Figure 1(c) shows a robot in an environment in which the gap sensor detects some near-to-far and far-to-near gaps.

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 $t = R$ means a discontinuity from near to far (the hidden portion is the right) and $t = L$ means a discontinuity from far to near (the hidden portion is to the left). For example, the gap sensor in Figure 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 send the robot on collision-free trajectories that possibly contact the obstacles (moving along the boundary of the free subset of the configuration space is necessary for most optimal paths). These motion primitives, described in detail in Section 2.2, allow the robot to rotate itself so that it is aligned to move the gap sensor directly toward a desired gap, move forward while chasing a gap, and follow ∂E while the sensor is aligned to a gap.

Imagine that a differential drive robot is used. It is assumed that the gap sensor can be moved to two different fixed positions on the robot 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. Figure 3(c) shows the sensor aligned to a near-to-far gap in which the gap sensor is on the right side of the robot. To align the sensor to a far-to-near gap, the robot moves the gap sensor to the left side of the robot.

Finally, let Λ be a static disc-shaped landmark in E with the same radius as the robot. A landmark Λ is said to be *recognized* if the landmark is visible at least partially from the location of the gap sensor. Furthermore, during the exploration phase, if required, the landmark can be reached (hence, the complete landmark would be visible from the location of the gap sensor), by traveling along optimal detours.

2) Side sensors: To detect obstacles that obstruct the robot while it chases a gap, our algorithms need 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 direction tangent to the robot boundary at rp is called rt. The particular direction tangent to the robot boundary at lp is called lt (See Figure 2(a)). Thus, we assume that the omnidirectional sensor is able to *measure* distance. 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 Figure 2(b)).

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

2(c). For avoiding blockages toward the vertex that generates a gap to be chased, the robot rotates. There are two types of rotation: clockwise and counterclockwise. A clockwise rotation can be executed either with respect to rp or a rotation in place (w.r.t. the robot center). Symmetrically, a robot counterclock wise rotation can be executed either with respect to lp or a rotation in place (w.r.t. the robot center). Finally, let d_u be the distance between the omnidirectional sensor and the vertex u_i that originated the gap g_i (in Figure 2(d) $g_i = g_0^R$).

Our motion strategy will require only comparisons of distances to determine which is larger, rather than needing precise distance measurements. Any small error in the comparison (if the distances are close) causes only a small deviation from optimality, which may be relatively harmless in practice. Our approach will furthermore require detecting whether the robot is contacting ∂E at rp or lp to enable wall-following motions.

Distance measurements between the obstacles and lp and rp 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 emulate the laser pointer reading. An omnidirectional sensor can also be used to determine whether the robot is touching ∂E at rp or lp. 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 Figure 2(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 gaps or landmarks are always chased with the omnidirectional sensor; the laser pointer is used to detect obstacles and to correctly align the robot.

2.2 Motion primitives

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 equivalent the vertex that generates it) or a landmark is given to the robot as goal. There are five motion primitives (see Figure 3). 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: $w_r = 1, w_l = -1$.

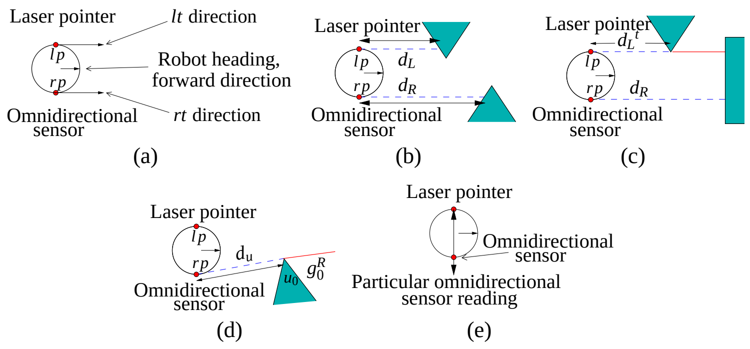


Fig. 2 Side Sensors: (a) Points rp and lp, and directions rt and lt, (b) d_L and d_R, (c) $d_L{}^t$, (d) d_u, (e) Omnidirectional sensor readings for contact detection.

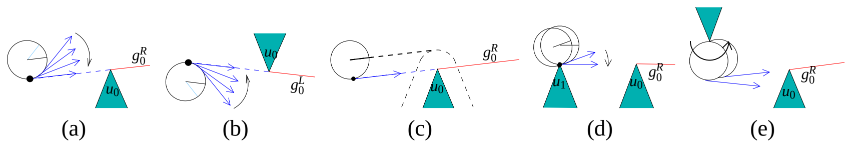


Fig. 3 The motion primitives: (a) Clockwise rotation in place, (b) Counterclockwise rotation in place, (c) Straight line motion, (d) Clockwise rotation w.r.t. to point rp, (e) Counterclockwise rotation w.r.t. to point lp.

- 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: $w_r = 1, w_l = 1$.

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 Sections 3 and 4.

3 The movement automaton

The algorithm for generating optimal navigation motions can be nicely captured by an automaton or (Moore) finite state machine M. See Figure 4. Every state corresponds to the selection and execution of a motion primitive on the robot or it is a decision. Each state transition is triggered by a sensor observation change. There are 21 total states, with clear right/left symmetry. The ten upper states in Figure 4 correspond to near-to-far gaps (the R cases) and the ten lower states correspond to far-to-near gaps (the L cases). The other remaining state is NTOUCHING, which is used when the robot is not touching ∂E and decides whether the gap to be chased is left or right.

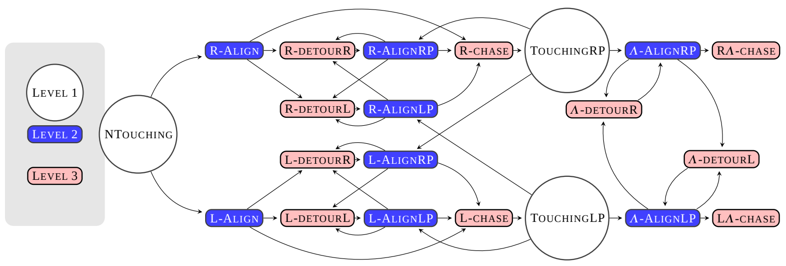


Fig. 4 The sequence of executed primitives depends on sensor feedback. The possible executions are captured by a Moore machine M in which each state applies a specific motion primitive and each transition edge is triggered by a sensing event.

The machine has three main levels (see Figure 4). The first one corresponds to decide whether the goal gap is far-to-near (left gap) or near-to-far (right gap). It also decides whether the robot is touching ∂E. If the robot is touching ∂E, then the first level determines whether the robot is touching it at lp (left side) or rp (right side). The second level is the main one, it detects blockages. According to the decisions made in the first level, the second level makes the robot execute one of the four types of rotations: 1) counterclockwise rotation in place, 2) clockwise rotation in place, 3) clockwise rotation w.r.t. rp and 4) counterclockwise rotation w.r.t. lp. The second level determines whether the path to the goal gap is blocked. According to this decision, the robot executes either a straight line motion toward the gap (the path is not blocked) or executes a detour (the robot travels in a straight line toward the subgoal vertex). The third level is in charge of executing the motion toward the gap to be chased.

In the first level no motion primitive is executed, and in this level, there are three states:

- NTOUCHING: This state happens when the robot is not touching ∂E. It decides whether the gap to be chased is left or right.
- TOUCHINGRP: This state is triggered when the robot is touching ∂E at rp and the gap being chased splits, or the robot goal is a landmark Λ (i.e., the landmark is totally visible to the omnidirectional sensor). The state decides whether the new gap to be chased is a left or right gap. If the new selected gap is a right gap (near-to-far), then the next state will be R-ALIGNRP. If the new selected gap is a left gap (far-to-near), then the next state will be L-ALIGNRP. Finally, whenever the goal is Λ the state will transit to Λ-ALIGNRP.
- TOUCHINGLP: This state is the left symmetric equivalent to TOUCHINGRP.

The second level determines whether the path to the chosen gap is blocked. There are eight states in the second level (four for the right case and four for the left). The states for the right case are:

- R-ALIGN: Right gap alignment executing clockwise rotation in place.
- R-ALIGNLP: Right gap alignment executing counterclockwise rotation w.r.t. lp.

- R-ALIGNRP: Right gap alignment executing clockwise rotation w.r.t. rp.
- Λ-ALIGNRP: Right landmark alignment executing clockwise rotation w.r.t. rp.

There are other four equivalent states when the goal gap is left g_0^L, or the landmark is chased with the omnidirectional sensor located at lp (these are L cases in M). For all of these states, there are three possible transitions. 1) If the path is not blocked, then a straight line motion is allowed. 2) The robot detects a blockage and the subgoal vertex corresponds to a right gap. 3) There is a blockage and the subgoal vertex generates a left gap.

In the third level, the robot always executes a straight line motion. Either the robot moves to the goal gap g_0 (these states are called CHASE) or toward the vertex u that represents the subgoal (these states are called DETOUR). Note that the goal vertex or the vertex that blocks the path can generate a left or right gap; for this reason the states are designed as left or right.

There are ten states in the third level (five for the right case and five for the left). The states for the right case are:

- R-CHASE: The robot moves toward the goal gap g_0^R.
- R-DETOURR: The robot moves toward a subgoal vertex u_n that generates a right gap g_n^R detour.
- R-DETOURL: The robot moves toward a subgoal vertex u_p that generates a left gap g_p^L detour.
- RΛ-CHASE: The robot moves toward Λ, and the omnidirectional sensor is located at rp.
- Λ-DETOURR: The robot moves toward a subgoal vertex u_n that generates a right gap g_n^R detour to the landmark.

There are five symmetrically equivalent states when the goal gap is left g_0^L or the landmark is chased with the omnidirectional sensor located at lp (the L cases in M). This establishes the details of the state machine M.

The next section analyzes blockage detection and gap selection. For lack of space, in this paper we briefly present an algorithm used to determine an optimal detour. This algorithm is described in detail in [12].

4 Blockage detection and optimal detours

During the navigation phase, to detect a blockage, distances d_L, d_R, ${d_L}^t$ and ${d_R}^t$ are used. If direction rt is aligned to a vertex that generates a right gap 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 vertex that generates a left gap and ${d_L}^t > d_R$ then a straight line robot path toward that vertex is blocked. In [12], we prove that during the navigation phase, these conditions are sufficient to detect blockages or declare the robot path collision free. Whenever the path is blocked, the robot executes a detour; that is, the robot travels in a straight line toward another vertex before reaching the vertex associated to the gap being chased (called goal gap). The vertex that generates the original gap to be chased, which is encoded in the GNT, is always visited. For this reason, we call the modified path a detour.

The selection of the gap (or equivalent a vertex) that corresponds to an optimal detour depends on the robot sense of rotation. We call u_p and u_n the vertices that determine an optimal detour. To determine u_p and u_n vertices, it is necessary to compute distances ${d_R}^t$ and ${d_L}^t$. It is also necessary to compute two angles: 1) the angle that the robot needs to rotate (either counterclockwise or clockwise) to align rt to a right vertex (a vertex that generates a right gap), this angle is called θ_R, and 2) the angle that the robot needs to rotate (either counterclockwise or clockwise) to align lt to a left vertex (a vertex that generates a left gap), this angle is called θ_L. To compute these distances and angles, we locate the vertices in *local reference frames*. Either a reference frame is defined by point rp and a right goal vertex or a reference frame is defined by point lp and a left goal vertex. In [12], a detailed description of the construction of these local reference frames is provided. To find a vertex u_n or a vertex u_p, we use two orders. In one order w.r.t. distance, distance ${d_R}^t$ is used to consider vertices that generate right gaps. Symmetrically, distance ${d_L}^t$ is used to consider vertices that generate left gaps. An order from smaller to larger distances including both ${d_R}^t$ and ${d_L}^t$ is generated. 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 vertices that generate right gaps and θ_L is used to consider vertices that generate left gaps. Now, we define u_n and u_p based on these two orders.

Definition 1. The next vertex u_n is the first vertex in clockwise order after the original goal vertex, which is aligned with rt. It is reachable by the robot travelling a straight line path and it corresponds to the optimal detour.

Definition 2. The previous vertex u_p is the last vertex in clockwise order before the original goal vertex, which is aligned with lt. It is reachable by the robot travelling a straight line path and it corresponds to the optimal detour. Refer to Figure 5 and Table 1.

Remark 1. There are two analogous definitions of u_n and u_p, for a counterclockwise rotation.

4.1 Algorithm to find an optimal detour

Table 1 shows an example of the execution of Algorithm 1 and the determination of a u_p vertex; $\uparrow$ indicates the subgoal vertex, $\times$ indicates the vertices that might block the path toward the subgoal vertex, $\otimes$ indicates the vertex selected as subgoal at each iteration, $-$ indicates that the distance to this vertex is smaller than the distance to the subgoal vertex, $+$ indicates that the distance to this vertex is larger than the distance to the subgoal vertex, $\rightarrow$ indicates that for a left vertex, the vertex that must be selected as subgoal is the last in the angular order, and $\leftarrow$ indicates that for a right vertices, the vertex that must be selected as subgoal is the first in the angular order. The algorithm determines that u_f is a u_p vertex corresponding to the optimal detour.

Algorithm 1 : Handle an optimal detour

```
1) The algorithm starts from the goal vertex. If the goal vertex is a left vertex, then go to Step 4.
2) Detect left vertices that block the path toward a right goal vertex.
To block the path toward right goal vertex $u^R_{goal}$, left vertices must have an angle $\theta_L$ larger than the angle $\theta_R$ related to the goal vertex, and a distance ${d_L}^t$ smaller than distance ${d_R}^t$ related to the goal right vertex $u^R_{goal}$. If no vertex blocks the path toward the goal vertex, then go to Step 6.
3) Selection of a left goal vertex.
The left vertex with largest $\theta_L$, the last in the angular order is selected as a new goal vertex.
4) Detect right vertices that block the path toward a left goal vertex.
To block the path toward left goal vertex $u^L_{goal}$, right vertices must have an angle $\theta_R$ smaller than the angle $\theta_L$ related to the goal vertex, and a distance ${d_R}^t$ smaller than distance ${d_L}^t$ related to the left goal vertex $u^L_{goal}$. If no vertex blocks the path toward the goal vertex, then go to Step 6.
5) Selection of a right goal vertex.
The right vertex with smallest angle $\theta_R$, the first in the angular order is selected as a new goal vertex. Go to Step 2.
6) The vertex selected as goal is not blocked.
```

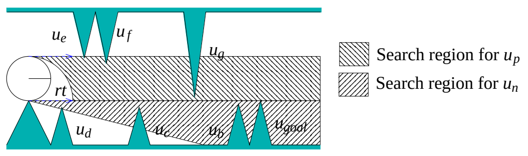


Fig. 5 A u_p vertex

Table 1 Example of orders for selecting a u_p vertex

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_b	u_e	u_c	u_f	u_d	u_g	Vertex	u_d	u_e	u_f	u_c	u_g	u_b	u_{goal}
$\rightarrow$	$\uparrow$		$\times$		$\times$		$\otimes$			-	-		-		$\uparrow$
$\leftarrow$				$\otimes$		$\times$	$\uparrow$		-			-	$\uparrow$	+	+
$\rightarrow$			$\times$	$\uparrow$	$\otimes$					-	-	$\uparrow$	+		
$\leftarrow$					$\uparrow$	$\times$			-		$\uparrow$	+		+	+

Lemma 1. *Algorithm 1 finds the optimal detour in the sense of Euclidean distance toward the goal vertex.*

Proof. The structure of the path representing a detour is a sequence of sub-paths between vertices, hence for the global path to be optimal, each element of the sequence must be locally optimal. Since each vertex selected as subgoal lies on the boundary of the restriction then each element of the sequence is locally optimal. Therefore, the resulting detour is optimal. □

5 The feedback motion strategy

Although M represents the decision component of the system, the commands to the motors can be implemented by simple sensor feedback. Only five binary sensor observations affect the control: 1) the robot is touching ∂E with the left side (point lp); (2) it is touching ∂E with the right side (point rp); (3) the robot is aligned to a gap; (4) there is a blockage; and (5) the type of gap (left 0, right 1). Depending on the observation, one of the five different motion primitives will be executed: (1) straight line motion, (2) counterclockwise rotation in place, (3) clockwise rotation in place, (4) counterclockwise rotation with respect to point lp and (5) clockwise rotation with respect to point rp. 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: $\gamma : \{0,1\}^5 \rightarrow \{-1,0,1\}^2$, in which the sensor observation vector is denoted as $y_i = (rp, lp, aligned, blockage, type)$, to obtain $\gamma(y_i) = (w_r, w_l)$. The set of all 32 possible observation vectors can be partitioned be letting x denote "any value" to obtain: $y_1 = (x,x,1,0,x)$, $y_2 = (0,0,x,1,0)$, $y_3 = (0,0,0,x,0)$, $y_4 = (0,0,x,1,1)$, $y_5 = (0,0,0,x,1)$, $y_6 = (x,1,0,x,x)$, $y_7 = (x,1,x,1,x)$, $y_8 = (1,x,0,x,x)$, $y_9 = (1,x,x,1,x)$.

The strategy γ can be encoded as

$$\begin{array}{ll} \gamma(y_1) = (1,1); & \gamma(y_2 \vee y_3) = (1,-1) \\ \gamma(y_4 \vee y_5) = (-1,1); & \gamma(y_6 \vee y_7) = (1,0) \\ \gamma(y_8 \vee y_9) = (0,1), & \end{array}$$

in which $\vee$ means "or".

6 Proof of optimal navigation

6.1 Non-blocked GNT-encoded paths

In this section we establish that the robot executes a Euclidean, distance-optimal path in the absence of blockages. 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. Let $H = (g_n, g_{n-1}, ..., g_0)$

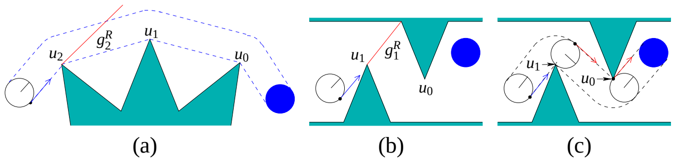


Fig. 6 a) A non-blocked GNT-encoded path that involves only near-to-far gaps, b) A non-blocked GNT-encoded path that involves both types of gaps (near-to-far and far-to-near), c) Robot path.

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 [18].

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. Note that if any of the first three conditions is violated, then the robot movement is blocked by an obstacle and therefore does not execute an optimal path. For the last condition, if C is not connected then there is no solution path. $\square$

Figure 6(a) shows an example of how M generates an optimal path for the non-blocked case. In the figure, the GNT encodes the sequence $H = (g_2^R, g_1^R, g_0^R)$. In this example, the machine M traverses the following sequence of states while generating the appropriate motion primitives: NTOUCHING, R-ALIGN, R-CHASE, TOUCHINGRP, R-ALIGNRP, R-CHASE, TOUCHINGRP, R-ALIGNRP, R-CHASE, TOUCHINGRP, Λ-ALIGNRP, RΛ-CHASE.

Now we describe the association of the states with each gap and the landmark. First, g_2^R is chased, executing states NTOUCHING, R-ALIGN, R-CHASE. Next, g_1^R is chased, executing states TOUCHINGRP, R-ALIGNRP, R-CHASE. Next, g_0^R is chased, executing again the states TOUCHINGRP, R-ALIGNRP, R-CHASE. Finally, Λ is chased, executing states TOUCHINGRP, Λ-ALIGNLP, RΛ-CHASE.

In the previous example all of the gaps in H were of the same type. Using the example illustrated in Figure 6(b), we explain the operation of M when there are different types of gaps. To reach Λ, the robot chases the sequence $H = (g_1^R, g_0^L)$. The resulting sequence is NTOUCHING, R-ALIGN, R-CHASE (from chasing g_1^R), then TOUCHINGRP, L-ALIGNRP, L-CHASE, (from chasing g_0^L), and finally TOUCHINGLP, Λ-ALIGNLP, LΛ-CHASE (from chasing Λ).

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 the configuration space, some bitangent lines disappear. Bitangent lines in the workspace that remain in the configuration space 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 to chasing $g_i \in H$ (or Λ). If this happens it means that: 1) there is no bitangent line between v_{i+1} and v_i in C, 2) the robot is in a zone in which it cannot detect the crossing of a bitangent line in C, 3) there is no clear path to chase Λ when the robot sees Λ, or 4) C is disconnected. These are the Lemma 2 conditions. We present a solution to deal with the first three cases presented above. However, we do not handle the disconnection of C because there is no path to Λ.

If the robot detects a blocked path, then it performs a detour to avoid the obstacles that blocks 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 Section 4, we have presented a proof (see Lemma 1) showing that the detour is optimal. Now, we present the theorem that ensure globally optimal navigation when using M.

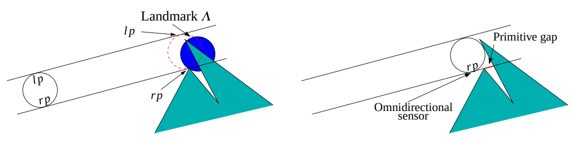
Theorem 1. *The path that the robot center follows when commanded by the automaton M, using the information encoded in the GNT and making detours when the path to chase $g_i \in H$ is blocked or when the path to chase Λ is blocked, is optimal in the sense of Euclidean distance.*

Proof. 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. We have shown that the sequence of connected intervals in E that the robot traverses is only changed when the conditions of Lemmas 2 are not satisfied; therefore, the shortest path for a disc contains the intervals in U. Since each detour is made between consecutive intervals of U, and they are locally optimal (as proved in Lemma 1). Hence, the resulting global path is optimal. □

Constructing a complete GNT: In [18], it has been proved that the construction of the GNT for a point robot will terminate (Lemmas 2 and 3 in [18]). Incompleteness of the GNT is caused by any non-primitive leaves (these that correspond to the portions of the environment that have not been perceived by the robot). The key observation to prove the completeness of the GNT for a point robot is that any time that a new gap appears, it must be primitive. If the gap is chased, it cannot split. Therefore, the only gaps that contribute to the incompleteness of the GNT are ones that either appeared in at the beginning of the construction or were formed by a sequence of splits of these gaps. Now, we prove that the construction GNT for a disc robot must also terminate. As mentioned above, compared with the GNT for a point robot, a path that the robot travels to chase a gap will be of two types: non-blocked paths and blocked paths.

Lemma 3. *The learning (construction) phase of the GNT for a disc robot always terminates and produces the same GNT as in [18].*

Proof. To chase a gap, a disc robot first aligns lt to a left vertex, or rt to a right vertex. If ${d_R}^t > d_L$ then a straight line robot path toward this vertex is blocked. Likewise, if ${d_L}^t > d_R$ then a straight line robot path toward that vertex is blocked. For blocked paths, it has been shown in Lemma 1 that the detour is correct and locally optimal. Furthermore, while the robot traverses the detours, it will always have the information about the identity of the original goal gap to be chased g_i, since it is codified in the GNT. For a non-blocked path, there are two sub-cases: 1) From the aligned robot configuration, vertices that can be touched by the robot with point rp (a right vertex) or point lp (a left vertex) traveling a straight line path from the initial robot configuration are equivalent to chase a gap for a point robot; hence the gap can be marked as a primitive gap. 2) From the aligned robot configuration, vertices that cannot be touched by the robot with point rp (a right vertex) or point lp (a left vertex) traveling a straight line path from the initial robot configuration cannot occlude the landmark. Refer to Figures 7(a) and 7(b). Hence, during the exploration phase the related gaps are marked as primitive gaps (explored gaps) at moment that robot touch ∂E with a point different to lp or rp. Therefore, the construction of the GNT for a disc robot must terminate if and only if it would terminate for a point robot. $\square$



(a) Learning phase (b) Primitive gap

Fig. 7 Gaps marked as primitive gaps during the exploration phase.

7 Implementation

We have implemented the method to further verify its correctness. Figure 8 shows a simulation of the optimal gap navigation for a disc robot. The figure shows snapshots of the simulation program. Figures 8(a) and 8(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 [18]. The landmark to be chased is marked as a blue triangle in the workspace and in the GNT as a leaf

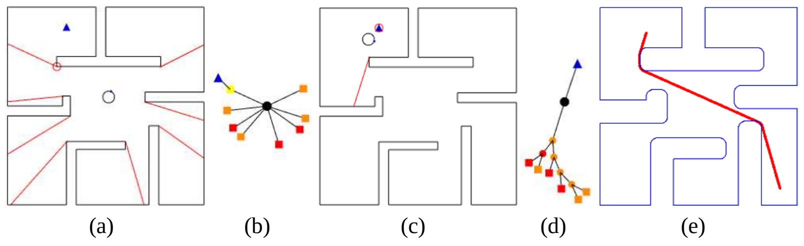


Fig. 8 A simulation of optimal gap navigation for a disc robot. Part (e) shows the path in the projected configuration space that the robot traverses to go to the landmark.

triangle node. Figure 8(a) shows the robot after the first gap split and the robot is chasing the gap that occludes the landmark. Finally, Figure 8(c) shows the robot chasing the landmark. Figure 8(e) shows the shortest path in the configuration space that the robot traverses to navigate to the landmark. This path was computed based in the information obtained by the robot sensors and using the automaton *M* from Section 3.

8 Conclusions

In this paper we have extended the GNT approach in [18] to a disc-shaped differential-drive robot. The robot is equipped with simple sensors and it is unable to build precise geometric maps or localize itself in a global Euclidean frame. This problem is considerably more challenging than in the case of a point robot because visibility information does not correspond exactly to collision free paths in the configurations space. Consequently, the robot must execute detours from the bitangents in the workspace. Indeed, 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 simple sensor feedback and then proved that the motion strategy yields globally optimal motions in the sense of Euclidean distance by characterizing all possible trajectories in terms of sequences of states visited in a finite state machine. Even if precise distance comparisons are not possible, the motion strategy is simple and effective in a broader setting. Important directions for future work include multiply connected environments, disconnected configuration spaces, and bounds with respect to optimality for the cases in which all sensing conditions are not met.

Acknowledgments: 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.

References

1. J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *Proceedings IEEE Symposium on Foundations of Computer Science*, pages 49–60, 1987.
2. D. Z. Chen and H. Wang. Paths among curved obstacles in the plane. In *Proceedings of Computing Research Repository*, 2011.
3. L. P. Chew. Planning the shortest path for a disc in $O(n^2 logn)$ time. In *Proceedings ACM Symposium on Computational Geometry*, 1985.
4. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications, 2nd Ed.* Springer-Verlag, Berlin, 2000.
5. H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: Part I. *IEEE Robotics and Automation Magazine*, 13(2):99–110, 2006.
6. S. K. Ghosh. *Visibility Algorithms in the Plane*. Cambridge University Press, Cambridge, U.K., 2007.
7. S. K. Ghosh and D. M. Mount. An output sensitive algorithm for computing visibility graphs. In *Proceedings IEEE Symposium on Foundations of Computer Science*, pages 11–19, 1987.
8. Y. Landa and R. Tsai. Visibility of point clouds and exploratory path planning in unknown environments. *Communications in Mathematical Sciences*, 6(4):881–913, December 2008.
9. S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Also available at http://planning.cs.uiuc.edu/.
10. S. M. LaValle. Sensing and filtering: A fresh perspective based on preimages and information spaces. *Foundations and Trends in Robotics Series*, 2012. Now Publishers, Delft, The Netherlands.
11. Yun-Hui Liu and S. Arimoto. Finding the shortest path of a disc among polygonal obstacles using a radius-independent graph. *Robotics and Automation, IEEE Trans on*, 11(5):682 –691, oct 1995.
12. R. Lopez-Padilla, R. Murrieta-Cid, and S.M. LaValle. Detours for optimal navigation with a disc robot. In *avaliable on line at:* http://www.cimat.mx/%7Emurrieta/Papersonline/AppendixWAFR12.pdf, pages 1-21, Feb 2012.
13. J. S. B. Mitchell. Shortest paths and networks. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 607–641. Chapman and Hall/CRC Press, New York, 2004.
14. J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem. *Journal of the ACM*, 38:18–73, 1991.
15. L. Murphy and P. Newman. Using incomplete online metric maps for topological exploration with the gap navigation tree. In *Proc. IEEE Int. Conf. on Robotics & Automation*, 2008.
16. 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, *Algorithmic and Computational Robotics: New Directions*, pages 191–203. A.K. Peters, Wellesley, MA, 2001.
17. S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, 2005.
18. B. Tovar, R Murrieta, and S. M. LaValle. Distance-optimal navigation in an unknown environment without sensing distances. *IEEE Trans on Robotics*, 23(3):506–518, June 2007.