Detours for Optimal Navigation with a Disc Robot

Rigoberto Lopez-Padilla[1)], Rafael Murrieta-Cid[1)] and Steven M. LaValle[2)]

1) Centro de Investigación en Matemáticas CIMAT Guanajuato México {rigolpz, murrieta}@cimat.mx

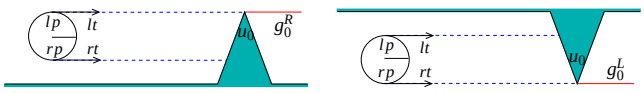
2) University of Illinois at Urbana-Champaign Urbana, IL 61801 USA lavalle@uiuc.edu

This document is an appendix of [1]. Here, we prove that the the detection of a blocked path toward a gap is correct, and that the new gap selected as sub-goal is the gap that must be chased to obtain locally optimal navigation.

1 Landmark Encoding and Blockage Detection

In order to chase a gap, a disc robot first aligns either lt or rt with the vertex that generates the gap and then the robot travels in straight line until the robot touches that vertex either with lp or rp. For chasing a right gap, the robot aligns direction rt with the vertex that generates this gap, this vertex is called right vertex. Symmetrically, for a left gap, the robot aligns direction lt to the vertex that generates that gap, this vertex is called left vertex.

Remark 1. Refer to Figure 1. Notice that if the robot aligns lt to a vertex that generates a right gap or rt to a vertex that generates a left gap then a straight line robot path toward that vertex is not collision free.



(a) Preferential direction lt aligned to a vertex that generates a right gap called a right vertex

(b) Preferential direction rt aligned to a vertex that generates a left gap called a left vertex

Fig. 1 Wrong alignments

Remark 2. The robot always chases gaps with the omni-directional sensor. It might happen that either *lt* or *rt* is aligned to the vertex that generates the gap to be chased, but the omni-directional sensor is in the opposite extremal robot point, if that happens the turret swap the omni-directional sensor in order to chase the gap with the omni-directional sensor.

A straight line path from the current robot configuration toward the vertex might be blocked, that is, a straight line collision free path does not exist from the current robot position to the vertex.

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.

Whenever the path is blocked, the robot executes a detour, that is, the robot travels in straight line toward another vertex before reaching the vertex associated to the gap been chased. 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.

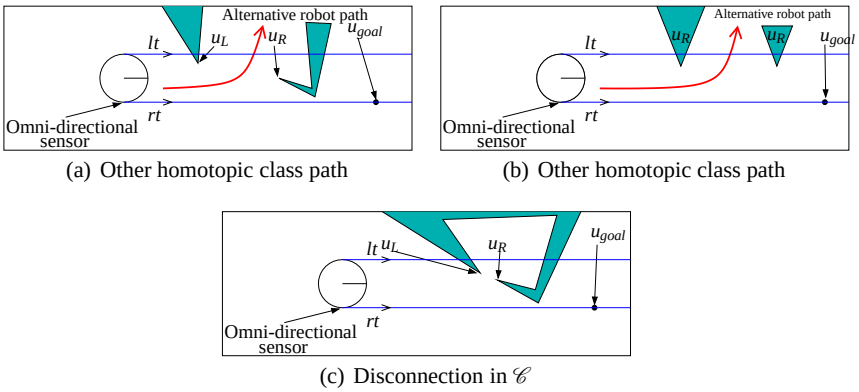
In this work, we have assumed that the configuration space $\mathscr{C}$ is simply connected. In our paper we mainly focus in the navigation task to reach a static disc-shaped landmark Λ in E with the same radius as the robot. Prior to navigation, the robot explores the environment to build the GNT. In the exploration phase the landmark is encoded in one of the nodes. The following remark precise the landmark codification in the GNT during the exploration phase.

Remark 3. A landmark Λ is said to be *recognized* if the landmark is at least partially visible from the location of the omni-directional gap sensor. A landmark Λ is encoded in the GNT at the moment at which at least one point of the landmark is occluded from the location of the omni-directional gap sensor. A landmark is associated with the gap originated by the vertex that occluded at least partially the landmark.

In the navigation task, whenever the landmark is not totally visible from the omni-directional sensor location, the goal given to the robot is a gap (which corresponds to a node in the GNT, which encodes the landmark or is a node belonging to path in the GNT to such node). We call such a gap, the goal gap.

It is important to mention that the correct execution of the navigation phase depend on a correct encoding of the landmark with a gap during the prior exploration phase. As mentioned above, in this work, we assume that $\mathscr{C}$ is simply connected, hence the cases shown in Figure 2 cannot happen. Cases shown in Figures 2(a) and 2(b) do not happen because $\mathscr{C}$ is multiple connected. Case shown in Figure 2(c) does not happen because $\mathscr{C}$ is disconnected.

Figures 3(a), 3(b), 3(c) and 3(d) show cases, in which the gaps generated by the vertices shown in the figures do not encode a landmark. Figure 3(a) shows a case in which during the exploration phase the robot has chased gaps generated by the vertices u_{goal} and u_L and the landmark was not detected.



(a) Other homotopic class path

(b) Other homotopic class path

(c) Disconnection in $\mathscr{C}$

Fig. 2 Cases a) b) and c) do not happen for the assumption of a simply connected component for $\mathscr{C}$. Cases a) and b) do not happen because $\mathscr{C}$ is multiple connected. Case c) does not happen because $\mathscr{C}$ is disconnected.

Notice that some vertices cannot occlude a landmark from the current robot configuration. Figures 3(b), 3(c) and 3(d) show cases in which a landmark cannot be occluded at least partially by the vertex called u_{goal}. Figures 3(b) and 3(c) show cases in which the landmark cannot be occluded at least partially from the current robot configuration by vertex u_{goal}. However, vertices u_{goal} can occlude a landmark at least partially from *some* robot configurations. In contrast, Figures 3(d) shows a case in which a landmark cannot be occluded at least partially by the vertex called u_{goal} from *any* robot configuration. Notice that a landmark cannot be occluded by this vertex, because there is not enough free space on the neighborhood of that vertex to place a landmark.

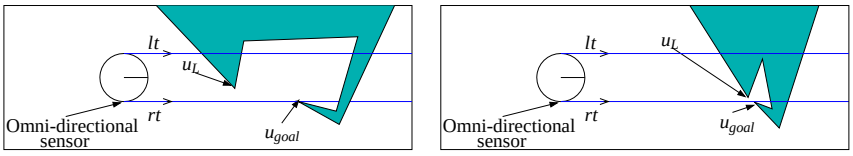
During the exploration phase, the robot chases gaps. Notice that if the vertex that generates a chased gap can encode a landmark (it can occlude partially such landmark) then the robot shall be able to reach the vertex and touch it either with rp or lp. Otherwise, the selected vertex cannot encode a landmark from the current robot configuration.

In order to more formally characterize vertices that can encode a landmark, let define a reachable region related to a given right vertex.

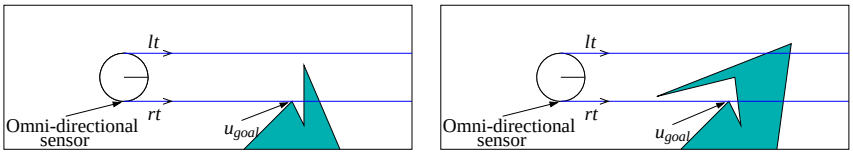
Definition 1. Refer to Figure 3(e), a reachable region is a circle with the robot radius. The circle delimiting the reachable region is touching the right vertex; there is a bitangent line between point rp and the right vertex and another bitangent line between point lp and a point on the reachable region placed diametrically opposed to the right vertex u_O.

There is an analogous reachable region for a given left vertex.

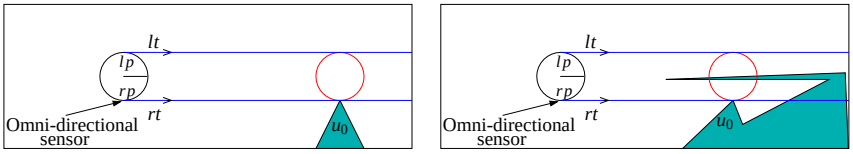
Remark 4. Refer to Figure 3(e). For a simply connected configuration space $\mathscr{C}$ and a no blocked robot path toward a goal vertex u_{goal}. If there is any obstacle (a segment or a vertex) inside the reachable region delimited by the red circle shown in Figure



(a) Dead-end path (b) No encoding the landmark



(c) No encoding the landmark (d) No encoding the landmark



(e) Reachable region (f) u_0, a vertex that is never given as a goal vertex

Fig. 3 Vertices that cannot occlude a landmark

3(e) then this vertex cannot be reached by the robot by traveling a straight line path from its current configuration toward u_0 and touch u_0 (which is the goal vertex) with the point at which the omni-directional sensor is placed (either rp or lp). Consequently, for path continuity, a landmark with the same shape and size of the robot and located beyond the reachable region cannot be reached by the robot. Also from the initial robot configuration the landmark cannot be occluded partially or totally by u_0 given that there exists a clear line of sight between the omni-directional sensor placed at point rp and u_0, which delimits the area visible from the omni-direction sensor, but the landmark cannot be placed beyond that line, such the landmark is partially or totally occluded from the omni-directional sensor initial position.

The exploration phase is executed following the same procedure proposed for the GNT for a point robot. That is, the robot chases gaps at random until all the gaps are primitive gaps. For no blocked paths toward the goal gap the robot chases a goal gap traveling a straight line path from its current configuration to the vertex that generates the goal gap. Whenever the path toward a goal gap is blocked an optimal detour is executed. In section 2, we will describe the method to carry out the optimal detour.

Now, we present a lemma that guarantees that the landmark is correctly detected and encoded in the GNT for a disc robot chasing gaps at random.

Lemma A. *A landmark is correctly detected and encoded in the GNT executing the following procedure during the exploration phase.*

1. *Gaps are chased selecting them at random.*
2. *For a no-blocked path toward a goal gap and a simply connected configuration space $\mathscr{C}$, robot executes a straight line path toward the gap being chased (a goal gap). If the goal gap is a right gap then rt is aligned with the vertex that generates the right gap and the omni-directional sensor is placed at rp. If the goal gap is a left gap then lt is aligned with the vertex that generates the left gap and the omni-directional sensor is placed at lp.*
3. *If traveling a straight line path, robot is not able to reach the vertex that generates a goal gap and touch this vertex either with point rp or lp then the robot will collide with an obstacle. If a collision is detected the goal gap is marked as a primitive gap.*
4. *A landmark is encoded in a gap (corresponding to a node in the GNT) at the moment at which the landmark is partially occluded by the vertex generating the gap.*
5. *Whenever the path toward a goal gap is blocked an optimal detour is executed.*

Proof. To prove this lemma, we show that executing the procedure listed in the lemma, all gaps are eventually marked as primitives and that the omni-directional sensor senses all portions of the environment able to encode a landmark. 1) First, the exploration phase for a point robot is guaranteed to terminate chasing gaps at random, all gaps will be marked primitives gaps (See lemma 3 in [2]). In [2], since the robot is a point, all vertices are reachable. For a disc-shaped robot some vertex u_{goal} that generates a goal gap might not be reached for the disc robot traveling a straight line no-blocked path, from the robot configuration that aligns either rt or lt with u_{goal}. A vertex that cannot be reached by the robot is called no-reachable vertex. Symmetrically, a vertex that can be reached by the robot is called reachable vertex. Consequently, for a disc-shaped robot some gaps might never disappear. From remark 4 if the robot cannot touch u_{goal} either with rp or lp then a landmark cannot be at least partially occluded by u_{goal}, from the omni-directional sensor location at the moment the collision occurs, and the gap can be marked as a primitive gap even if it does not disappear. Hence, if a collision is detected (robot touches ∂E with a point different to rp or lp) then the chased gap is marked as a primitive gap even if it does not disappear. Therefore, all gaps will be marked as primitive gaps. 2) For a point robot, the exploration phase guarantees to see the whole environment with the omni-directional sensor [2]. A landmark is detected by the disc robot whenever the omni-directional sensor is able to see a disc-shaped landmark at least partially. From [2], it is guaranteed that a point omni-directional sensor is able to see every portion of the environment that at some moment was occluded by a reachable vertex. For a disc robot, a no-reachable vertex cannot totally occlude a disc landmark when the robot chases the corresponding gap, hence the landmark is detected surely. 3) The landmark is encoded in a node in the GNT when the landmark is occluded by an obstacle. In order to show that a landmark is correctly encoded in the GNT only two cases need to be considered. a) There exists a vertex that has occluded the landmark at least partially, then landmark is encoded in the GNT. b) Landmark is totally visible from the omni-directional sensor location at the moment all gaps are marked as primitive gaps. Finally, if a blockage is detected toward a g_{goal} then the

robot executes a detour with guarantees that the vertex will be reached or g_{goal} is a no-reachable vertex, the result follows.

Lemma B. *For detecting a blockage toward a right gap g^R distance measures d_L and ${d_R}^t$ obtained respectively along preferential directions lt and rt are sufficient to detect blockages or declare the robot path collision free.*

Proof. For a simply connected configuration space $\mathscr{C}$. Given that in the exploration phase only vertices that can occlude a landmark will be given as a goal to the robot then there is not obstacle inside the reachable region associated to the vertex that generates the right goal gap, and the obstacle that blocks the path toward the right goal gap must cross the line along preferential direction lt from a current robot position. Therefore distance measures d_L and ${d_R}^t$ obtained along preferential directions rt or lt are able to detect obstacles that block the path toward the goal gap or declare the path to reach the vertex that generates the goal gap collision free.

There is an analogous lemma for detecting a blockage toward a left gap g^L or to declare the robot path collision free.

Refer to Figure 3(f), this figure shows a case in which the vertex being chased cannot occlude a landmark, hence during the navigation phase this vertex is never given as a goal to the robot, therefore blockage detection using distance measures along preferential direction rt and lt (which are not sufficient to detect robot collision) are not required, since the vertex is never chased.

2 Optimal Detour

The main complication is to determine which is the vertex that must be reached by the robot to obtain an optimal detour (in the sense of Euclidean distance), or equivalent which is the new gap to be chased. Our approach to choose the new vertex (or equivalent a new gap) yielding the optimal detour is based on the main following definitions and remarks.

Remark 5. For lt aligned to a left vertex or rt to a right vertex. There is a bitangent line either between lp and the vertex associated to the gap to be chased or rp and that vertex. There might be other vertices that generate left or right gaps, but notice that those vertices do not cross the bitangent segment between either lp or rp and the vertex, otherwise the bitangent segment does not exist.

Remark 6. A robot clockwise rotation can be executed either w.r.t rp or w.r.t. the robot center (a clockwise rotation in place). Similarly, a robot counterclockwise rotation can be executed either w.r.t. lp or w.r.t. the robot center (a counterclockwise rotation in place).

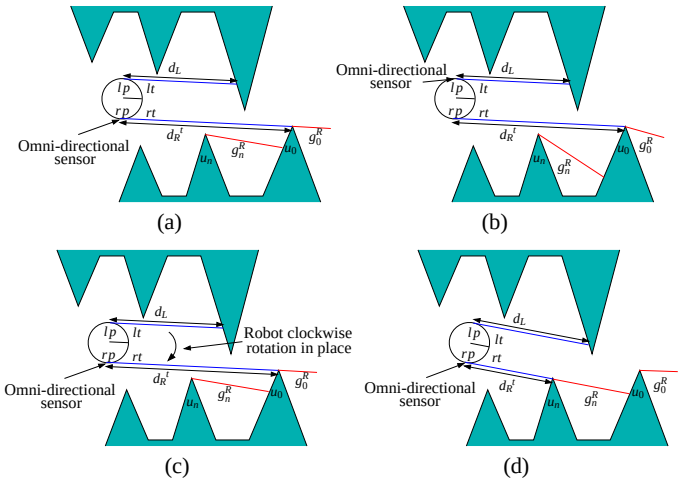


Fig. 4 A u_n vertex. a) blockage detection, b) turret motion and sub-goal selection, a u_n vertex, c) turret moves omni-directional sensor to rp to execute a rotation in place based on feedback information. d) The robot starts a straight line motion toward u_n

Remark 7. The selection of the gap (or equivalent a vertex) that corresponds to an optimal detour depends on the robot sense of rotation. We call these vertices u_p and u_n. First, we intuitively describe u_n and u_p vertices for a clockwise and counterclockwise rotation. Either u_n or u_p corresponds to the vertex associated to the optimal detour. Later a more formal definition is provided.

For a clockwise rotation, next vertex u_n is the first vertex in clockwise order after the original goal vertex, which is aligned with rt and is reachable by the robot traveling a straight line path. Previous vertex u_p is the last vertex in clockwise order before the original goal vertex, which is aligned with lt and is reachable by the robot traveling a straight line path.

For a counterclockwise rotation, next vertex u_n is the first vertex in counterclockwise order after the original goal vertex, which is aligned with lt and is reachable by the robot traveling a straight line path. Previous vertex u_p is the last vertex in counterclockwise order before the original goal vertex, which is aligned with rt and is reachable by the robot traveling a straight line path.

Here, we describe the selection of both u_n and u_p vertices considering the case in which the robot is not touching the boundary of the environment ∂E. Later, we point out some extra considerations that must be taken into account for the case, in which the robot is touching the boundary of the environment ∂E.

Figure 4 shows an example of the blockage detection and the selection of the new gap. Figure 4 shows the case of a u_n vertex. Figure 4(a) shows the moment rt is aligned to the goal gap (a right gap) g_0^R, since ${d_R}^t > d_L$ then the path to the vertex that generates the goal gap g_0^R is blocked. First, the turret swap the omni-directional sensor to lp and the laser pointer to rp, see Figure 4(b). Figure 4(c) illustrate a turret

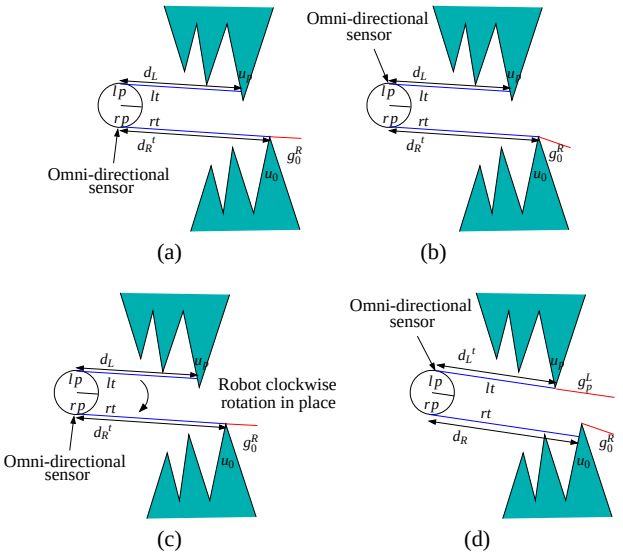


Fig. 5 A u_p vertex. a) blockage detection, b) turret motion and sub-goal selection, a u_p vertex, c) turret moves omni-directional sensor to rp to execute a rotation in place based on feedback information., d) d) turret moves omni-directional to lp to chase the left gap and robot starts a straight line motion toward u_p.

motion to bring the omni-directional sensor to rp in order to execute a rotation in place based on feedback information. Finally, robot chase the gap g_n^R, see Figure 4(d).

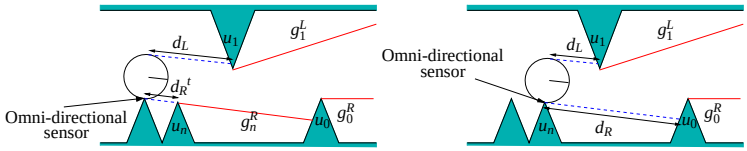
Remark 8. If a blockage is detected then the turret always swap the omni-directional sensor to the opposite extremal robot point to consider all vertices (right and left) that might represent an optimal detour (see Section 2.2).

Figure 5 shows the case of a u_p vertex selection. Figure 5(a) shows the moment rt is aligned to the goal gap (a right gap) g_0^R, since ${d_R}^t > d_L$ then the path to the vertex that generates the goal gap g_0^R is blocked. First, the turret swap the omni-directional sensor to lp and the laser pointer to rp, see Figure 5(b). Figure 5(c) illustrate a turret motion to bring the omni-directional sensor to rp in order to execute a rotation in place based on feedback information. Finally, the turret moves the omni-directional sensor to point lp to chase the gap g_p^L, see Figure 4(d).

Remark 9. For an optimal detour, only one either u_p or u_n can exist. If u_p exists then it will be the first vertex to be visited by the robot to obtain an optimal path toward the original goal gap. Symmetrically, if u_n exists then it will be the first vertex to be visited by the robot to obtain an optimal path toward the original goal gap.

The following remark points out some extra considerations that must be taken into account for the case, in which the robot is touching the boundary of the environment ∂E.

Remark 10. It is important to notice that when the robot reaches a given vertex coming from a detour, the robot might not be aligned to original goal gap. It might happen that the robot cannot align either *rt* or *lt* to the original goal gap, because this motion will produce an unnecessary robot translation, and the global optimality would be lost. For an example of this situation see Figure 6. In Figure 6(b) robot reaches a u_n vertex, but it cannot align *rt* with the vertex that generates the original right goal gap g_0^R, because this motion will produce to lose global path optimality, the robot would translate unnecessary. Notice that, from that robot configuration, it is needed to detect a vertex u_p or u_n. Consequently, the detection of u_p or u_n vertices needs to be performed whether or not *lt* or *rt* is aligned to a vertex generating a goal gap.



(a) robot detects a u_n vertex and it moves toward it.

(b) robot reaches a u_n vertex, it cannot align *rt* with the vertex that generates the original right goal gap g_0^R .

Fig. 6 Robot reaches a vertex after having executed a detour, robot cannot align *rt* with the right goal vertex, because path optimality will be lost

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, 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, this angle is called θ_L.

To compute these distances and angles, we locate the vertices in *local reference frames*. Either a reference frame defined by point rp and a right goal vertex or a reference frame defined by point lp and a left goal vertex (See Sections 2.1 and 2.3).

To find a vertex u_n or a vertex u_p, we use two orders. 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. Indeed, these definitions help to obtain an algorithm to determine u_n or u_p. Recall that u_n and u_p depend on the robot sense of rotation. The following definitions consider both cases, whether the robot executes a rotation in place or whether the robot executes a rotation with

respect to lp (a counterclockwise rotation) or a rotation with respect to rp (a clockwise rotation).

Definition 2. For a clockwise robot rotation, a given vertex u that generates a right gap, is a u_n vertex if and only if u is the *first* vertex in clockwise angular order of the vertices that generate a right gap enjoying the next properties. It corresponds to the optimal detour, and the distance ${d_L}^t$ related to the *next vertex* in clockwise angular order that generates a left gap is larger (posterior in the order of distance) than the distance ${d_R}^t$ related to the vertex u in question that generates a right gap. Refer to Figure 9(a) and Tables 1.

Definition 3. For a clockwise rotation, a given vertex u that generates a left gap is a u_p vertex if and only if u is the *last* vertex in clockwise angular order of the vertices that generate a left gap enjoying the next properties. It corresponds to the optimal detour, and the distance ${d_R}^t$ related to the *previous vertex* in clockwise angular order that generates a right gap is larger (posterior in the order of distance) than the distance ${d_L}^t$ related to the vertex u in question that generates a left gap. Refer to Figure 9(b) and Table 2.

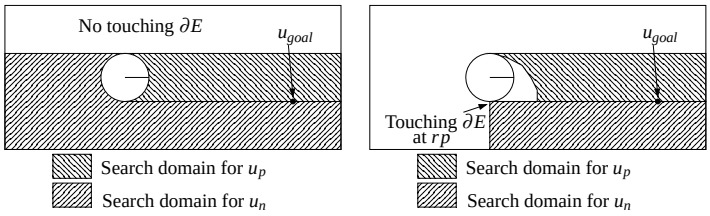
Remark 11. There are two analogous definitions of u_n and u_p, for a counterclockwise rotation.

As mentioned above, key to our approach is to determine u_p and u_n vertices, which correspond to the optimal detour toward the goal gap given by the GNT. Figure 7(a) shows the regions in the space (called search domains), in which a u_p and u_n vertices can be located, for the case in which the robot is not touching ∂E and the goal gap is a right gap. Notice that in such case the robot might need to rotate in place clockwise an angle within the interval defined by $[0, 2\pi)$, in order to align either rt with a vertex generating a right gap corresponding to a u_n vertex, or lt with a vertex generating a left gap corresponding to a u_p vertex.

Figure 7(b) shows the search domains, in which a u_p and u_n vertices can be located, for the case in which the robot is touching ∂E at point rp and the goal gap is a right gap. In such case the robot might need to rotate clockwise with respect to rp an angle within the interval defined by $[0, \frac{2}{\pi})$, in order to align either rt with a vertex generating a right gap corresponding to a u_n vertex, or lt with a vertex generating a left gap corresponding to a u_p vertex.

2.1 Robot is touching ∂E

The planning step corresponds to the determination of the optimal detour (equivalent to determine a u_p vertex or a u_n vertex). Here, we provide equations to calculate angles and distances needed to determine u_p and u_n vertices.



(a) Case in which the robot is not touching ∂E

(b) Case in which the robot is touching ∂E at point rp

Fig. 7 Regions in the space (called search domains), in which a u_p and u_n vertices can be located

2.1.1 Planning: a u_p vertex or a u_n vertex

To determine u_p and u_n vertices, it is necessary to compute distances ${d_R}^t$ and ${d_L}^t$, and angles θ_R and θ_L.

To compute these distances and angles, we locate the vertices in *local reference frames*. Either a reference frame defined by point rp and a right goal vertex or a reference frame defined by point lp and a left goal vertex.

Figure 8(a) shows, the location of left vertex, coordinates (x_{uL}, y_{uL}), over a local reference frame defined by point rp and u_0. The location of the vertex is computed based on distance d_u and the angle θ_0 measured w.r.t y axis of the local reference frame, and an angle β_L measured w.r.t preferential direction lt. Notice that, θ_O represents an offset angle, this offset angle is needed because there are cases, in which the robot cannot aligned rt with the right goal vertex (see Figure 6(b)). Hence, it is needed to determine the location of the vertices, even in the case, in which rt (or lt) is not aligned with the goal right vertex (or goal left vertex).

Equation 1 indicates the coordinates of left vertices in a reference frame defined by the point rp and the right goal vertex u_{goal} based on distance d_u and preferential direction lt (angle β_L). See Figure 8(a).

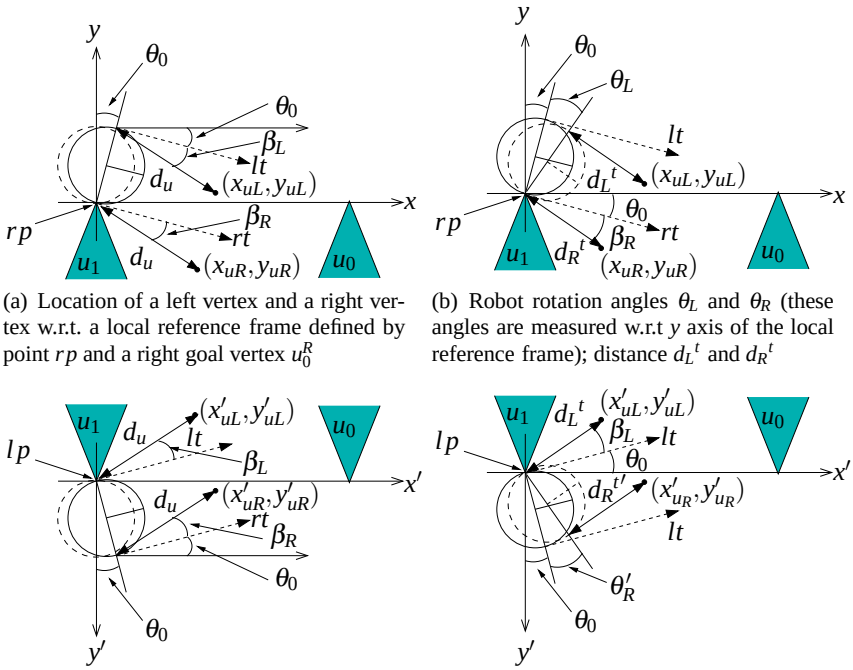
$$\begin{aligned} x_{uL} &= 2r\sin\theta_0 + d_u\cos(\theta_0+\beta_L) \\ y_{uL} &= 2r\cos\theta_0 - d_u\sin(\theta_0+\beta_L) \end{aligned} \tag{1}$$

The coordinates of right vertices are given by Equation 2 (see Figure 8(a)).

$$\begin{aligned} x_{uR} &= d_u\cos(\theta_0+\beta_R) \\ y_{uR} &= d_u\sin(\theta_0+\beta_R) \end{aligned} \tag{2}$$

There are equivalent equations to indicate the coordinates of right and left vertices over a reference frame defined by point lp and a left goal vertex, see Figure 8(c).

Equation 3 calculates the robot rotation angle with respect to the y axis of the local referent frame; equivalent to the angle needed to align preferential direction lt to a left vertex executing a robot clockwise rotation w.r.t. point rp (see Figure 8(b)),



(a) Location of a left vertex and a right vertex w.r.t. a local reference frame defined by point rp and a right goal vertex u_0^R

(b) Robot rotation angles θ_L and θ_R (these angles are measured w.r.t y axis of the local reference frame); distance ${d_L}^t$ and ${d_R}^t$

(c) Location of a right vertex and a left vertex w.r.t. a local reference frame defined by point lp and a right goal vertex u_0^R

(d) Robot rotation angles ${\theta_R}'$ and ${\theta_L}'$ (these angles are measured w.r.t y' axis of the local reference frame); distance ${d_R}^{t'}$ and ${d_L}^{t'}$

Fig. 8 Local reference frames for robot touching ∂E

$$\theta_L = \arcsin\left(\frac{2rx_{uL} - y_{uL}\sqrt{{x_{uL}}^2 + {y_{uL}}^2 - 4r^2}}{{x_{uL}}^2 + {y_{uL}}^2}\right) - \theta_0 \tag{3}$$

Equation 4 calculates distance ${d_L}^t$. Figure 8(b) shows distance ${d_L}^t$, which is measured along preferential direction lt from point lp to a left vertex. This distance optains when robot has executed a clockwise robot rotation w.r.t. point rp to align lt to a left vertex.

$${d_L}^t = \sqrt{[x_{uL} - 2r\sin(\theta_L + \theta_0)]^2 + [y_{uL} - 2r\cos(\theta_L + \theta_0)]^2} \tag{4}$$

Equation 5 indicates the value of ${d_R}^t$, which is measured directly by the omni-directional sensor (see Figure 8(b)).

$${d_R}^t = d_u \tag{5}$$

Equation 6 indicates the value of θ_R, which is also measured directly by the omni-directional sensor (see Figure 8(b)).

$$\theta_R = \beta_R \tag{6}$$

There are totally analogous equations to compute angles $\theta_R{}'$ $\theta_L{}'$, ${d_R}^{t'}$ and ${d_L}^{t'}$ over a local reference frame defined by point lp and left goal vertex. Refer to Figure 8(d).

2.2 Algorithm to find a u_p or a u_n vertex

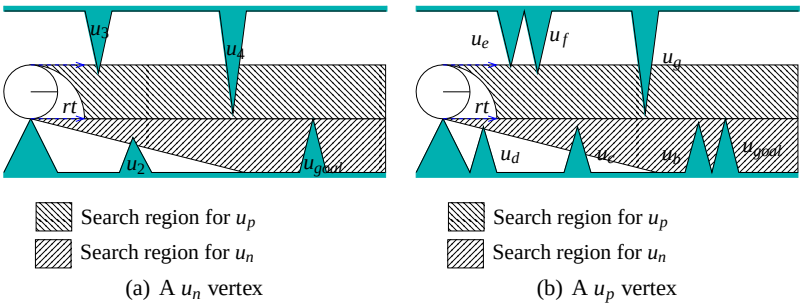
Angles θ_R and θ_L and distances ${d_R}^t$ and ${d_L}^t$ are used to find vertices u_p and u_n.

Algorithm 1 a u_p vertex or a u_n vertex

1) The algorithm starts from the goal vertex.
There are two cases: Case I) corresponds to a right goal vertex and case II) to a left goal vertex.
2) Case I)
3) 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 θ_L larger than the angle θ_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}.
4) Selection of a left goal vertex
The left vertex with largest θ_L, the last in the angular order is selected as a new goal vertex.
5) Case II)
6) 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 θ_R smaller than the angle θ_L related to the goal vertex, and a distance ${d_R}^t$ smaller than distance ${d_R}^t$ related to the left goal vertex u^L_{goal}.
7) Selection of a right goal vertex
The right vertex with smallest angle θ_R, the first in the angular order is selected as a new goal vertex.
8) This procedure is repeated until the vertex selected as new goal is not blocked.

Table 1 shows an example of the execution of algorthim 1 and the determination of a u_n 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 vertices, 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_2 is a u_n vertex corresponding to the optimal detour.

Table 2 shows another example of the execution of algorthim 1 and the selection of a u_p vertex, u_f is a u_p vertex corresponding to the optimal detour.



(a) A u_n vertex (b) A u_p vertex

Fig. 9 Examples of a u_n vertex and a u_p vertex

Table 1 Orders for selecting a u_n vertex

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}
$\rightarrow$	$\uparrow$	$\times$		$\otimes$		-		-	$\uparrow$
$\leftarrow$			$\otimes$	$\uparrow$			-	$\uparrow$	+
$\rightarrow$		$\times$	$\uparrow$			-	$\uparrow$	+	

Table 2 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}
$\rightarrow$	$\uparrow$		$\times$		$\times$		$\otimes$			-	-		-		$\uparrow$
$\leftarrow$				$\otimes$		$\times$	$\uparrow$		-			-	$\uparrow$	+	+
$\rightarrow$			$\times$	$\uparrow$	$\otimes$					-	-	$\uparrow$	+		
$\leftarrow$					$\uparrow$	$\times$			-		$\uparrow$	+		+	+

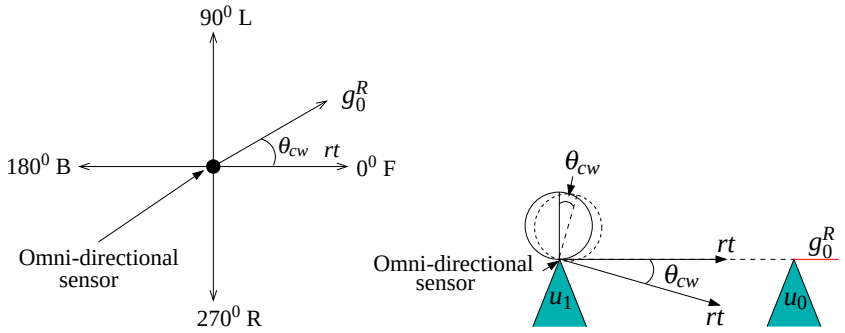
Lemma C. *Algorithm 1 find the optimal detour in the sense of Euclidean distance toward the goal vertex.*

Proof. It is based on two facts. 1) The structure of the path representing a detour is a sequence of sub-paths between vertices. 2) Each element of the sequence is locally optimal since each vertex selected as sub-goal lies on the boundary of the restriction. In other words, each vertex selected as sub-goal is the one, that deforms the path the most, toward the goal vertex. Therefore, the resulting detour is optimal.

2.2.1 Executing a rotation to align lt to a u_p vertex or rt to a u_n vertex

Once the optimal detour is determined, the robot rotates to align lt or rt with the vertex representing the optimal detour. This rotation is executed based on sensing feedback. The sensing feedback corresponds to an angle between preferential direction rt and a right vertex; or an angle between preferential direction lt and a left vertex.

Figure 10 shows an example of an angle between preferential direction rt and a right vertex, for the case of a clockwise robot rotation with respect to point rp. The angle measuring the clockwise robot rotation is called θ_{cw}.



(a) Preferential direction rt (b) Clockwise robot rotation w.r.t point rp

Fig. 10 Example of an angle between preferential direction rt and a right vertex, for the case of a clockwise robot rotation with respect to point rp.

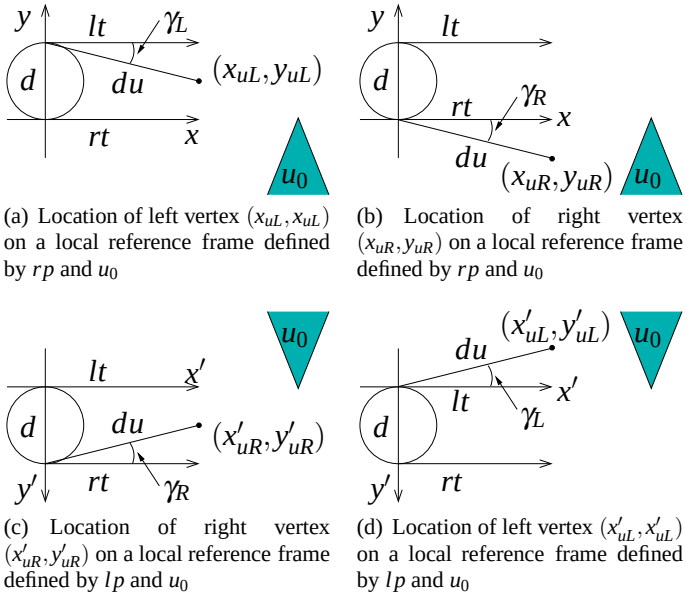
2.3 Robot is not touching ∂E

Here, we provide equations to calculate distances ${d_R}^t$ and ${d_L}^t$, and angles θ_R and θ_L for the case of a clockwise or counterclockwise robot rotation in place.

The planning step (optimal detour determination) is totally analogous to the case of a robot rotation with respect to point rp or lp described in Section 2.1.

2.3.1 A u_p vertex and a u_n vertex

Figures 11(a) and 11(b) show the location of right vertex (x_{uR}, x_{uR}) and left vertex (x_{uL}, x_{uL}) on a local reference frame defined by point rp and a right goal vertex u_0, Figures 11(c) and 11(d) show the location of right vertex (x'_{uR}, x'_{uR}) and left vertex (x'_{uL}, x'_{uL}) on a local reference frame defined by point lp and a right goal vertex u_0.



(a) Location of left vertex (x_{uL}, x_{uL}) on a local reference frame defined by rp and u_0

(b) Location of right vertex (x_{uR}, y_{uR}) on a local reference frame defined by rp and u_0

(c) Location of right vertex (x'_{uR}, y'_{uR}) on a local reference frame defined by lp and u_0

(d) Location of left vertex (x'_{uL}, x'_{uL}) on a local reference frame defined by lp and u_0

Fig. 11 Location of right and left vertices over local reference frames

Equation 7 indicates the coordinates of left vertices in a reference frame defined by the point rp and the right goal vertex u_0 based on distance d_u and angle γ_L, see Figure 11(a). Analogously, equation 8 indicates the coordinates of right vertices in the same reference frame based on distance d_u and angle γ_R, see Figure 11(b).

$$\begin{aligned} x_{uL} &= d_u \cos\gamma_L \\ y_{uL} &= d - d_u \sin\gamma_L \end{aligned} \tag{7}$$

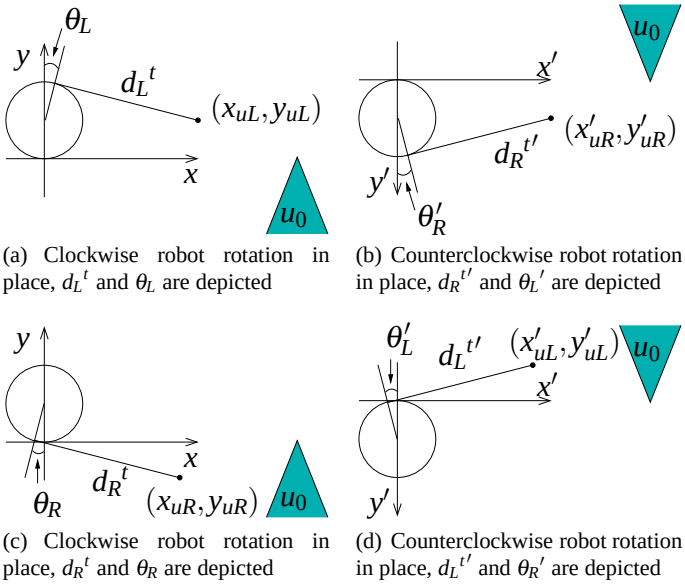
$$\begin{aligned} x_{uR} &= d_u \cos\gamma_R \\ y_{uR} &= d_u \sin\gamma_R \end{aligned} \tag{8}$$

There are other similar equations to compute the coordinates of left a right vertices in the reference frame defined by point lp and a goal vertex. See Figures 11(c) and 11(d).

Figure 12 shows the case of a robot rotation in place. Equation 9 calculates the robot rotation angle with respect to the y axis of the local referent frame; equivalent the angle measured from preferential direction lt to a left vertex, corresponding to a robot clockwise rotation in place (see Figure 12(a)).

$$\theta_L = \arcsin\left(\frac{x_{uL} r + (y_{uL} - r)\sqrt{{x_{uL}}^2 - 2y_{uL} r - {y_{uL}}^2}}{(y_{uL} - r)^2 + {x_{uL}}^2}\right) \tag{9}$$

There is a similar equation to compute the angle with respect to the y' axis of another local reference frame; equivalent to the angle measured from preferential



(a) Clockwise robot rotation in place, $d_L{}^t$ and θ_L are depicted

(b) Counterclockwise robot rotation in place, $d_R{}^{t'}$ and $\theta_L{}'$ are depicted

(c) Clockwise robot rotation in place, $d_R{}^t$ and θ_R are depicted

(d) Counterclockwise robot rotation in place, $d_L{}^{t'}$ and $\theta_R{}'$ are depicted

Fig. 12 Robot rotation in place: Distances and angles of alignment used to find a u_p and a u_n vertex

direction *rt* to a right vertex, corresponding to a robot counterclockwise rotation in place (see Figure 12(b)). Notice that these angles are used to determine a u_p vertex.

Equation 10 calculates distance $d_L{}^t$. Figure 12(a) shows distance $d_L{}^t$, which is computed assuming that preferential direction *lt* is pointing to a left vertex, and corresponding to a robot clockwise rotation in place.

$$d_L{}^t = \sqrt{(x_{uL} - r\sin\theta_L)^2 + (y_{uL} - r - r\cos\theta_L)^2} \tag{10}$$

There is a similar equation to compute $d_R{}^{t'}$. Figure 12(b) shows distance $d_R{}^{t'}$, which is computed assuming that preferential direction *rt* is pointing to a right vertex, and corresponding to a robot counterclockwise rotation in place. These distances are used to determine a u_p vertex.

Equation 11 calculates the robot rotation angle with respect to the y axis of the local referent frame; equivalent to the angle measured from preferential direction *rt* to a right vertex, corresponding to a robot clockwise rotation in place (see Figure 12(c)).

$$\theta_R = \arcsin\left(\frac{-x_{uR}r + (y_{uR} - r)\sqrt{{x_{uR}}^2 - 2y_{uR}r - {y_{uR}}^2}}{(y_{uR} - r)^2 + {x_{uR}}^2}\right) \tag{11}$$

There is a similar equation to compute the angle with respect to the y' axis of another local reference frame; equivalent to the angle measured from preferential

direction lt to a left vertex, corresponding to a robot counterclockwise rotation in place (see Figure 12(d)). These angles are used to determine a u_n vertex.

Equation 12 calculates distance ${d_R}^t$. Figure 12(c) shows distance ${d_R}^t$, which is computed assuming that preferential direction rt is pointing to a right vertex, and corresponding to a robot clockwise rotation in place.

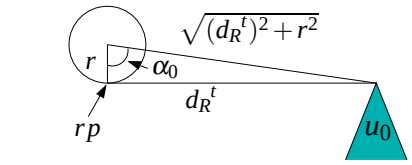
$$ {d_R}^t = \sqrt{(x_{uR} + r\sin\theta_R)^2 + (y_{uR} - r + r\cos\theta_R)^2} \tag{12} $$

Figure 12(d) shows distance ${d_L}^{t'}$, which is computed assuming that preferential direction lt is pointing to a left vertex, and corresponding to a robot counterclockwise rotation in place.

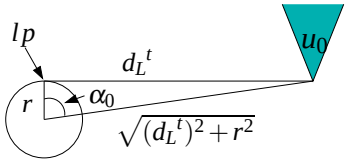
These distances are associated to the determination of a u_n vertex.

2.3.2 Executing a rotation to align lt to a left vertex or rt to a right vertex

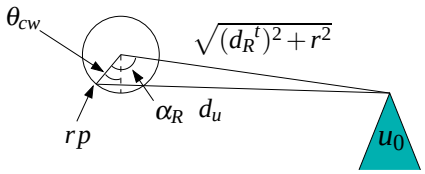
Once the optimal detour is determined, the robot rotates to align lt or rt with the vertex representing the optimal detour. This rotation is executed based on sensing feedback. The sensing feedback corresponds to the distance between point rp or lp and the goal vertex, which can be a right or left vertex. Figure 13 depicts a robot rotation in place when the robot is not touching ∂E.



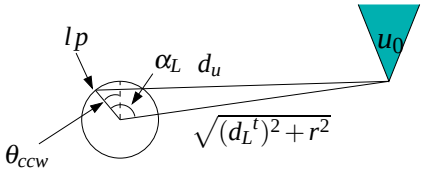
(a) Angle α_0: case of a right goal vertex



(b) Angle α_0: case of a left goal vertex



(c) Angle of rotation θ_{cw}: case of a clockwise robot rotation in place



(d) Angle of rotation θ_{ccw}: case of a counterclockwise robot rotation in place

Fig. 13 Robot is not touching ∂E: Computation of θ_{cw} and θ_{ccw} based on feedback over measurement of d_u

Equation 13 corresponds to a clockwise robot rotation in place, it indicates the rotation angle that the robot must execute to align rt to a right vertex (see Figure 13(c)), or to align lt to a left vertex, both cases are based on feedback distance measurement d_u between the omni-directional sensor and the right goal vertex.

$$\theta_{cw} = \alpha_R - \alpha_0 \tag{13}$$

Equation 14 corresponds to a counterclockwise robot rotation in place, it indicates the rotation angle that the robot must execute to align lt to a left vertex (see Figure 13(d)), or to align rt to a right vertex, again both cases are based on feedback distance measurement d_u between the omni-directional sensor and the left goal vertex.

$$\theta_{ccw} = \alpha_L - \alpha_0 \tag{14}$$

Equation 15 calculates the value of α_R needed to compute a clockwise robot rotation in place.

$$\alpha_R = \arccos\left(\frac{(d_R{}^t)^2 + 2r^2 - d_u{}^2}{2r\sqrt{(d_R{}^t)^2 + r^2}}\right) \tag{15}$$

Equation 16 calculates the value of α_L needed to compute a counterclockwise robot rotation in place.

$$\alpha_L = \arccos\left(\frac{(d_L{}^t)^2 + 2r^2 - d_u{}^2}{2r\sqrt{(d_R{}^t)^2 + r^2}}\right) \tag{16}$$

3 Chasing a Landmark

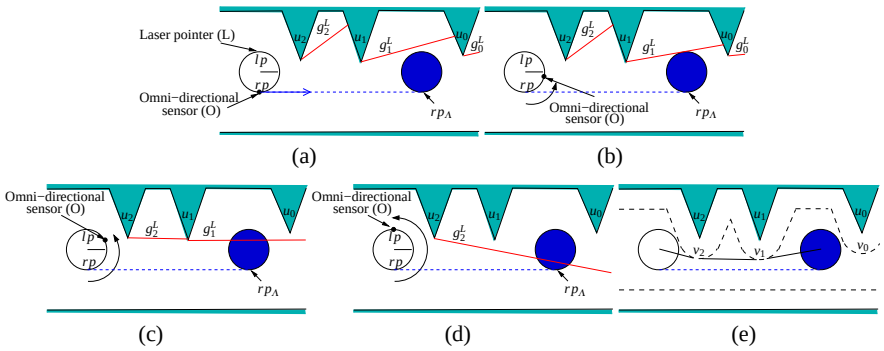


Fig. 14 Case Landmark

Figure 14 shows case of chasing a Landmark. The robot is not touching ∂E, the omni-directional sensor is located at rp, the robot goal is to reach a landmark Λ and the path to reach Λ is blocked.

Let define rp_Λ as the particular point in the boundary of Λ, such that there is a bitangent line between point lp in the robot and point rp_Λ in Λ. The first motion primitive executed by the robot is a clockwise rotation in place until rt is aligned with rp_Λ. See Figure 14(a).

At that robot configuration the procedure to detect the blockage is as follows:

The turret swap the omni-directional sensor from rp to lp, While the turret is rotating counter-clockwise, the gaps change orientation due to the motion of the omni-directional sensor. If one gap touches the boundary of the landmark (the landmark will be, at least partially, occluded by a vertex from this omni-directional sensor position) then a straight line path toward Λ is blocked, and Λ will be associated to this gap and encoded in GNT. See Figure 14(b).

While the turret keeps rotating, the gap associated with Λ might merge with another gap. If the gap associated to Λ merges with other gap then Λ is associated to the gap resulting from the merging of the two gaps. See Figure 14(c).

The turret finishes the motion at the moment the omni-directional sensor reaches point lp. See Figure 14(d). If one blockage is detected then the robot is commanded to chase the gap associated to the Λ.

The path toward vertex that generates the gap to be chased might or might not be blocked. The case of a blocked path to reach a vertex is handled by the selection of a u_n vertex or a u_p vertex associated to the optimal detour to reach the vertex.

Figure 14(e) shows with dotted black lines the configuration space. The optimal robot path is shown with a solid line. This path first visits two vertices and later the landmark.

Lemma D. *The condition that Λ-ALIGN uses to detect a blockage toward a right landmark is correct, and the gap selected as goal is the gap that must be chased to obtain locally optimal navigation.*

Proof. Since, while the turret was rotating counter-clockwise a gap touched the boundary of Λ then a clear line of sight does not exist between lp and lp_Λ. Given that lp and rp are the extremal side robot points in the direction that delimits the area of the robot therefore a straight collision free path between the robot and the landmark does not exist. Hence a left gap must be chased. If a straight line robot path toward the vertex that generates the left gap to be chased exists then this path is locally optimal. Otherwise, the path toward the vertex is blocked. In this last situation the determination of a u_p vertex or a u_n vertex determines the locally optimal path to perform a detour. □

There are other two cases totally analogous to case in which the robot is not touching ∂E to chase a landmark. One corresponds to robot touching the environment at rp and the other to robot touching the environment at lp. The only difference among these two cases and the case in which the robot is not touching E is that robot will rotate either w.r.t lp (when the robot is touching ∂E at lp) or w.r.t rp (when the robot is touching ∂E at rp) to align lt with lp_Λ or rt with rp_Λ.

References

1. R. Lopez-Padilla, R. Murrieta-Cid, and S. M. LaValle. Optimal Gap Navigation for a Disc Robot. Accepted to the Workshop on the Algorithmic Foundations of Robotics, 2012.
2. B. Tovar, R. Murrieta-Cid, and S. M. LaValle. Distance-Optimal Navigation in an Unknown Environment without Sensing Distances. IEEE Transactions on Robotics, Vol. 23, No. 3, pages 506-518, June 2007.