

## Local path planning for mobile robots based on intermediate objectives

Yingchong Ma, Gang Zheng, Wilfrid Perruquetti, Zhaopeng Qiu

► **To cite this version:**

Yingchong Ma, Gang Zheng, Wilfrid Perruquetti, Zhaopeng Qiu. Local path planning for mobile robots based on intermediate objectives. *Robotica*, Cambridge University Press, 2014, pp.1-15. <10.1017/S0263574714000186>. <hal-00960019>

**HAL Id: hal-00960019**

**<https://hal.inria.fr/hal-00960019>**

Submitted on 1 Apr 2014

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Local path planning for mobile robots based on intermediate objectives \*

Yingchong Ma <sup>1</sup>, Gang Zheng <sup>2</sup>, Wilfrid Perruquetti <sup>1,2</sup> and Zhaopeng Qiu <sup>1</sup>

1. LAGIS CNRS UMR 8219, Ecole Centrale de Lille, BP 48, 59651 Villeneuve d'Ascq, France.

2. Non-A team, INRIA - Lille Nord Europe, 40 avenue Halley, 59650 Villeneuve d'Ascq, France.

(email: yingchong.ma@ec-lille.fr, gang.zheng@inria.fr, wilfrid.perruquetti@inria.fr, zhaopeng.qiu@ec-lille.fr)

## Abstract

*This paper presents a path planning algorithm for autonomous navigation of non-holonomic mobile robots in complex environments. The irregular contour of obstacles is represented by segments. The goal of the robot is to move towards a known target while avoiding obstacles. The velocity constraints, robot kinematic model and non-holonomic constraint are considered in the problem. The optimal path planning problem is formulated as a constrained receding horizon planning problem and the trajectory is obtained by solving an optimal control problem with constraints. Local minima are avoided by choosing intermediate objectives based on the real time environment.*

---

\*This work was supported by EU INTERREG IVA 2 Mers Seas Zeeen Cross-border Cooperation Programme under SYSIASS project 06-020. It was also supported by Ministry of Higher Education and Research Nord-Pas de Calais Regional Council and FEDER through the "Contrat de Projets Etat Region (CPER) CIA 2007-2013".

# 1 Introduction

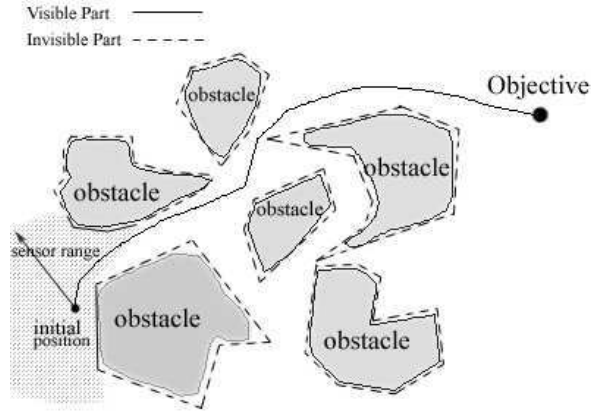
Autonomous navigation is an important issue in robotics research. This problem is of theoretically interesting properties and of practical importance. Navigation is a task that an autonomous robot must do correctly in order to move safely from one location to another without getting lost or colliding with other objects [1]. Three general problems are involved in navigation: localization, path planning and trajectory tracking. In these problems, path planning is quite important since it enables the selection and the identification of a suitable path for robots to traverse in the environment.

When the environment is completely known before the robot moves, a collision free trajectory with lowest cost from the starting point to the target can be obtained by global path planning algorithms, the cost can be defined as the traveled distance, the expended energy, time exposed to danger, etc. In such cases, the complete information of the environment is obtained in static environment, and collision free paths are selected and planned off-line. Different kinds of approaches have been proposed, such as cell decomposition [2, 3], visibility graph [4, 5, 6], retraction [7], heuristic-based algorithms [8, 9], genetic algorithms [10, 11], and projection [12] etc. A well-known algorithm of global heuristics search is  $A^*$  [13], which can find the shortest collision free path through a fully mapped environment by using a priority queue.  $D^*$  search [14] is an extension of  $A^*$  algorithm, and has been used in many applications [15]. It can modify the planned path dynamically if unknown obstacles are encountered. When a robot has only partial knowledge about the environment before it starts, the robot has to plan the path locally with the information captured by the sensor equipped on the robot [16]. The Bug1 and Bug2 algorithms [17] are among the earliest and simplest sensor-based path planning algorithms, and the algorithms are

based on the boundary following method. Another famous algorithm is the artificial potential field approach (APF) proposed in [18]. One of the main drawbacks of APF is local minima when the composition of all forces on the robot equals to zero. Some extended algorithms based on APF have been proposed [19, 20].

When considering the path planning problem for unicycle-like mobile robots, the physical limitations and kinematic constraints have to be taken into account. Due to those constraints, the algorithms mentioned above cannot be applied to this type of mobile robots, since these constraints might render many unfeasible spatial paths. Some algorithms have been proposed for this kind of robot [21, 22, 23], and the algorithm proposed in [24] describes the path planning problem as a nonlinear optimization problem with constraints, which guarantees the navigation of the robot in unknown environments. However, they either compute the path not in an optimal way, or simply represent the obstacles as circles, and there are at least two drawbacks for these algorithms: firstly only the circular obstacles are taken into account, and secondly local minima cannot be avoided when robots getting close to complex obstacles. Therefore, this algorithm is not suitable for complex environments with different shapes of obstacles. An extended algorithm is proposed in [25] based on the Tangent Bug algorithm [26] to treat this problem by following the obstacle boundary, which however involves unnecessary detours along obstacle boundaries and leads to non optimal trajectories.

In this paper, the irregular contours of obstacles are represented by segments. The path planning problem for unicycle-like mobile robots is described as an optimal control problem by involving all physical constraints. Local minima are avoided by choosing intermediate objectives based on the real-time environment.



**Figure 1. Description of the environment**

The outline of this paper is as follows. The problem statement and the optimal control problem are described in Section 2. Section 3 gives the main results. Simulation results are detailed in Section 4.

## 2 Path planning: an optimal control point of view

### 2.1 Problem Statement

In general cases, the environment may be complex and normally obstacles cannot be described as circles, as assumed in [24] (see Fig. 1 for example). Moreover, due to the distance limitation of sensors equipped on robots, only a portion of an obstacle can be captured, so that the robot may not know the exact shape of obstacles. In this case, obstacles can neither be described as circles nor be described as complete polygons.

As a result, the goal of this paper is to represent obstacles in a more accurate way, and to propose an efficient path planning algorithm which guarantees the safe navigation of a robot from a known initial position to a desired target in unknown environments while satisfying the physical

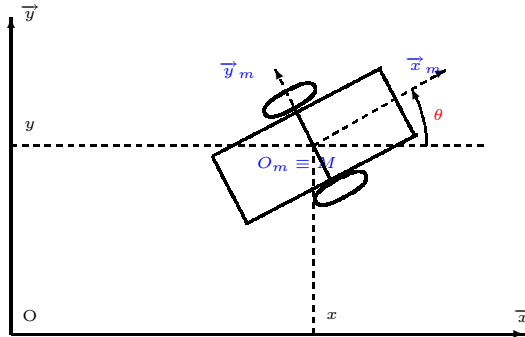
constraints of the robot.

## 2.2 Mobile robot modeling

This paper considers the unicycle-type mobile robot whose kinematic model can be described as:

$$\begin{cases} \dot{x} = v \cos \theta \\ \dot{y} = v \sin \theta \\ \dot{\theta} = \omega \end{cases} \quad (1)$$

where  $v$  and  $\omega$  are the linear and angular velocity respectively,  $\theta$  is the orientation of the robot body with respect to X-axis,  $U = [v, \omega]^T$  is the control input, and  $q = [x, y, \theta]^T$  is the system state (see Fig. 2).



**Figure 2. Unicycle-type mobile robot**

Without loss of generality, let us make the following assumptions:

*Assumption 1:* There is pure rolling situation (i.e., no slipping and sliding phenomenon) for robots, thus non-holonomic constraint of the robot can be described as:

$$[-\sin \theta \quad \cos \theta \quad 0] \dot{q} = 0$$

*Assumption 2:* This paper considers the unicycle mobile robot which allows turning in-place.

*Assumption 3:* The robot has only local view, thus only the closest obstacle in one direction can be detected.

It can be shown that  $x$  and  $y$  are flat outputs (see [27] for the definition) for the robot system (1). Indeed,  $\theta$ ,  $v$  and  $\omega$  can be expressed by  $x$ ,  $y$  and their first and second-order derivatives as follows:

$$\begin{cases} \theta &= \arctan \frac{\dot{y}}{\dot{x}} \\ v &= \sqrt{\dot{x}^2 + \dot{y}^2} \\ \omega &= \frac{\dot{y}\ddot{x} - \ddot{y}\dot{x}}{\dot{x}^2 + \dot{y}^2} \end{cases} \quad (2)$$

Thus one needs only to optimize  $x$  and  $y$  to obtain the optimal values of  $\theta$ ,  $v$  and  $\omega$ , where  $x$  and  $y$  are parameterized trajectories as stated in [24], and the optimal trajectories can be obtained by optimizing the parameters of the parameterized trajectories, which is described in the following.

## 2.3 Optimal control problem

### 2.3.1 Nonlinear Optimization Problem Formulation

As mentioned before, the path planning problem for mobile robots with physical constraints can be formulated as an optimal control problem. Generally speaking, it is to find the optimal control  $U = [v, \omega]$  for system (1) and to minimize the following cost function:

$$J = \int_{t_0}^{t_f} F(U(t), q(t), t) dt \quad (3)$$

where  $t_0$  and  $t_f$  are the initial time and the final time respectively,  $U = [v, \omega]$  and  $q = [x, y, \theta]$ .

$F$  is a function of  $U$  and  $q$  which defines the cost function to be minimized.  $F$  can be chosen in

advance, and can take several different forms. For example, when  $F = 1$ , i.e. to minimize the time  $t_f - t_0$ , it implies that the robot reaches the target as fast as possible. In this paper the cost function is chosen as the following one to guarantee the robot moving towards the objective:

$$F = ((x(t) - x_f)^2 + (y(t) - y_f)^2) \tag{4}$$

where  $(x_f, y_f)$  is the desired final position.

Moreover, the expected optimal control and the resulting states should satisfy the following constraints:

$C_1$  : the constraint on optimal control and state, i.e. the optimal control  $U$  and the states  $q$  should satisfy the kinematic model (1) for  $t \in [ t_0, t_f ]$ .

$C_2$  : the constraint on initial and final conditions, i.e.

$$q(t_0) = q(0), \quad q(t_f) = q_{final}$$

$C_3$  : the constraint on boundedness of control, i.e.

$$| v | \leq v_{max} \quad \text{and} \quad | \omega | \leq \omega_{max}$$

$C_4$  : the constraint on collision avoidance, i.e.

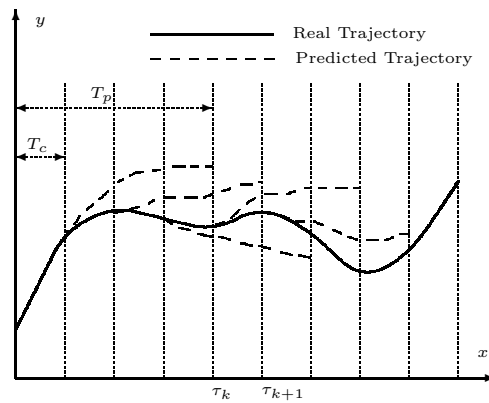
$$d(O, R) \geq r$$

where  $d(O, R)$  is the distance between the robot and any obstacle, and  $r$  is the given distance which guarantees the obstacle avoidance criterion.



### 2.3.2 Receding horizon planner

When the map is large, or is partially known, it is impossible to solve the above optimal control problem to obtain the whole optimal trajectory. In order to avoid this problem, the receding horizon planner [28] can be used to compute only a part of the trajectory from the current position to the final one over a time interval  $[\tau_k, \tau_k + T_c]$ , where  $T_c$  is the update period, and  $0 < T_c < T_p$ , where  $T_p$  is the trajectory planning horizon.



**Figure 3. Planning and update horizons**

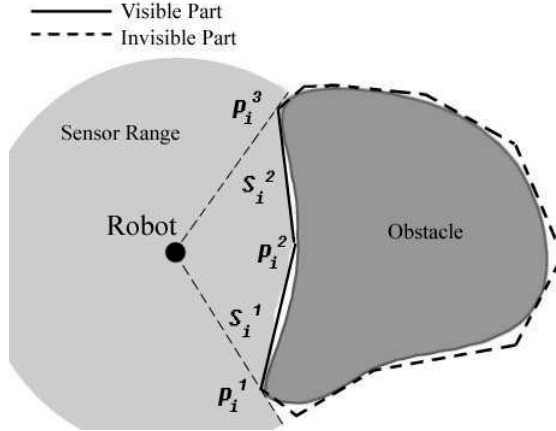
As shown in Fig. 3, the robot only computes a trajectory of horizon  $T_p$  and updates at each step  $\tau_k = \tau_{initial} + kT_c$ .

The optimal control problem with constraints over a receding horizon can be numerically solved by using the flatness property of the system [27], the parameterized trajectory and constrained feasible sequential quadratic optimization algorithm, for example CFSQP [29] (Feasible Sequential Quadratic Programming Algorithm proposed in [30]), for details see [24]. Then the open loop control  $U = [v, \omega]^T$  is deduced by using equation (2).

### 3 Path planning algorithm with intermediate objectives

#### 3.1 Representation of obstacles

In real situations, as stated in the problem statement, obstacles can neither be described as circles nor as complete polygons.



**Figure 4. Approximation of obstacles with complex shape**

Since a robot can only see a portion of an obstacle contour, as shown in Fig 4, the visible portion of the  $i^{th}$  obstacle contour can be approximated by a succession of segments  $S_i^j$ , where  $j = 1, 2, \dots, q$ , and  $q$  is the number of segments on an obstacle. Each segment is represented by its two end points  $p_i^j$  and  $p_i^{j+1}$ , and each point has their coordinates  $(x_{p_i^j}, y_{p_i^j})$  and  $(x_{p_i^{j+1}}, y_{p_i^{j+1}})$  respectively. The functions of the segments can be obtained by applying the image processing algorithms [31], which is beyond the scope of this paper. Thus, this paper assumes that the irregular obstacles are represented by a serial of segments.

**Remark 1** *If the distance between two obstacles  $d_{obs} < 2r$ , where  $r$  is the given distance which guarantees the obstacle avoidance criterion, then the two obstacles are considered as one, since*

the robot can not pass through the space between the two obstacles.

### 3.2 Distance between robot and segments

Since obstacles are represented by segments, the obstacle avoidance constraint  $C_4$  in the optimal control problem (3) becomes the distance constraint between the robot and segments.

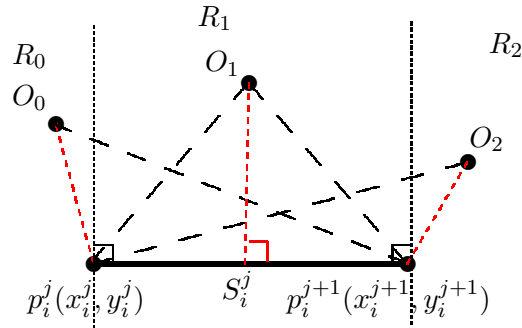
Denote  $O(x_o, y_o)$  the robot position,  $p_i^j(x_i^j, y_i^j)$ ,  $p_i^{j+1}(x_i^{j+1}, y_i^{j+1})$  the two end points of segment  $S_i^j$ . Then one can define the distances between those points:

$$d(O, p_i^j) = \sqrt{(x_o - x_i^j)^2 + (y_o - y_i^j)^2}$$

$$d(O, p_i^{j+1}) = \sqrt{(x_o - x_i^{j+1})^2 + (y_o - y_i^{j+1})^2}$$

$$d(p_i^j, p_i^{j+1}) = \sqrt{(x_i^j - x_i^{j+1})^2 + (y_i^j - y_i^{j+1})^2}$$

Thus, the distance between the robot and the segment  $S_i^j$ , noted as  $d(O, S_i^j)$ , can be calculated according to the relative position of the robot and the segment. There are three possible cases (see  $O_0, O_1, O_2$  in Fig. 5) :



**Figure 5. The three cases for distance calculation**

**Case 1**  $d(O, p_i^{j+1})^2 > d(O, p_i^j)^2 + d(p_i^j, p_i^{j+1})^2$ . In this case the robot locates in the left region  $R_0$

of  $S_i^j$ . It is easy to see that  $d(O, S_i^j) = d(O, p_i^j)$ , which is the red dotted line in  $R_0$ .

**Case 2**  $d(O, p_i^j)^2 > d(O, p_i^{j+1})^2 + d(p_i^j, p_i^{j+1})^2$ . In this case the robot locates in the right region  $R_2$  of  $S_i^j$ . It is obvious that  $d(O, S_i^j) = d(O, p_i^{j+1})$ , which is the red dotted line in  $R_2$ .

**Case 3** If not in case 1 and case 2, the robot is in region  $R_1$ . The distance between the robot and the segment  $d(O, S_i^j)$  can be obtained by simply using Heron's formula. A straightforward computation yields:

$$d(O, S_i^j) = 2 \frac{\sqrt{L(L - d(O, p_i^j))(L - d(O, p_i^{j+1}))(L - d(p_i^j, p_i^{j+1}))}}{d(p_i^j, p_i^{j+1})}$$

where  $L = \frac{d(O, p_i^j) + d(O, p_i^{j+1}) + d(p_i^j, p_i^{j+1})}{2}$ . See the red dotted line in  $R_1$ .

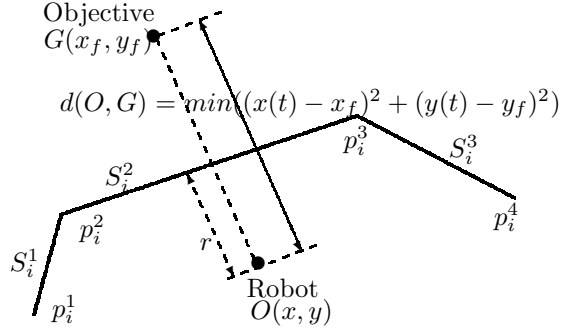
Summary, the distance between the robot and segment  $S_i^j$  is determined by the following equation:

$$d(O, S_i^j) = \begin{cases} d(O, p_i^j), & \text{case 1} \\ d(O, p_i^{j+1}), & \text{case 2} \\ 2 \frac{\sqrt{L(L - d(O, p_i^j))(L - d(O, p_i^{j+1}))(L - d(p_i^j, p_i^{j+1}))}}{d(p_i^j, p_i^{j+1})}, & \text{case 3} \end{cases} \quad (5)$$

However, it will be explained in the next section that this algorithm of using segments to represent obstacles suffers from local minima problems.

### 3.3 Local minima

It is worth noting that using of segments to represent obstacle contours inevitably involves local minima problems. This phenomenon happens when a robot arrives a point where the distance between the robot and the objective is minimum under the constraint of obstacle avoidance.



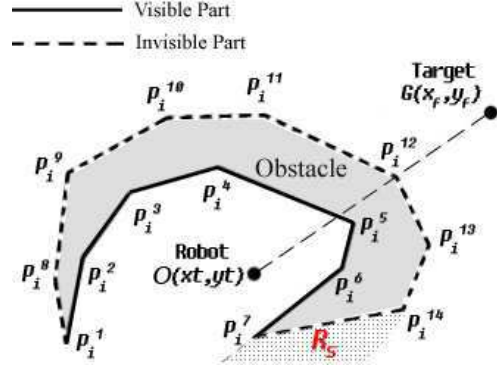
**Figure 6. Local minima**

As shown in Fig. 6,  $S_i^1$ ,  $S_i^2$  and  $S_i^3$  are segment obstacles, the robot gets to the local minima point  $O(x, y)$ ,  $r$  is the obstacle avoidance criterion. The robot needs to go left or right to avoid these obstacles, however, no matter the robot moves to left side or right side of point  $O(x, y)$ , the value of the cost function  $((x(t) - x_f)^2 + (y(t) - y_f)^2)$  in the optimization problem (3) will increase, thus the robot will stop at this point.

### 3.4 Avoidance of local minima by choosing intermediate objectives

This local minima problem cannot be avoided by the optimal path planning algorithm stated above. However one can notice that local minima problems might occur when the connection between the current robot position and the objective crosses with the segment (see in Fig. 6 the segment  $GO$  crosses  $S_i^2$ ). As a result, one can introduce some intermediate objectives for the robot if these intermediate objectives can guide the robot to escape local minima and to achieve the final objective.

Generally, the selection of intermediate objectives is according to the information detected by the sensor equipped on the robot. Once the intermediate objectives are chosen, optimal path



**Figure 7. Complex environment**

planning algorithm can be then used to calculate optimal trajectory between the current position of the robot and the intermediate objectives without local minima.

For example, in Fig. 6, one can choose intermediate objectives  $\{p_i^1, p_i^2, G\}$  instead of the final objective  $\{G\}$ , navigating the robot to reach  $p_i^1$ , then  $p_i^2$  and finally  $G$ . Then the optimal sub trajectories:  $O \rightarrow p_i^1, p_i^1 \rightarrow p_i^2$  and  $p_i^2 \rightarrow G$  can be calculated by solving the optimal control problem stated in section 2. It can be seen that if the robot follows this way, there will be no local minima phenomena.

### 3.5 Path planning algorithm with intermediate objectives

The “following the obstacle boundary mode” proposed in [25] can guarantee the avoidance of obstacles without local minima, but the robot needs to unnecessarily detour along with the contour of obstacles. For example in Fig. 7, robot need follow the contours  $\{p_i^4, p_i^3, p_i^2, p_i^1\}$  or  $\{p_i^5, p_i^6, p_i^7\}$  to avoid the concave obstacle. The proposed algorithm in this paper takes into account only the disjoint endpoints (the head  $p_i^1$  and the tail  $p_i^7$ ) of a serial of joint segments which is used to represent the detected partial obstacle.

The procedure of the proposed algorithm with intermediate objectives can be illustrated in Fig. 7. At the first time, the robot detects via sensors a serial of joint segments:  $\{p_i^1, \dots, p_i^7\}$  around its local environment. Since the dotted part  $\{p_i^8, \dots, p_i^{14}\}$  is invisible for this moment, the robot assumes that there is no obstacle in the invisible part, thus it thinks that the obstacle is only  $\{p_i^1, \dots, p_i^7\}$ . Then the robot chooses a temporary set of intermediate objectives in order to avoid local minima, noted as  $IO\_List = \{p_i^7, p_i^6, p_i^5, G\}$ . The robot gets the head of  $IO\_List$ , i.e.  $p_i^7$ , then it generates an optimal sub trajectory  $O \rightarrow p_i^7$  by solving optimal control problem defined in section 2. When the robot arrives in the region  $R_s$ , i.e. the region where robot can always see the second element in  $IO\_List$ ,  $p_i^6$  in this scenario, we remove the reached point  $p_i^7$  into a close list, noted as  $CloseList \leftarrow p_i^7$ . Robot scans again its surrounding and detects new obstacles represented by  $\{p_i^7, p_i^{14}\}$ . Since the head point  $p_i^7$  belongs to  $CloseList$ , which means that the robot has already reached this point, thus the intermediate objectives should be deduced from the tail point  $p_i^{14}$ , and the temporary list of intermediate objectives is updated as:  $IO\_List = \{p_i^{14}, G\}$ . Finally the robot can reach  $G$  by following this list.

**Remark 2** *Although the optimization method can be applied to any kind of mobile robots to drive the robot from one point to another one, however the approach described in this paper might not be applicable for a general non-holonomic robot (like car-like mobile robot) if the approaching direction to the intermediate point is not considered. The reason is that, after achieving the intermediate point, some kinds of robots will not have enough space to turn (due to the non-holonomic constraint) in order to achieve the next trajectory. However, the unicycle model considered in this paper has not such a problem since it allows turning in-place.*

From the above description, the proposed algorithm contains the following three aspects:

1. Select intermediate objectives from local information to generate a temporary list  $IO\_List$ ;
2. Be sure that the robot can reach another region (for example,  $R_s$  in Fig. 7);
3. Judge when the robot arrived at this region.

Before explaining these three aspects, let us give some notations which will be used in the sequel. Define  $\mathbb{P} = \{P_i\}$  for  $1 \leq i \leq N$  to be all sets of the obstacle boundaries detected by the equipped sensors, where  $N$  is the number of detected obstacle boundaries. Note  $P_i = \{p_i^j\}$  for  $1 \leq j \leq N_i$  as the set of joint points to represent the  $i^{th}$  obstacle boundary, where  $N_i$  is the number of points. Each segment  $S_i^j$  is defined by its endpoints  $S_i^j = (p_i^j, p_i^{j+1})$ . Let  $IO\_List = \{p_k, G\}$  for  $1 \leq k \leq m$  save the selected intermediate objectives. Denote  $dis(p_k p_{k+1})$  for  $1 \leq k \leq m - 1$  the function to calculate the distance between point  $p_k$  and point  $p_{k+1}$ , and note

$$dis(\{O\} \cup IO\_List) = dis(Op_1) + \sum_{k=1}^{m-1} dis(p_k p_{k+1}) + dis(p_m G)$$

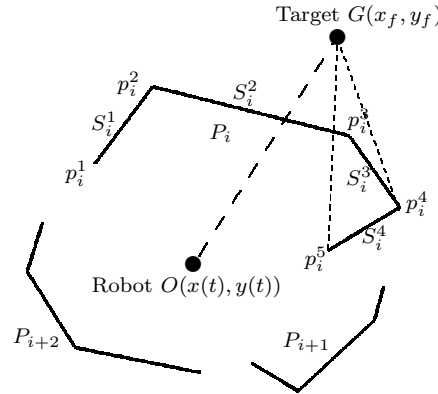
as the function to compute the complete path cost from the robot current position  $O$  to the final target  $G$  by following  $IO\_List$ .

For the reached intermediate points which have already treated, we remove them from  $IO\_List$  and save them on a  $CloseList$  to avoid unnecessary returns. Thus for an endpoint belonging to  $CloseList$ , the path cost from this path is set to be  $+\infty$ .  $List\_H$  and  $List\_T$  are defined to save two possible lists of intermediate objectives from the head and the tail of  $IO\_List$ , being initialized as  $List\_H = List\_T = \{G\}$ .



### 3.5.1 The intermediate objectives selection

Whenever the robot detects several obstacles around its surrounding, it always chooses the one with which the begin-final segment  $OG$  has an intersection. If  $OG$  has no intersection with all obstacles, then the robot can see the target  $G$  directly and thus the optimal path is the straight line  $OG$ . Otherwise,  $OG$  can have only one intersection with all obstacles, since it can detect only visible part of obstacles. For example, in Fig. 8,  $OG$  intersects with  $P_i$ , and the possible optimal trajectory might from  $p_i^1$  or  $p_i^5$ , but it is absolutely not possible from the obstacles  $P_{i+1}$  or  $P_{i+2}$  since those paths from  $P_{i+1}$  or  $P_{i+2}$  are obviously larger than the ones from  $P_i$ .



**Figure 8. Intermediate objectives generation**

After determining the exact obstacle (in Fig. 8, it is  $P_i$  since segment  $p_i^2 p_i^3$  in  $P_i = \{p_i^1, \dots, p_i^5\}$  intersects with  $OG$ ), we search intermediate objectives from both sides of  $OG$ . Take the right region of  $OG$  for example, one has the segment  $p_i^3 p_i^4$ , and check whether the segment  $Gp_i^4$  has an intersection with  $P_i$ . If not, that means the robot can see the final objective  $G$  after passing over the point  $p_i^4$ , thus the point  $p_i^3$  does not need to be added on  $List\_T$ . If segment  $Gp_i^4$  has an intersection with  $P_i$ , which implies that the robot is not able to see  $G$  after passing over  $p_i^4$ , thus

the robot needs to go to point  $p_i^3$  in order to see  $G$ , and  $p_i^3$  should be added on  $List\_T$ . Iteratively search next segment ( $p_i^4 p_i^5$  in Fig. 8) until the end of the segment of  $P_i$ , one obtains

$$List\_T = \{p_i^5, p_i^4, G\}$$

By applying the same procedure for the left region of  $OG$ , one gets

$$List\_H = \{p_i^1, p_i^2, G\}$$

Finally, the temporary list of intermediate objectives is then determined by the path cost of these two lists, i.e. if

$$dis(O, List\_T) > dis(O, List\_H)$$

then  $IO\_List = List\_H$ , otherwise  $IO\_List = List\_T$ .

The routine to generate the list of intermediate objectives is given in Algorithm 1.

### 3.5.2 Reach switching region

In order to clearly explain the algorithm, let us consider the following simple segment obstacle depicted in Fig. 9, and suppose that one has obtained the following list of intermediate objectives:

$$IO\_List = \{p_i^1, p_i^2, G\}$$

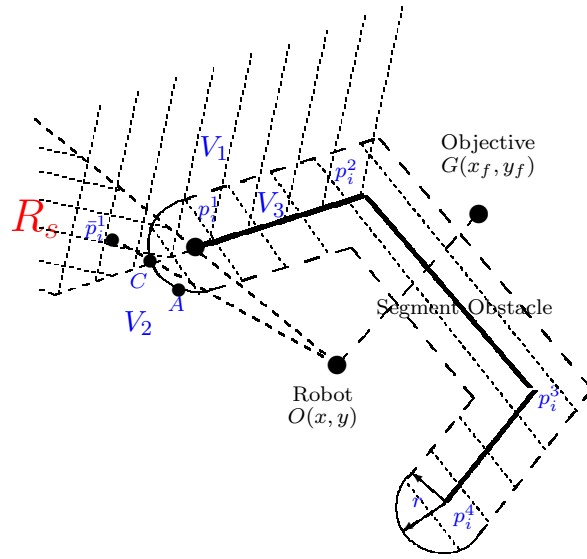
Thus the robot is guided to reach the first element in  $IO\_List$ , i.e.  $p_i^1$ , then  $p_i^2$  and finally  $G$ .

Then one can solve the optimal problem with constraints  $C_1 - C_4$  by minimizing the cost function

with respect to current intermediate objective, i.e.

$$\min \int_{T_p} \|O - p_i^1\|^2 dt, \text{ s.t. } C_1 - C_4 \quad (6)$$

The solution of this optimal problem yields optimal parameterized trajectories  $x$  and  $y$ , then one can get optimal control  $(v, \omega)$  according to (2).



**Figure 9. Intermediate objectives selection**

However, choosing directly  $p_i^1$  as final target in (6) results in local minima again. For example, as shown in Fig. 9, where  $r$  is the collision free distance, point  $A$  is on the boundary satisfying the obstacle avoidance criterion. In this situation, the robot may stop at point  $A$  since no other trajectories are better than to stay at this point (it has already minimized the cost function (6) under the constraints  $C_1 - C_4$ ). Finally, the robot cannot pass over the segment  $p_i^1 p_i^2$  to see the second intermediate objective  $p_i^2$ .

In order to make sure that the robot can always pass over the endpoint of the obstacle, let us give the following notations. Denote  $V_1$  the top region of line  $p_i^1 p_i^2$  (the region where the second intermediate objective  $p_i^2$  is visible), and  $V_2$  the left region of line  $O p_i^1$  (the region where robot can reach freely). Define  $V_3$  the collision constraint region:

$$V_3 = \{(x, y) : (x - x_i)^2 + (y - y_i)^2 \leq r^2, \forall (x_i, y_i) \in p_i^1 p_i^2\}$$

Then define the switching region  $R_s$  as follows:

$$R_s = (V_1 \cap V_2) \cap \bar{V}_3$$

where  $\bar{V}_3$  is the complement of  $V_3$ .

As the switching region is defined, one can see that the optimal path for the robot is to go directly into the switching region, as a result one can choose  $\bar{p}_i^1 \in R_s$  and replace  $p_i^1$  by  $\bar{p}_i^1$  in (6) to ensure that the robot goes into the switching region and avoids detours around the endpoint of the obstacles. Finally this optimal problem can be solved without local minima, since the robot can always pass over the segment  $p_i^1 p_i^2$  to see the second intermediate objective  $p_i^2$ .

In this paper, the modified intermediate objective  $\bar{p}_i^1$  is determined as follows (see Fig. 9): firstly find out the point  $C$  at a distance of  $r$  to the point  $p_i^1$  on the extension of segment from the endpoint  $p_i^2$  to the endpoint  $p_i^1$ , and then select  $\bar{p}_i^1$  at a distance of  $r$  to the point  $C$  on the extension of segment from the robot to the point  $C$ .

It is worth noting that the connection between the modified intermediate point  $\bar{p}_i^1$  and robot position may crosses with another obstacle, if the line connected between  $\bar{p}_i^1$  and robot crosses

with another obstacle, the intermediate objective should be selected from the obstacle that crosses with the line, and add the new intermediate objective to  $IO\_List$ , then generate new modified intermediate objective  $\bar{p}_i^1$  from new  $IO\_List$ . (See line 10-16 in Algorithm 4)

The routine to select intermediate objectives is given in Algorithm 2.

### 3.5.3 Judge the switching time

Suppose that one has  $IO\_List = \{p_i^1, p_i^2, \dots, G\}$  and the associated switching region  $R_s$ . Since the robot reinitializes and solves the optimal problem after every  $T_c$ , thus one can use the position of robot at  $t = 0$  and  $t = T_c$  (named as  $(x(0), y(0))$  and  $(x(T_c), y(T_c))$ ) to judge whether it enters  $R_s$ . For this, note the function  $f(x, y) = 0$  representing the first segment  $p_i^1 p_i^2$  in  $IO\_List$ . If

$$f(x(0), y(0))f(x(T_c), y(T_c)) < 0$$

one can judge that the robot has already entered the region  $R_s$ , which implies that the robot has passed over  $p_i^1$  and now can see  $p_i^2$ . Then we move  $p_i^1$  from  $IO\_List$  to  $CloseList$ .

The routine to judge the switching time is given in Algorithm 3.

## 3.6 Algorithm Description

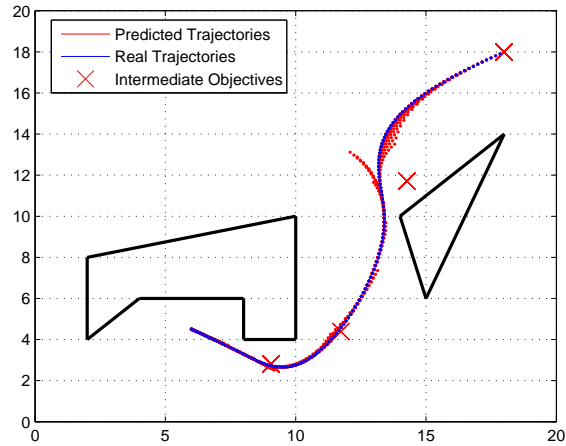
Given the temporary list of intermediate objectives  $IO\_List$ , which enables to define the switching region  $R_s$  and calculate the modified intermediate objective  $\bar{p}_i^1$ , one can solve optimal problem over  $T_p$  to get an optimal trajectory, which yields the optimal controls  $v$  and  $\omega$  for robot over  $[0, T_p]$ . Then one applies those optimal controls only for an interval  $[0, T_c]$ . When  $t = T_c$ , one iterates the same procedure as before, i.e., scans the surroundings to get obstacles  $\mathbb{P}$ , generates

the list of intermediate objectives  $IO\_List$ , calculates the modified intermediate objective  $\bar{p}_i^1$  and solves the optimal problem over  $T_p$  and implements the optimal controls for  $[0, T_c]$ . The algorithm stops when the robot reaches the final target  $G$ . The routine is detailed in Algorithm 4.

## 4 Simulation results

In order to show the feasibility and efficiency of the proposed algorithm, three simulations for different scenarios are made, and comparisons with visibility graph with expanded obstacles are made in the latter two simulations. The simulation settings are as follows: the range of robot sensors is 3  $m$ ; the maximum speed of robot is 1.0  $m/s$ , the maximum acceleration is 1.0  $m/s^2$ , the maximum angular velocity is 1.0  $rad/s$ , the maximum angular acceleration is 1.0  $rad/s^2$ . The planning horizon interval  $T_p$  is 2  $s$ , and the update period  $T_c$  is 0.2  $s$ .

For the simple scenario, depicted in Fig. 10, black polygons represent obstacles, containing a concave obstacle and a triangle obstacle. In this scenario, there exists a broad zone of local minima. The robot starts from the initial point (6, 4.5) to the target (18, 18). The red crosses in the figure are the intermediate objectives chosen by the proposed algorithm, the red trajectories are the predicted ones planned by the receding horizon planner, and the blue trajectories are the real trajectories. It can be seen that the proposed algorithm generates a safe and optimal path of intermediate objectives and avoids local minima successfully.



**Figure 10. Scenario 1: Simple environment**

Two more complex scenarios are shown in Fig. 11 and Fig. 12, arrows in the figures indicate the orientation of the robot, and the red crosses in the figure are the intermediate objectives chosen by the proposed algorithm. Comparisons with visibility graph with expanded obstacles are made, the blue trajectories are generated by the path planning algorithm proposed above, and the pink ones are generated by visibility graph.

One can see that in Fig. 11 several local minima exist, the robot starts from  $(7, 2)$  to  $(10, 20)$  and avoids all local minima by choosing intermediate objectives and reaches the target successfully by using only local sensor information. In Fig. 12, where there is a long winding corridor, the robot starts from  $(7, 2)$  to  $(25, 10)$ . It can be seen that the robot manages to walk through the long corridor and reach the target successfully while avoiding local minima and all the obstacles.

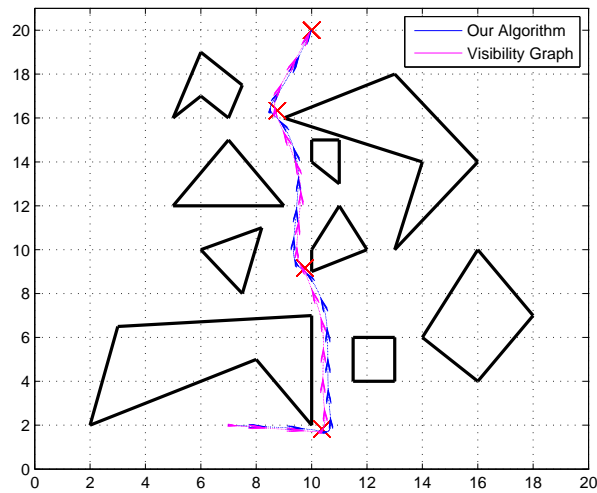
Comparisons with visibility graph are made. Normally visibility graph is used in global planning when the map is completely known, in order to use visibility graph in local planning with unknown map, the algorithm needs to generate expanded polygon for each obstacle in the local map and

search for the shortest path among all the obstacles, then iterate until the robot reaches the target. Instead, in our proposed algorithm the robot search for the shortest path only in some obstacles (normally one or two) in each iteration, which reduces the computational complexity compared to visibility graph approach.

As we can see in the figure 11, 12 and table 1, there are no big differences between the trajectories generated by two different methods, however it costs less time by using the method proposed in this paper.

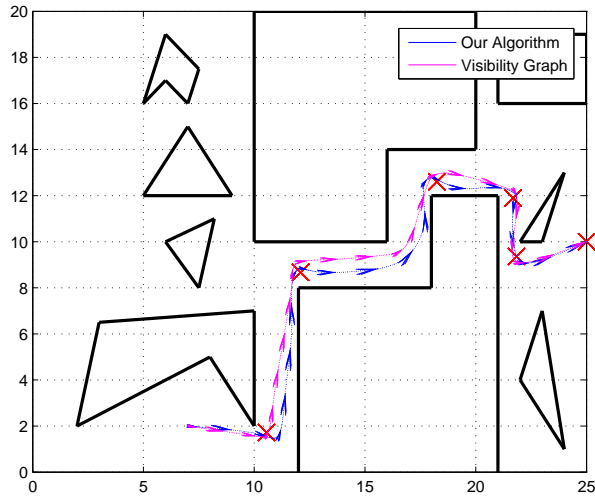
**Table 1. Comparison of simulation results**

	Method	Running Time (s)	Time-saving	Trajectory Length (m)
Scenario 2	Our method	3.02	19.2%	22.2
	Visibility Graph	3.74		22.1
Scenario 3	Our method	3.21	20.1%	30.2
	Visibility Graph	4.02		29.8



**Figure 11. Scenario 2: Complex environment**





**Figure 12. Scenario 3: Environment with corridor**

Two implementations made in real robot and real environment are in the attached video, and also can be found in the following link: [ROBOT VIDEO](#)

## 5 Conclusion

This paper presents a path planning algorithm for the navigation of non-holonomic mobile robots in unknown complex environments. The new algorithm takes into account irregular obstacles which are impossible to be approximated by circles. In order to avoid local minima problems, an algorithm of choosing intermediate objectives is proposed. The robot can reach the target and avoid obstacles by choosing appropriate intermediate objectives. Efficiency of the proposed algorithm is shown thereafter via different simulations and implementations in a wifibot, the simplicity of the algorithm is shown via the comparison with visibility graph.

## References

- [1] J. Pearsall, *Concise Oxford Dictionary*. Oxford University Press. Tenth Edition, Revised ed., 2001.
- [2] R. Brooks and T. Lozano-Perez, “A subdivision algorithm in configuration space for findpath with rotation,” *Systems, Man and Cybernetics, IEEE Transactions on*, vol. SMC-15, no. 2, pp. 224–233, 1985.
- [3] D. Glavaški, M. Volf, and M. Bonkovic, “Robot motion planning using exact cell decomposition and potential field methods,” in *Proceedings of the 9th WSEAS international conference on Simulation, modelling and optimization*, ser. SMO’09, 2009, pp. 126–131.
- [4] G. Dudek and M. Jenkin, *Computational Principles of Mobile Robotics*, ser. Computational Principles of Mobile Robotics. Cambridge University Press, 2010.
- [5] H.-P. Huang and S.-Y. Chung, “Dynamic visibility graph for path planning,” in *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3, 2004, pp. 2813 – 2818.
- [6] A. Bicchi, G. Casalino, and C. Santilli, “Planning shortest bounded-curvature paths for a class of nonholonomic vehicles among obstacles,” *Journal of Intelligent and Robotic Systems*, vol. 16, no. 4, pp. 387–405, 1996.
- [7] C. ’Dnlaing and C. K. Yap, “A ”retraction” method for planning the motion of a disc,” *Journal of Algorithms*, vol. 6, no. 1, pp. 104–111, 1985.
- [8] E. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.
- [9] P. Hart, N. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.
- [10] A. C. Nearchou, “Path planning of a mobile robot using genetic heuristics,” *Robotica*, vol. 16, no. 5, pp. 575–588, 9 1998.
- [11] A.-T. Ismail, A. Sheta, and M. Al-Weshah, “A mobile robot path planning using genetic algorithm in static environment,” *Journal of Computer Science*, vol. 4, no. 4, pp. 341–344, 2008.
- [12] J. T. Schwartz and M. Sharir, “On the piano movers’ problem i. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers,” *Communications on Pure and Applied Mathematics*, vol. 36, no. 3, pp. 345–398, 1983.
- [13] N. J. Nilsson, *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980.

- [14] A. Stentz, “Optimal and efficient path planning for partially-known environments,” in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, 1994, pp. 3310–3317.
- [15] H. Choset and K. Nagatani, “Topological simultaneous localization and mapping (slam): toward exact localization without explicit localization,” *Robotics and Automation, IEEE Transactions on*, vol. 17, pp. 125–137, apr 2001.
- [16] Y. Goto, , Y. Goto, and A. Stentz, “Mobile robot navigation: The cmu system,” *IEEE Expert*, vol. 2, pp. 44–54, 1987.
- [17] V. J. Lumelsky and A. A. Stepanov, “Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape,” *ALGORITHMICA*, 1987.
- [18] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2, 1985, pp. 500–505.
- [19] J.-C. Latombe, *ROBOT MOTION PLANNING.: Edition en anglais*, ser. The Springer International Series in Engineering and Computer Science. Springer, 1991.
- [20] Y. Koren and J. Borenstein, “Potential field methods and their inherent limitations for mobile robot navigation,” in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, vol. 2, 1991, pp. 1398–1404.
- [21] I. Kolmanovsky and N. McClamroch, “Developments in nonholonomic control problems,” *Control Systems, IEEE*, vol. 15, no. 6, pp. 20–36, 1995.
- [22] J. Laumond, *Robot motion planning and control*, ser. Lecture notes in control and information sciences. Springer, 1998.
- [23] Y. Guo and T. Tang, “Optimal trajectory generation for nonholonomic robots in dynamic environments,” in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, may 2008, pp. 2552–2557.
- [24] M. Defoort, J. Palos, A. Kokosy, T. Floquet, and W. Perruquetti, “Performance-based reactive navigation for non-holonomic mobile robots,” *Robotica*, vol. 27, no. 2, pp. 281–290, mar 2009.
- [25] A. Kokosy, F.-O. Defaux, and W. Perruquetti, “Autonomous navigation of a nonholonomic mobile robot in a complex environment,” in *Safety, Security and Rescue Robotics, 2008. SSRP 2008. IEEE International Workshop on*, 2008, pp. 102–108.
- [26] I. Kamon, E. Rimon, and E. Rivlin, “A new range-sensor based globally convergent navigation algorithm for mobile robots,” in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 1, apr 1996, pp. 429–435 vol.1.
- [27] M. Fliess, J. Lvine, and P. Rouchon, “Flatness and defect of nonlinear systems: Introductory theory and examples,” *International Journal of Control*, vol. 61, pp. 1327–1361, 1995.

- [28] D. Mayne and H. Michalska, “Receding horizon control of nonlinear systems,” *Automatic Control, IEEE Transactions on*, vol. 35, no. 7, pp. 814–824, 1990.
- [29] C. Lawrence, J. Zhou, and A. Tits, “User’s guide for CFSQP version 2.5: AC code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints,” *Institute for Systems Research TR*, vol. 94, p. 16r1, 94.
- [30] C. T. Lawrence and A. L. Tits, “A computationally efficient feasible sequential quadratic programming algorithm,” *SIAM Journal on Optimization*, vol. 11, pp. 1092–1118, 2001.
- [31] R. L. Graham, “An efficient algorithm for determining the convex hull of a finite planar set,” *Information Processing Letters*, vol. 1, no. 4, pp. 132–133, 1972.

## Appendix

---

**Algorithm 1** The Intermediate Objectives List Generation Function

---

1: **Function** *IO\_Generation* ( $G(x_f, y_f), O(x(t), y(t)), \mathbb{P}$ )

2: **for** each  $P_i$  **do**

3:   **if**  $\exists p_i^j, p_i^{j+1} \in P_i$  s.t.  $p_i^j p_i^{j+1} \cap OG \neq \emptyset$  **then**

4:     select  $P_i$ , break

5:   **end if**

6: **end for**

7:  $List\_T = List\_H = \{G\}$

8: **for**  $k=j+1: 1:m-1$  **do** ▷  $m$  is the number of points on  $P_i$

9:   **if**  $Gp_i^{j+2} \cap P_i \neq \emptyset$  **then**

10:      $List\_T = \{p_i^{j+1}\} \cup List\_T$

11:   **end if**

12: **end for**

13:  $List\_T = \{p_i^m\} \cup List\_T$

14: **for**  $k=j:-1:2$  **do**

15:   **if**  $Gp_i^{j-1} \cap P_i \neq \emptyset$  **then**

16:      $List\_H = \{p_i^j\} \cup List\_H$

17:   **end if**

18: **end for**

19:  $List\_H = \{p_i^1\} \cup List\_H$

20: **if**  $p_i^m \in CloseList$  **then**

21:    $dist(List\_T) = +\infty$

22: **end if**

23: **if**  $p_i^1 \in CloseList$  **then**

24:    $dist(List\_H) = +\infty$

25: **end if**

26: **if**  $dist(\{O\} \cup List\_T) \leq dist(\{O\} \cup List\_H)$  **then**

27:   Return  $List\_T$

28: **else** 28

29:   Return  $List\_H$

30: **end if**

---

**Algorithm 2** Selection of the Intermediate Objective

---

- 1: **Function**  $IO\_Selection (O(x(t), y(t)), IO\_List)$
  - 2: Get the first segment  $p_i^1 p_i^2$  from  $IO\_List$
  - 3: Get the function  $f(x, y) = 0$  for this segment;
  - 4: Compute point  $C$  s.t.  $f(x_c, y_c) = 0$ ,  $dis(Cp_i^1) = r \ dis(Cp_i^2) = r + dis(p_i^1 p_i^2)$ ;
  - 5: Determine the function  $g(x, y) = 0$  for the segment  $OC$ ;
  - 6: Select  $\bar{p}_i^1$  s.t.  $g(x_{\bar{p}_i^1}, y_{\bar{p}_i^1}) = 0$ ,  $dis(C\bar{p}_i^1) = r \ dis(O\bar{p}_i^1) = r + dis(OC)$ ;
  - 7: Return  $\bar{p}_i^1$
- 

---

**Algorithm 3** Switching Time

---

- 1: **Function**  $Switch ((x(0), y(0)), (x(T_c), y(T_c)), IO\_List)$
  - 2: Get the first segment  $p_i^1 p_i^2$  from  $IO\_List$
  - 3: Get the function  $f(x, y) = 0$  for this segment;
  - 4: **if**  $f(x(0), y(0)) \times f(x(T_c), y(T_c)) < 0$  **then**
  - 5:     Remove the 1st element  $p_i^1$  from  $IO\_List$
  - 6:     Add  $p_i^1$  into  $CloseList$
  - 7: **end if**
-

---

**Algorithm 4** Path Planning

---

```
1: Function PathPlanning( $G(x_f, y_f)$ , Ini. conditions)
2:  $t = 0$ 
3: while  $(x(0) - x_f)^2 + (y(0) - y_f)^2 \geq \varepsilon$  do
4:   Get  $\mathbb{P}$  from sensor
5:    $IO\_List = IO\_Generation((G(x_f, y_f), O(x(0), y(0)), \mathbb{P}))$ 
6:   if  $IO\_List = \{G\}$  then ▷ Can see  $G$ 
7:      $[x, y] = Optimisation(G)$  over  $T_p$ 
8:   else
9:     Get  $\bar{p}_i^1 = IO\_Selection(IO\_List)$  ▷ Opt. with the 1st intermediate objective
10:    for each  $P_i$  do ▷ Check  $\bar{p}_i^1$  can be seen or not
11:      if  $\exists p_i^j, p_i^{j+1} \in P_i$  s.t.  $p_i^j p_i^{j+1} \cap \bar{p}_i^1 O \neq \emptyset$  then
12:         $add\_List = IO\_Generation(\bar{p}_i^1, O(x(0), y(0)), \mathbb{P})$ 
13:         $IO\_List = add\_List \cup IO\_List$ 
14:        Get  $\bar{p}_i^1 = IO\_Selection(IO\_List)$ 
15:      end if
16:    end for
17:     $[x, y] = Optimisation(\bar{p}_i^1)$  over  $T_p$ 
18:  end if
19:  for  $t \in [0, T_c]$  do
20:    Get  $(v, \omega)$  from (2) based on  $x$  and  $y$ 
21:    Apply  $(v, \omega)$  to the robot
22:  end for
23:   $Switch((x(0), y(0)), (x(T_c), y(T_c)), IO\_List)$ 
24:  Reset  $t = 0$ 
25: end while
```

---