Collision detection between robots moving along specified trajectories [1]

Nelson Feyeux

Ecole Centrale de Lyon
36 Avenue Guy de Collongue, 69134 Ecully Cedex, France
Nelson.Feyeux@ecl2013.ec-lyon.fr

Chantal Landry

Weierstrass Institute
Mohrenstr. 39, 10117 Berlin, Germany
chantal.landry@wias-berlin.de

2010 Mathematics Subject Classification. 51M20, 51K99, 52B10, 68T40

Key words and phrases. Collision detection, distance computation, motion planning, robotics

[1]This work has been partially supported by the DFG Research Center Matheon "Mathematics for key technologies" in Berlin.

Abstract. An algorithm to detect collisions between robots moving along given trajectories is presented. The method is a combination of the adaptive dynamic collision checking developed by Schwarzer et al. and Lin and Canny's algorithm, which computes efficiently the distance between two polyhedra. The resulting algorithm is part of a global model that computes the optimal task assignment, sequencing and kinodynamic motion planning in a robotic work-cell.

1 Introduction

The application of our research lies in automotive industry and their production lines. A production line is divided into work-cells, which consists of a workpiece, several robots and some obstacles. Typical obstacles are the conveyor belt, which connects the cells to each other. In a work-cell, the robots perform tasks on the same workpiece. The total time taken by the robots to complete all the tasks is called the *makespan* of the work-cell. To ensure the competitiveness of car manufacturers, the makespan of the work-cells must be as small as possible. The makespan is minimal when the following three points are optimized:

1. kinodynamic motion planning of each robot,
2. task assignment between the robots,
3. sequencing of the tasks of each robot.

The kinodynamic motion planning computes the fastest trajectory that relies two locations and does not collide with the obstacles. Task assignment involves deciding which robot performs which tasks. The sequencing determines in which order the robot executes its assigned tasks. One sequence is defined per robot. The first and last elements of a sequence are the initial position of the robot. The elements in between are task locations. The traversal time of a sequence is the sum of the travel time of the fastest collision-free trajectory between two consecutive elements of the sequence. The travel times are obtained by solving the kinodynamic motion planning problem. The makespan is then equal to the largest traversal time.
The problem of minimizing the makespan is called the *Work-Cell Problem* (WCP). A complete description of (WCP) can be found in [13]. This problem is a typical instance of Vehicle Routing Problem [25]. Therefore, (WCP) is modelled by a directed graph. The nodes of the graph are the task locations and the initial position of the robots. Two nodes are connected by an arc if a robot can move from one node to the other. The weight on the arc is the travel time.
An iterative method is used to solve (WCP). First, a discrete optimization method computes a sequence of tasks for each robot. These sequences are such that the makespan is minimized. By definition of (WCP), the trajectories between two consecutive nodes avoid the obstacles. The collision avoidance between the robots has not been taken into consideration yet. That is why the second step involves detecting such collisions. If the robots hit each other, the algorithm goes back to the first step and looks for new sequences. If no collision occurs, then (WCP) is solved. The aim of this paper is to present how to achieve the second step efficiently.
There exist two classes of collision detection methods: the *static* and the *dynamic* collision detection. The static approach checks if there is a collision between two objects at each time step [19]. The *dynamic* collision checking determines if for all configurations given

on a continuous path a collision occurs between the objects. Cameron in [2] compared the static approach with two types of dynamic checking:

- *The space-time approach*: build for each object a volume that represents the spatial occupancy of the object along its trajectory. The volume is obtained by extruding the original configuration of the object.

- *The sweeping approach* [1]: compute the volume swept out by the object.

Once these volumes are built, both methods check a possible intersection between the volume of each object.
Cameron pointed out the advantages and drawbacks of each approach. The static detection is simple, but can miss a collision if the time discretization is too rough. On the other side taking small time steps is time consuming. The choice of a good number of time steps is still a difficult task. The space-time approach is an elegant method. However, extruding the volume can be complicated when the object undergoes rotations and has complex geometry. The sweeping approach has the same drawback. Moreover the technique can detect a collision even though no intersection exists between the objects. In conclusion, none of theses methods is the best.
After Cameron's article new techniques have been introduced such as (See Jiménez et al. [12] and Lin and Gottschalk [18] for a good survey)

- The *trajectory parameterization* method [4, 8, 23]: the collision checking is modelled by finding numerically roots of an algebraic polynomial in a single variable t.

- The *feature-tracking* method [5, 17, 21]: there exist three kinds of features: vertices, edges and faces. The feature-tracking methods involves computing the pair of closest features (one feature per object) and looking if this pair remains separated along the trajectory.

- The *bounding volume hierarchy* (BVH) method [6]: this technique computes first a hierarchy of bounding volumes (spheres [11, 22], oriented bounding boxes [10], axis-aligned bounding boxes [5], rectangular swept sphere [14], ...) to describe each object at various levels of details. Then the algorithm checks for collision by considering first the volumes of the roughest level. If a collision is detected, then the next level is considered and the associated volumes are tested for overlaps. If a pair of volumes in the finest level of detail is overlapping, a collision between the objects occurs. This method is then applied at given configurations on the trajectory. If all these configurations are collision-free, then no collision occurs between the objects.

As Schwarzer et al. observed in [24], all these methods have drawbacks. The trajectory parameterization technique is computationally expensive and could suffer of numerical problems when the polynomials have high degree. Feature-tracking requires small increments to avoid missing collisions, which can become computationally expensive when working with real robots. (BVH) methods is a static collision detection method. So, (BVH) could miss a collision if the number of time steps is too small.
Schwarzer et al. suggested in [24] a new dynamic collision detection method. They established a sufficient condition to determine if two polyhedra do not collide over a time interval. The condition involves checking if an inequality holds. If the inequality is not satisfied, then the time interval is split into two subintervals and the condition is checked

on both subintervals. This method is simple, never fails and adapts automatically the sampling resolution. For all these reasons, we choose to follow Schwarzer et al.'s method. In Section 2, the input data for the second step in the iterative method to solve (WCP) are presented. In Section 3, Schwarzer et al.'s inequality to determine if a collision occurs between two moving robots is established. The computation of the left hand side of the inequality is given in Section 4, whereas Section 5 is concerned with the right hand side. Finally, numerical results are given in Section 6.

2 Input data

Let us consider two robots that we denote, without loss of generality, R_1 and R_2. Both robots are a two-dimensional convex polyhedron. The robot R_1 is assumed to move from the position V_I^1 to V_F^1, whereas the robot R_2 goes from V_I^2 to V_F^2 during the same period. We would like to know if R_1 and R_2 collide during this period. In the sequel let R_i denote the robots R_1 and R_2 in a general way.
The motion of the robots between their initial and final position, is given. As part of (WCP), the trajectory was obtained by solving a kinodynamic motion planning problem [13]. This problem was solved with a time discretization [7]. Therefore, the motion is a discretized trajectory given in the following manner: if $0 \leq t_1 < \ldots < t_n$ denote the time discretization with n time steps, then the configuration of the robot R_i at time step t_k is approximated by the tuple

$$p_i^{(k)} := (G_i^{(k)}, v_i^{(k)}, \theta_i^{(k)}, \mu_i^{(k)}),$$

where $G_i^{(k)} = (x_i^{(k)}, y_i^{(k)})$ is the position of the center of gravity of the robot, $v_i^{(k)} = (v_{i,x}^{(k)}, v_{i,y}^{(k)})$ is the velocity of the center of gravity, $\theta_i^{(k)}$ is the angle of rotation and $\mu_i^{(k)}$ is the velocity of the angle of rotation. The axis of rotation is placed at the center of gravity of the robot. In particular, we have $G_i^{(1)} = V_I^i$ and $G_i^{(n)} = V_F^i$.
In the sequel, the trajectory must be evaluated at intermediate times. For that purpose, the trajectory is approximated by a linear interpolation. If we consider the time interval $[t_k, t_{k+1}]$, the intermediate configurations are approximated by

$$\bar{p}_i^{(k)}(t) := \frac{t_{k+1} - t}{t_{k+1} - t_k} p_i^{(k)} + \frac{t - t_k}{t_{k+1} - t_k} p_i^{(k+1)}, \quad \forall t \in [t_k, t_{k+1}]. \tag{1}$$

With each robot, we associate a Cartesian coordinate system (G_i, e_x^i, e_y^i), $i = 1, 2$, whose origin is the center of gravity G_i. The pair (e_x^i, e_y^i) is the $x-$ and $y-$ axes respectively. This coordinate system is called *body frame* according to [15]. A Cartesian coordinate system is also associated to the workspace. This system is named *world frame* and denoted by (O, e_x, e_y). The configuration $p_i^{(k)}$ allows us to represent the robot R_i at time step t_k in the world frame. The position of the robot is obtained by rotating the robot counterclockwise of angle $\theta_i^{(k)}$ and translating it along the vector $\overrightarrow{OG_i}^{(k)}$, see Figure 1.
Similarly, if P_i is a vertex of the robot R_i, whose coordinates are given in the body frame, then the coordinates of P_i in the world frame at time $t^{(k)}$ are

$$P_i^{(k)} := \mathcal{R}(\theta_i^{(k)}) P_i + G_i^{(k)}, \tag{2}$$

where

$$\mathcal{R}(\theta_i^{(k)}) = \begin{pmatrix} \cos(\theta_i^{(k)}) & -\sin(\theta_i^{(k)}) \\ \sin(\theta_i^{(k)}) & \cos(\theta_i^{(k)}) \end{pmatrix} \tag{3}$$

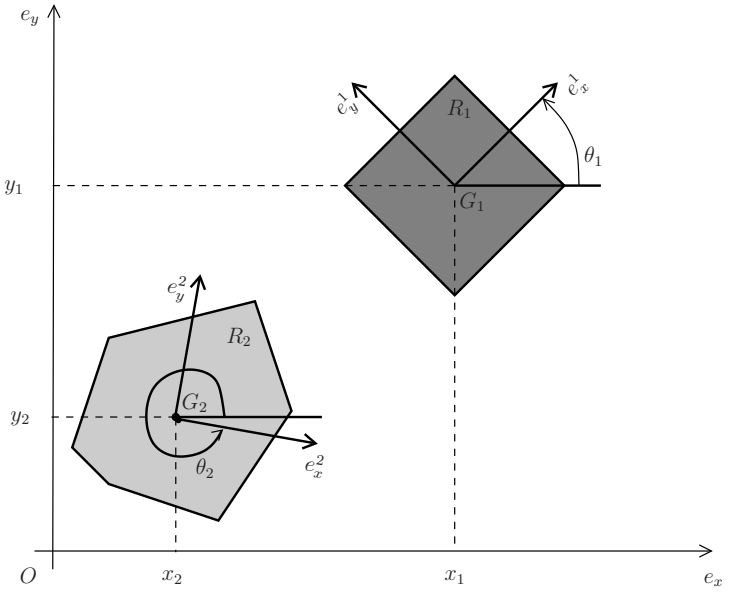


Figure 1: The position of the robot R_1 (resp. R_2) in the world frame (O, e_x, e_y), is given by its center of gravity G_1 (resp. $G_2 = (x_2, y_2)$) and the rotation angle θ_1 (resp. θ_2).

is the rotation matrix.
In summary, the approximated trajectory of a robot R_i is given by a sequence of time steps $(t_k)_{k=1}^n$ and the associated sequence of configurations $(p_i^{(k)})_{k=1}^n$. Since the number and values of time steps are specific to each robot, we define $(\tau_k)_{k=1}^n$ and $(p^{(k)})_{k=1}^n$ as the sequences associated with R_1, and $(T_k)_{k=1}^N$ and $(q^{(k)})_{k=1}^N$ as those with R_2. Knowing these four sequences, we would like to detect if a collision occurs between R_1 and R_2.
Since the robots have usually a different time discretization, we need first to classify the time steps $(\tau_k)_{k=1}^n$ and $(T_k)_{k=1}^N$ in an ascending order as illustrated in Figure 2. The time interval is then decomposed on subintervals $[t_l, t_u]$ of type: $[\tau_\ell, \tau_{\ell+1}]$, $[\tau_\ell, T_j]$, $[T_j, \tau_\ell]$ and $[T_j, T_{j+1}]$.
In the next section, we check on each subinterval if a collision between the robots occurs. For that purpose, the configuration of the robots must be evaluated at the boundary of the subinterval $[t_l, t_u]$. If t_l (resp. t_u) belongs to $[\tau_1, \tau_n]$, then the configuration p_l (resp. p_u) of the robot R_1 is given by the linear interpolation defined in (1). The case $t_l < \tau_1$ means that the robot R_2 moves during the time interval $[t_l, \tau_1]$, whereas the robot R_1 stays at its initial place. In that case, we have $p_l = p^{(1)}$. Similarly, the scenario $t_u > \tau_n$ signifies that the robot R_1 has reached its final destination V_F^1 and the robot R_2 is still moving. Consequently, $p_u = p^{(n)}$. In summary, the configuration of robot R_1 at t_l and t_u is given by

$$p_s := \begin{cases} p^{(\ell)} & \text{if } t_s = \tau_\ell, \\ p^{(1)} & \text{if } t_s = T_j \text{ and } T_j < \tau_1, \\ \bar{p}^{(\ell)}(T_j) & \text{if } t_s = T_j \text{ and } T_j \in [\tau_\ell, \tau_{\ell+1}],\ \ell = 1, \dots, n-1, \\ p^{(n)} & \text{otherwise,} \end{cases} \tag{4}$$

where $s \in \{l, u\}$ and $\bar{p}^{(\ell)}$ is the linear interpolation defined in (1). For robot R_2, the

configuration at the boundary of the time interval is determined similarly:

$$q_s \;:=\; \begin{cases} q^{(j)} & \text{if } t_s = T_j, \\ q^{(1)} & \text{if } t_s = \tau_\ell \text{ and } \tau_\ell < T_1, \\ \bar{q}^{(j)}(\tau_\ell) & \text{if } t_s = \tau_\ell \text{ and } \tau_\ell \in [T_j, T_{j+1}],\ j = 1, \ldots, N-1, \\ q^{(N)} & \text{otherwise }, \end{cases} \tag{5}$$

with $s \in \{l, u\}$.

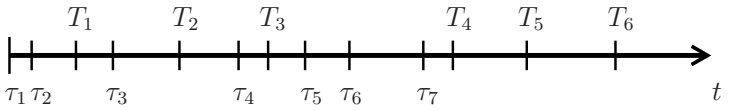


Figure 2: Ascending order of the time steps $(\tau_k)_{k=1}^n$ and $(T_k)_{k=1}^N$. The case $T_1 > \tau_2$ means that the robot R_2 stays at its initial position V_I^2 when R_1 is moving. Likewise, $\tau_7 < T_4$ signifies that the robot R_1 has reached its final destination V_F^1, whereas the robot R_2 is still moving.

3 Collision detection algorithm

In this section, the dynamic collision detection method developed by Schwarzer, Saha and Latombe [24] is presented. Let us consider the time subinterval $[t_l, t_u]$ as defined in Section 2. Let us remind that the robots are a convex compact polyhedron. Therefore, the word robot is equivalent to polyhedron in the sequel.
The idea of Schwarzer et al. is to compare lower bounds of the distance travelled by the points of the robots during $[t_l, t_u]$ with an upper bound of the distance between both robots. Let us define the following quantity

- $\eta(t)$ is a non-trivial lower bound of the Hausdorff distance between the robots at time t. The robots overlap when $\eta(t) = 0$. Actually, we consider that the robots overlap when $\eta(t) < \delta$, where δ is a small positive parameter. In this way, a safety margin around the robots is guaranteed.

- $\lambda_i(t_a, t_b)$ is an upper bound of the length of the curves traced by all points in the robot R_i between t_a and t_b, with $t_a, t_b \in [t_l, t_u]$, $i \in \{1, 2\}$.

Schwarzer, Saha and Latombe established the following sufficient condition

Lemma 1 *Two polyhedra R_1 and R_2 do not collide at any time $t \in [t_l, t_u]$ if*

$$\lambda_1(t_l, t_u) + \lambda_2(t_l, t_u) \;<\; \eta(t_l) + \eta(t_u). \tag{6}$$

Hence, this inequality, which only depends on the boundary values t_l and t_u, allows us to know if R_1 and R_2 are collision-free over the whole interval $[t_l, t_u]$.
Proof of Lemma 1 (Schwarzer et al. [24]): Let us assume that the polyhedra R_1 and R_2 overlap at time $\tilde{t} \in [t_l, t_u]$. Then, a point S_1 of R_1 must coincide with a point S_2 of R_2 at $\tilde{t}$. Let $\ell_1(t_a, t_b)$ (resp. $\ell_2(t_a, t_b)$) define the length of the curve traced by S_1 (resp. S_2) during $[t_a, t_b]$. An illustration is given in Figure 3, where R_1 is the white square and R_2

is the black square. The robots are represented at t_l, $\tilde{t}$ and t_u. Intermediate locations are illustrated with dotted lines. The grey lines are the curves followed by S_1 and S_2. Because S_1 and S_2 coincide at $\tilde{t}$, we have

$$\ell_1(t_l, \tilde{t}) + \ell_2(t_l, \tilde{t}) \geq \eta(t_l), \quad (7)$$
$$\ell_1(\tilde{t}, t_u) + \ell_2(\tilde{t}, t_u) \geq \eta(t_u). \quad (8)$$

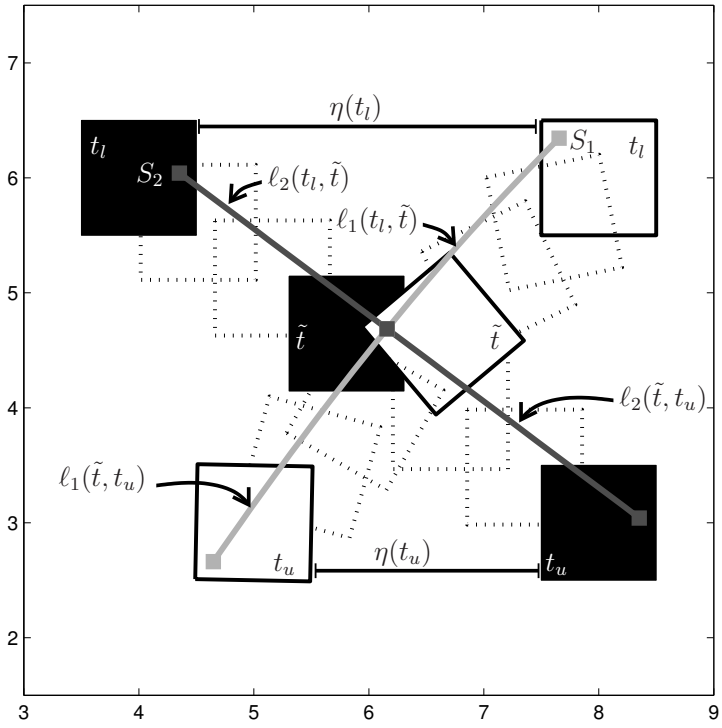


Figure 3: Illustration of the colliding case. The robot R_1 is the white square and the robot R_2 is the black square. The robots collide at time $\tilde{t}$.

Summing (7) and (8), we obtain:

$$\ell_1(t_l, t_u) + \ell_2(t_l, t_u) \geq \eta(t_l) + \eta(t_u). \quad (9)$$

The quantities $\lambda_1(t_l, t_u)$ and $\lambda_2(t_l, t_u)$ being an upper bound, we deduce from (9):

$$\lambda_1(t_l, t_u) + \lambda_2(t_l, t_u) \geq \ell_1(t_l, t_u) + \ell_2(t_l, t_u) \geq \eta(t_l) + \eta(t_u),$$

what contradicts the inequality (6).

■

Lemma 1 is a sufficient condition. Therefore, the reverse of the lemma is not true. We cannot conclude the collision-freeness on $[t_l, t_u]$, when the inequality is not satisfied. In that case, the time interval is split into two subintervals $[t_l, t_m]$ and $[t_m, t_u]$, where $t_m = \frac{1}{2}(t_l + t_u)$. Then, the lower bound $\eta(t_m)$ is computed. If $\eta(t_m) < \delta$, a collision between the robots is detected. Otherwise, the sufficient condition of Lemma 1 is checked on both subintervals $[t_l, t_m]$ and $[t_m, t_u]$. If the inequality (6) is not satisfied for a subinterval, then

Table 1: Collision detection algorithm on $[t_l, t_u]$

```
If η(t_l) < δ or η(t_u) < δ then
   return false
else
   return Inequality(t_l, t_u)
end if
```

where the recursive function `Inequality`(t_a, t_b) is

```
If λ_1(t_a, t_b) + λ_2(t_a, t_b) < η(t_a) + η(t_b) then
   return true
else if η((1/2)(t_a + t_b)) < δ then
   return false
else
   return Inequality(t_a, (1/2)(t_a + t_b)) or Inequality((1/2)(t_a + t_b), t_b)
end if
```

the subinterval is divided into two parts again and (6) is tested on each part. This process is repeated until either a collision is detected or all subintervals satisfy (6). The collision detection algorithm between R_1 and R_2 on $[t_l, t_u]$ is summarized in Table 1, where the output "true" means that R_1 and R_2 are separated and "false" indicates that a collision occurs.

A key point of this collision detection checking is that the algorithm automatically decides whether a time interval must be divided further. Furthermore, the method can never fail. To prove this fact, let us observe first that the upper bounds $\lambda_i(t_a, t_b) \to 0$ when $|t_b - t_a| \to 0$. Then, let us distinguish the cases:

- If no collision occurs in $[t_l, t_u]$, there exists a constant $\eta_{\min} \geq \delta > 0$ such that

  $$\eta(t) \geq \eta_{\min}, \ \forall t \in [t_l, t_u].$$

  With the bisection, the length of the new subintervals is always smaller. The left-hand side of the inequality (6) becomes smaller with the bisection, whereas the right-hand side remains lower-bounded by $\eta_{\min}$. Therefore, there exists a set of subintervals of $[t_l, t_u]$ such that

  1. the union of all subintervals is equal to $[t_l, t_u]$,
  2. the inequality (6) is satisfied on each subinterval.

- If the polyhedra overlap, then there is a time subinterval $[t_a, t_b] \subseteq [t_l, t_u]$ such that

  $$\eta(t) < \delta, \ \forall\, t \in [t_a, t_b],$$

  since the motion of the polyhedra is continuous. Then, by splitting the time intervals, the inequality (6) remains unsatisfied until the new middle point of the time interval falls into $[t_a, t_b]$. Let us illustrate this argument with the situation depicted

in Figure 4. The time interval $[t_a, t_b]$ is represented by the grey segment. The algorithm checks first if the polyhedra collide at t_l and t_u. The second step of the algorithm establishes that the inequality (6) is not satisfied. The first bisection is executed by computing $t_1 = \frac{1}{2}(t_l + t_u)$. No collision occurs at t_1 since $\eta(t_1) \geq \delta$. Inequality (6) may be satisfied on $[t_l, t_1]$, but not on $[t_1, t_u]$. Consequently, the middle point of $[t_1, t_u]$: $t_2 = \frac{1}{2}(t_1 + t_u)$ is computed. The quantity $\eta(t_2)$ is larger than δ. The bisection is then executed and we obtain the following subintervals $[t_1, t_2]$ and $[t_2, t_u]$. The inequality (6) is not satisfied on $[t_1, t_2]$. Next, let us compute $t_3 = \frac{1}{2}(t_1 + t_2)$ and check if $\eta(t_3)$ is larger than δ. Let us do so on until $t_4 = \frac{1}{2}(t_3 + t_2)$. For that point, we have $\eta(t_4) < \delta$. The collision is detected.

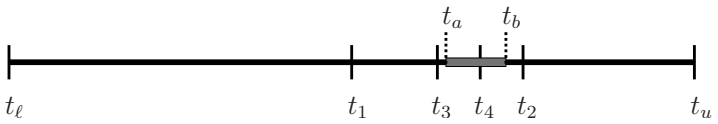


Figure 4: The collision detection algorithm when the polyhedra overlap. The collision is detected by checking if $\eta(t_4) < \delta$.

The algorithm in Table 1 is applied to all intervals $[t_l, t_u]$ issued from the ordering of $(\tau_k)_{k=1}^n$ and $(T_k)_{k=1}^N$. There exists different techniques to decide in which order the subintervals are checked, see e.g. [24]. These techniques allow us to detect the collisions faster. In our case, no technique has been implemented since we are interested in detecting the first collision. In the next sections, the computation of the upper and lower bounds in the inequality (6) are presented.

4 Computation of the upper bounds λ_1 and λ_2

In this section, we are interested in finding an upper bound of the length of the curves traced by the points of the robot between t_l and t_u. Schwarzer et al. established in [24] an upper bound when the robot is a planar linkage, whose joints are either revolute or prismatic. In that case, the configuration of the robot is either a rotation angle for the revolute joints or a translation vector for the prismatic joints. In our case, the robot is a polyhedron that moves freely in the workspace. We do not have any joints. The configuration space of the robot R_i is composed of the position of the center of gravity of the robot, G_i, the velocity of the center of gravity, v_i, the rotation angle, θ_i and the velocity of the rotation angle μ_i as mentioned in Section 2. Hence, we develop our own upper bound, but recommend to use the upper bound of Schwarzer et al. as soon as the robot is a linkage.
Let us consider the subinterval $[t_l, t_u]$ and two time steps $t_a, t_b \in [t_l, t_u]$ with $t_a < t_b$. The quantity $\lambda_i(t_a, t_b)$ is defined as an upper bound of the distance travelled by each point of the robot R_i. Consequently, $\lambda_i(t_a, t_b)$ satisfies

$$\lambda_i(t_a, t_b) \geq \max_{M_i \in R_i} \int_{t_a}^{t_b} \left\| \frac{d}{dt} \overrightarrow{OM_i}(t) \right\| dt,$$

where M_i is a point in R_i.

The right-hand side of the above inequality is hard to calculate exactly, since the robots are rotating. So, let us find an upper bound of the integral by exploiting the definition of $\overrightarrow{OM_i}$. This vector can be decomposed as follows (compare (2)):

$$\overrightarrow{OM_i}(t) \;=\; \overrightarrow{OG_i}(t) + \overrightarrow{G_iM_i}(t) \;=\; \overrightarrow{OG_i}(t) + \mathcal{R}(\theta_i(t))\,\overrightarrow{G_iM_i},$$

where $\mathcal{R}(\theta_i(t))$ is the rotation matrix defined by (3). Note that $\overrightarrow{G_iM_i}$ is fixed. Then, we have:

$$\begin{aligned}
\frac{d}{dt}\overrightarrow{OM_i}(t) &= \frac{d}{dt}\overrightarrow{OG_i}(t) + \frac{d}{dt}\mathcal{R}(\theta_i(t))\,\overrightarrow{G_iM_i},\\
&= v_i(t) + \theta_i'(t)\mathcal{R}\left(\theta_i(t)+\frac{\pi}{2}\right)\overrightarrow{G_iM_i},\\
&= v_i(t) + \mu_i(t)\mathcal{R}\left(\theta_i(t)+\frac{\pi}{2}\right)\overrightarrow{G_iM_i}.
\end{aligned}$$

For all rotation matrices $\mathcal{R}$ and all vectors x in $\mathbb{R}^2$ we have

$$||\mathcal{R}\,x||_2 = ||x||_2.$$

Using this relation, we obtain:

$$\begin{aligned}
\left\|\frac{d}{dt}\overrightarrow{OM_i}(t)\right\|_2 &\le \|v_i(t)\|_2 + |\mu_i(t)|\left\|\mathcal{R}\left(\theta_i(t)+\frac{\pi}{2}\right)\overrightarrow{G_iM_i}\right\|_2\\
&= \|v_i(t)\|_2 + |\mu_i(t)|\|\overrightarrow{G_iM_i}\|_2.
\end{aligned}$$

Let $R_{\max,i}$ be the radius of the smallest disc centered at G_i and containing all vertices of R_i, that is:

$$R_{\max,i} = \max_{M_i\in Ri} ||\overrightarrow{G_iM_i}||_2.$$

An illustration is given in Figure 5. Introducing $R_{\max,i}$ in the last inequality yields:

$$\left\|\frac{d}{dt}\overrightarrow{OM_i}(t)\right\|_2 \;\le\; \|v_i(t)\|_2 + |\mu_i(t)|R_{\max,i}.$$

Let us now integrate the above relation on both sides:

$$\int_{t_a}^{t_b}\left\|\frac{d}{dt}\overrightarrow{OM_i}(t)\right\|_2 dt \;\le\; \int_{t_a}^{t_b}\|v_i(t)\|_2 dt + R_{\max,i}\int_{t_a}^{t_b}|\mu_i(t)|dt. \tag{10}$$

The values of μ_i and v_i, $i=1,2$, are given according to the following linear interpolation

$$\begin{aligned}
\mu_i(t) &= a(t)\,\mu_i(t_a) + (1-a(t))\,\mu_i(t_b),\\
v_i(t) &= a(t)\,v_i(t_a) + (1-a(t))\,v_i(t_b), \;\text{ with } a(t) \;=\; \frac{t_b-t}{t_b-t_a} \in [0,1].
\end{aligned} \tag{11}$$

The integrals in the right-hand side of (10) become

$$\begin{aligned}
\int_{t_a}^{t_b}\|v_i(t)\|_2 &= \int_{t_a}^{t_b}\|a(t)v_i(t_a) + (1-a(t))v_i(t_b)\|_2 dt,\\
&\le \|v_i(t_a)\|_2\int_{t_a}^{t_b}a(t)\,dt + \|v_i(t_b)\|_2\int_{t_a}^{t_b}(1-a(t))\,dt,\\
&= \frac{1}{2}(t_b-t_a)(\|v_i(t_a)\|_2 + \|v_i(t_b)\|_2).
\end{aligned}$$

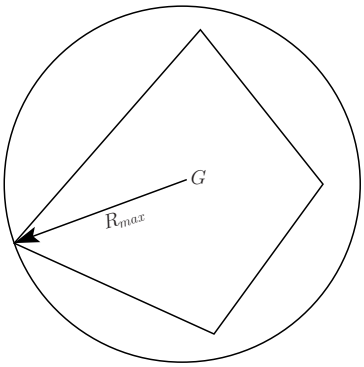


Figure 5: Disc of radius $R_{\max}$ and center G that contains the polyhedron.

Similarly, we have

$$\begin{aligned}\int_{t_a}^{t_b} |\mu_i(t)| &\leq |\mu_i(t_a)| \int_{t_a}^{t_b} a(t)\,dt + |\mu_i(t_b)| \int_{t_a}^{t_b} (1-a(t))\,dt,\\ &= \frac{1}{2}(t_b - t_a)(|\mu_i(t_a)| + |\mu_i(t_b)|).\end{aligned}$$

Finally, we get

$$\int_{t_a}^{t_b} \left\| \frac{d}{dt}\overrightarrow{OM_i}(t) \right\|_2 dt \leq \frac{1}{2}(t_b - t_a)(R_{\max,i}(|\mu_i(t_a)| + |\mu_i(t_b)|) + \|v_i(t_a)\|_2 + \|v_i(t_b)\|_2),\ i = 1, 2.$$

We define the upper bound $\lambda_i(t_a, t_b)$ as the right-hand side of the above inequality:

$$\lambda_i(t_a, t_b) := \frac{1}{2}(t_b - t_a)\left(R_{\max,i}(|\mu_i(t_a)| + |\mu_i(t_b)|) + \|v_i(t_a)\|_2 + \|v_i(t_b)\|_2\right). \tag{12}$$

In this formula, the terms $\mu_i(t_a)$, $\mu_i(t_b)$, $v_i(t_a)$ and $v_i(t_b)$ come also from a linear interpolation. Since $[t_a, t_b] \subseteq [t_l, t_u]$, we have

$$\begin{aligned}\mu_i(t_a) &= b(t_a)\,\mu_i(t_l) + (1 - b(t_a))\,\mu_i(t_u),\\ \mu_i(t_b) &= b(t_b)\,\mu_i(t_l) + (1 - b(t_b))\,\mu_i(t_u),\\ v_i(t_a) &= b(t_a)\,v_i(t_l) + (1 - b(t_a))\,v_i(t_u),\\ v_i(t_b) &= b(t_b)\,v_i(t_l) + (1 - b(t_b))\,v_i(t_u), \quad \text{with } b(t) = \frac{t_u - t}{t_u - t_l} \in [0, 1],\end{aligned}$$

and

$$\begin{aligned}p_l &= (G_1(t_l), v_1(t_l), \theta_1(t_l), \mu_1(t_l)), \qquad p_u = (G_1(t_u), v_1(t_u), \theta_1(t_u), \mu_1(t_u)),\\ q_l &= (G_2(t_l), v_2(t_l), \theta_2(t_l), \mu_2(t_l)), \qquad q_u = (G_2(t_u), v_2(t_u), \theta_2(t_u), \mu_2(t_u))\end{aligned}$$

derive from (4)-(5).

Remark 1 *The upper bound $\lambda_i(t_a, t_b)$ tends to zero when $|t_b - t_a|$ tends to zero. This fact was used to justify the convergence of the collision detection algorithm at the end of Section 3.*

5 Computation of the lower bound η

The function η in the inequality (6) is defined as a non-trivial lower bound of the real distance between two polyhedra. As in [5, 21], a two-phase approach is considered. This approach is composed of a *broad phase* and *a narrow phase*. In the broad phase, the polyhedra are approximated by a simple bounding volume such as an axis-aligned box or a sphere. The lower bound η is defined as the distance between the bounding volumes. As long as the bounding volumes are disjoint, the broad phase is applied. Once the bounding volumes overlap, the narrow phase is used. This phase computes the exact distance between the polyhedra. Thus the two-phase approach induces a minimal cost in the computation of η since the exact distance is determined only when the polyhedra are close to each other. If the robots would have a more complex geometry, then a hierarchy of bounding volumes would be defined such as in [5, 10, 11, 14, 22].
In our two-dimensional case, we define the bounding volume of the broad phase as the smallest disc surrounding the polyhedron and whose center is the center of gravity of the polyhedron. An illustration is given in Figure 5. Then, η is defined as the distance between the two disks, i.e.:

$$\eta(t) = \max(\|G_1G_2(t)\|_2 - R_{\max,1} - R_{\max,2}, 0), \tag{13}$$

where G_i and $R_{\max,i}$ are respectively the center and the radius of the disc $\mathcal{D}_i$ surrounding the robot R_i, $i = 1, 2$.
This distance is smaller than the real distance between two polyhedra, but automatically calculated once G_i and $R_{\max,i}$ are known. The radius $R_{\max,i}$, $i = 1, 2$, are constant, whereas the position of the center G_i, $i = 1, 2$, evolves in time. The components of G_i at $t \in [t_l, t_u]$ is given, as mentioned in the previous sections, by the linear interpolation:

$$G_i(t) = b(t)G_i(t_l) + (1 - b(t))G_i(t_u),\ \forall t \in [t_l, t_u],$$

with $b(t) = \frac{t_u - t}{t_u - t_l}$ and $G_i(t_l), G_i(t_u)$ derive from (4)-(5).
In the narrow phase, the exact distance between the robots is computed. Two main methods exist for distance calculation. The first method is Gilbert, Johnson and Keerthi's algorithm published in 1988 [9] and referred as *GJK*. This algorithm computes the Hausdorff distance of the Minkowski difference $R_2 - R_1$ from the origin. The second method is Lin and Canny's algorithm [16, 17]. This algorithm tracks the closest pair of features between the polyhedra, where the features of a polyhedron are its vertices, its edges and its faces located on its boundary. Several extensions of both approaches exist such as *Enhanced GJK* [3], *I-collide* [5] and *V-Clip* [20]. We choose to follow Lin and Canny's algorithm since the approach is fast, easy to implement and perfectly suited when polyhedra move slightly between two time steps, as it is the case with the robots R_1 and R_2.
In two dimensions, the features of a convex polyhedron are the vertices and the edges of the polyhedron. As described in [15], the vertices of the polyhedron are given in counterclockwise order. Let m be number of vertices of the polyhedron. Let E_i be the edge going from the vertex P_i to $P_{mod(i,m)+1}$ for $1 \leq i \leq m$, where $mod(i, m)$ means i modulo m. For simplicity, we write the vertex $P_{mod(i,m)+1}$ as $P_{fo(i)}$, where fo is the following transformation

$$\begin{array}{rcl} fo : \{1, \ldots, m\} & \rightarrow & \{1, \ldots, m\} \\ b & \mapsto & fo(i) = mod(i, m) + 1. \end{array}$$

More generally, the edge E_i, $i = 1, \ldots, m$ is an open subset defined as follows

$$E_i \;=\; \{x \in \mathbb{R}^2 \,|\, x \;=\; P_i + k\, e_i,\; k \;\in\; (0,1)\,\},$$

where $e_i = \begin{pmatrix} e_{i,x} \\ e_{i,y} \end{pmatrix} = \overrightarrow{P_i P_{fo(i)}}$. Hence, the vertices P_i and $P_{fo(i)}$ do not belong to the edge.
With this definition, the interior of the convex polyhedron is always located on the left side of the edges as illustrated in Figure 6-(a). Finally, let n_i be the outward normal vector to edge E_i whose components stemmed from the vector e_i as follows

$$n_i \;=\; \begin{pmatrix} e_{i,y} \\ -e_{i,x} \end{pmatrix},\; \forall i = 1, \ldots, m.$$

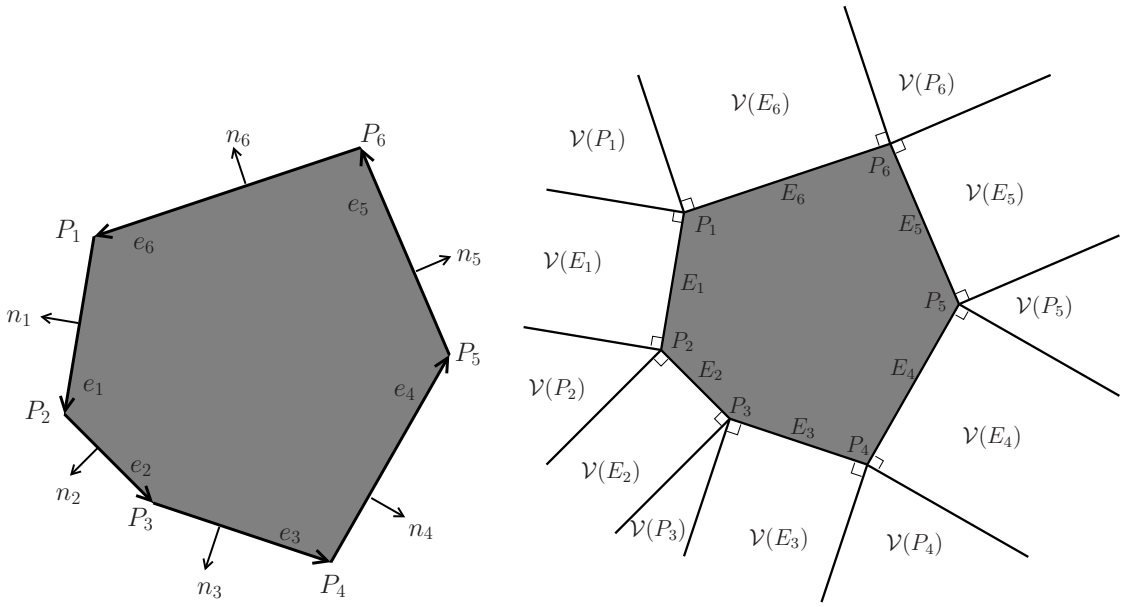


Figure 6: (a) The vertices P_i, $i = 1, \ldots, 6$, of the polyhedron are given in counterclockwise order. The edges E_i, $i = 1, \ldots, 6$, are generated by the direction vector e_i, which connects the tail P_i to the head $P_{fo(i)}$ of the edge. The vectors n_i, $i = 1, \ldots, 6$, are the outward normal vectors to the edges. (b) The Voronoi region of the vertices P_i and of the edges E_i, $i = 1, \ldots, 6$.

Let f_k denote a feature of a convex polyhedron R. With each feature, a Voronoi region is associated. The Voronoi region of f_k is the set of points that are located outside R and closer to f_k than any other features of R. If $\mathcal{V}(f_k)$ denotes the Voronoi region of f_k, then we have:

$$\mathcal{V}(f_k) \;=\; \{x \in \mathbb{R}^2 \setminus R \,|\, d(x, f_k) \;\leq\; d(x, f_j),\; \forall j \;\neq k\},$$

where the distance function is defined as: $d(x, f) \;=\; \inf\{\|x - a\|_2 \,|\, a \in\; f\}$. In two dimensions, the Voronoi region of a vertex P_i of R is the area contained between the half-lines that start from P_i and are perpendicular to the edges for which P_i is an endpoint. The Voronoi region of an edge E_i is the region located above the edge and between the half-lines that start from the endpoints of the edge E_i and are perpendicular to E_i. Hence,

the Voronoi regions form a partition of $\mathbb{R}^2 \backslash R$. An illustration of the Voronoi region of all features in polyhedron R is given in Figure 6-(b).
The distance between the compact convex polyhedra R_1 and R_2 is

$$d_R(R_1, R_2) = \min_{S_a \in R_1, S_b \in R_2} \|\overrightarrow{S_a S_b}\|_2.$$

Since R_1 and R_2 are compact, the minimum is reached. A pair of points that achieves the minimum is called the *pair of closest points*. This pair may not be unique. Lin and Canny's algorithm tracks the pair of features that contains the closest points. This pair of features is named the *closest pair of features* and found when the following conditions are satisfied:

Theorem 1 *Let R_1 and R_2 be disjoint compact convex polyhedra. Let S_a and S_b be the closest points between the feature f_a of R_1 and f_b of R_2. If $S_a \in \mathcal{V}(f_b)$ and $S_b \in \mathcal{V}(f_a)$, then (S_a, S_b) is the pair of closest points between R_1 and R_2 and (f_a, f_b) is the closest pair of features.*

The proof of this theorem can be found in [16]. Lin and Canny's algorithm starts with an initial pair of features and loops on the following three steps until the conditions in Theorem 1 are fulfilled:

1. compute the pair of closest points (S_a, S_b) between the features,
2. test if $S_a \in \mathcal{V}(f_b)$,
3. test if $S_b \in \mathcal{V}(f_a)$.

If both tests succeed, then Theorem 1 implies that the pair (S_a, S_b) is the closest pair of points and the distance between R_1 and R_2 is equal to $\|\overrightarrow{S_a S_b}\|_2$. On the contrary, if $S_a \notin \mathcal{V}(f_b)$, then there exists a feature f_b' of R_2 that is closer to S_a than f_b. The pair (f_a, f_b') is the new candidate pair and the method continues with the first step of the loop, i.e. the computation of the pair of closest points between f_a and f_b'. Similarly, if $S_b \notin \mathcal{V}(f_a)$, there exists a feature f_a' of R_1 that is closer to S_b than f_a and a new pair of features is to be tested. Hence, Lin and Canny's algorithm builds a sequence of pair of features whose next candidate is always closer to the previous one. The algorithm stops when Theorem 1 is satisfied or a collision is detected. A sketch of the algorithm is given in Figure 7.
In two dimensions, only four kinds of pair of features exist: Vertex-Vertex, Vertex-Edge, Edge-Vertex and Edge-Edge. In the next subsections, we present our two-dimensional version of Lin and Canny's algorithm. We establish how to check if a point belongs to the Voronoi region of a feature and how to choose the next pair of features.

5.1 Case Vertex-Vertex

Let us start with the first case, where the pair of features is the vertices (P_a, P_b). An illustration is given in Figure 8, where the pair of vertices is (P_5, Q_1). In this first case, the pair of closest points between the features is naturally the pair of features itself. The next step in the loop of Lin and Canny's algorithm is to check if $P_a \in \mathcal{V}(P_b)$. By definition, the Voronoi region of $\mathcal{V}(P_b)$ is the area between the half-lines starting at P_b and perpendicular to the preceding edge, $E_{mod(b-2,m)+1}$, and the following edge, E_b, of the

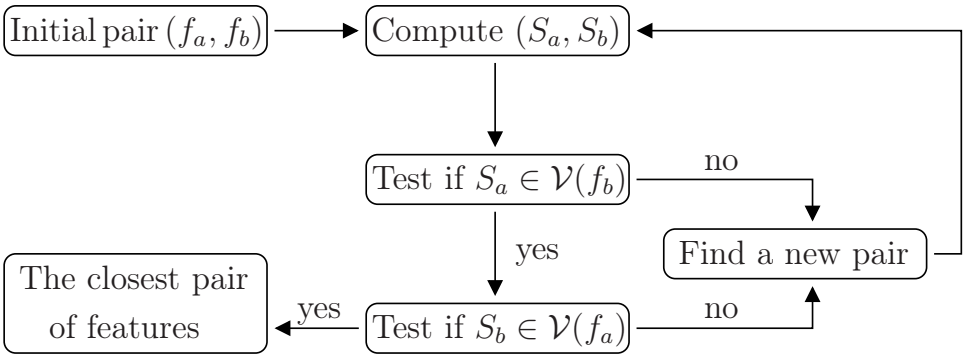


Figure 7: Lin and Canny's algorithm.

vertex P_b (compare Figure 6-(a)). For simplicity, we write the preceding edge as $E_{pr(b)}$, where pr is the following transformation

$$\begin{aligned} pr \,:\, \{1,\ldots,m\} &\rightarrow \{1,\ldots,m\} \\ i &\mapsto pr(i) = mod(i-2,m)+1. \end{aligned}$$

The point P_a belongs to $\mathcal{V}(P_b)$ if and only if P_a is located in the lower half-plane $\pi(P_b, e_b)$ generating by the outward normal vector e_b and the point P_b, and in the lower half-plane $\pi(P_b, -e_{pr(b)})$.
Let us consider the orthogonal projection of P_a onto the line $\ell(e_b, P_b)$ defined by the direction vector e_b and the point P_b. By definition, the orthogonal projection, noted S, is given by

$$S \,=\, P_b \,+\, \alpha_b\, e_b, \quad \text{with } \alpha_b \,=\, \frac{\overrightarrow{P_bP_a}\cdot e_b}{\|e_b\|_2^2}. \tag{14}$$

If the projection coefficient $\alpha_b \leq 0$, then the projection S is located before P_b on the line $\ell(e_b, P_b)$. If $\alpha_b \in (0,1)$, then the projection is between P_b and $P_{fo(b)}$, and if $\alpha_b \geq 1$, then S is located after $P_{fo(b)}$ on the line $\ell(e_b, P_b)$. Consequently, the point P_a belongs to $\pi(P_b, e_b)$ if and only if its projection coefficient is negative. Similarly, the point P_a belongs to $\pi(P_b, e_{pr(b)})$ if and only if the projection of P_a on the line $\ell(e_{pr(b)}, P_{pr(b)})$ is located after the point P_b. In other words, $P_a \in \pi(P_b, e_{pr(b)})$ if and only if the projection coefficient is larger than 1. Therefore, $P_a \in \mathcal{V}(P_b)$ when:

$$\alpha_b = \frac{\overrightarrow{P_bP_a}\cdot e_b}{\|e_b\|_2^2} \leq 0 \;\text{ and }\; \alpha_{pr(b)} = \frac{\overrightarrow{P_{pr(b)}P_a}\cdot e_{pr(b)}}{\|e_{pr(b)}\|_2^2} \geq 1. \tag{15}$$

To avoid numerical problems, the above inequalities are relaxed as follows

$$\alpha_b \leq \delta_c, \;\text{ and }\; \alpha_{pr(b)} \geq 1-\delta_c. \tag{16}$$

where δ_c is a small positive parameter.
Similarly, we check if the vertex P_b belongs to the Voronoi region $\mathcal{V}(P_a)$. If so, the pair of features (P_a, P_b) is the closest pair. In Figure 8-(a), we can observe that the vertex P_5 belongs to the lower half-planes $\pi(Q_1, e_1)$ and $\pi(Q_1, -e_3)$. We deduce that $P_5 \in \mathcal{V}(Q_1)$. In Figure 8-(b), the vertex Q_1 does not belong to $\mathcal{V}(P_5)$ since Q_1 is located above the half-plane $\pi(P_5, -e_4)$. The second inequality in (16) is therefore violated. In fact, Q_1 is closer

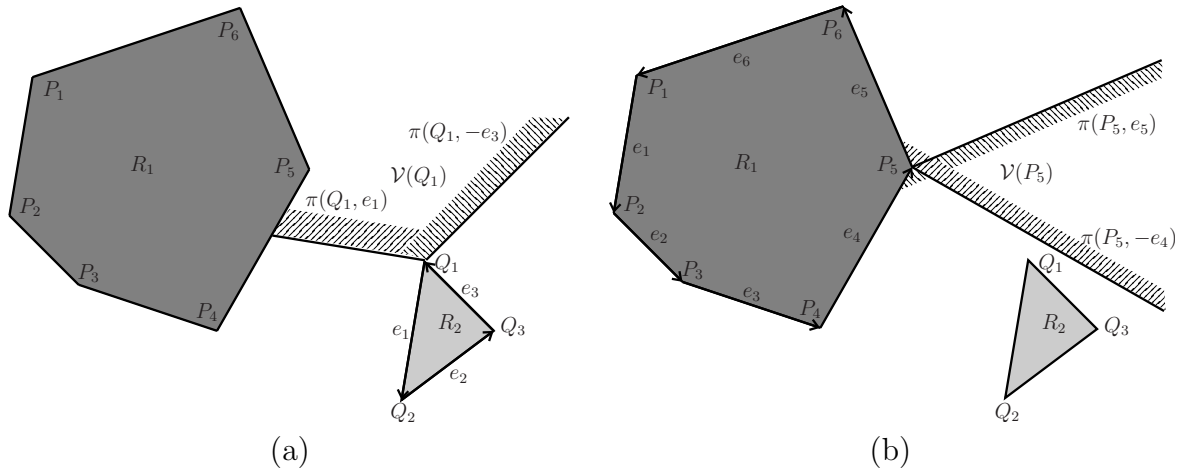


Figure 8: The pair of features (P_5, Q_1) is considered. (a) The vertex P_5 belongs to $\mathcal{V}(Q_1)$ since P_5 is located in the lower half-planes $\pi(Q_1, e_1)$ and $\pi(Q_1, -e_3)$. (b) The vertex Q_1 does not belong to $\mathcal{V}(P_5)$ since Q_1 is located above the lower half-plane $\pi(P_5, -e_4)$. The vertex Q_1 is actually closer to the edge e_4 than to the vertex P_5.

to the feature e_4 than to P_5. The pair of features is updated to (E_4, Q_1), the violated condition in (16) indicating the new feature to consider. The algorithm to handle the case Vertex-Vertex is summarized in Table 2 where the projection coefficients α_a, $\alpha_{pr(a)}$, α_b and $\alpha_{pr(b)}$ are computed according to (14) and (15).

Table 2: Algorithm "Vertex(P_a)-Vertex(P_b)"

```
If α_b > δ_c then
   return new pair (P_a, E_b)
Else if α_pr(b) < 1 − δ_c then
   return new pair (P_a, E_pr(b))
Else
   If α_a > δ_c then
     return new pair (E_a, P_b)
   Else if α_pr(a) < 1 − δ_c then
     return new pair (E_pr(a), P_b)
   Else
     return (P_a, P_b) is the closest pair
   End if
End if
```

5.2 Case Edge-Vertex

Let us assume that the feature E_a is an edge of R_1 and the feature P_b is a vertex of R_2. To apply Theorem 1, one needs to compute first the closest points S_a and S_b between the pair of features (E_a, P_b). Then, $S_a \in \mathcal{V}(P_b)$ and $S_b \in \mathcal{V}(E_a)$ are checked. The former

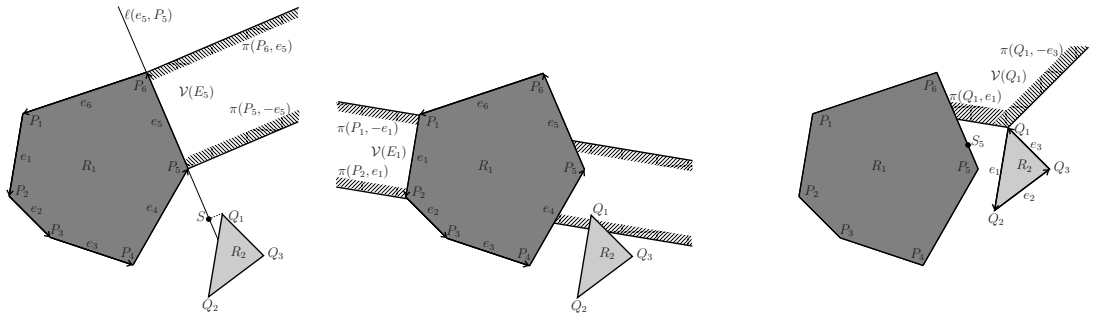
relation is verified thank to (16), whereas the point S_b belongs to $\mathcal{V}(E_a)$ if

$$S_b \text{ lies in the lower half-plane } \pi(P_a, -e_a), \tag{17}$$
$$S_b \text{ lies in the lower half-plane } \pi(P_{fo(a)}, e_a), \tag{18}$$
$$S_b \text{ is above the edge } E_a. \tag{19}$$

These three conditions follow directly from the definition of the Voronoi region of an edge. Since P_b is a vertex, the point S_b is equal to P_b. For the point S_a, one needs to compute the orthogonal projection of S_b onto the line $\ell(e_a, P_a)$ defined by the direction vector e_a and the point P_a. By definition, the orthogonal projection, noted S, is given by

$$S = P_a + \alpha\, e_a, \quad \text{with } \alpha = \frac{\overrightarrow{P_a S_b} \cdot e_a}{\|e_a\|_2^2}. \tag{20}$$

If $\alpha \leq 0$, then the projection S is located before P_a on the line $\ell(e_a, P_a)$. This means that the vertex P_b is closer to the vertex P_a than to the edge E_a. Hence, the pair of features (P_a, P_b) is a closer pair than (E_a, P_b). This case is illustrated in Figure 9-(a), where P_b is Q_1 and E_a is the edge E_5. With the help of the figure, we can notice that $\alpha \leq 0$ also means that P_b lies above the lower half-plane $\pi(P_a, -e_a)$. Consequently, the vertex P_b cannot belong to $\mathcal{V}(E_a)$. The case, $\alpha \leq 0$ implies that $P_b \notin \mathcal{V}(E_a)$ and the new candidate pair of features is (P_a, P_b). The case $\alpha \geq 1$ is treated similarly. It follows that $P_b \notin \mathcal{V}(E_a)$ and the new candidate pair is $(P_{fo(a)}, P_b)$.



(a) (b) (c)

Figure 9: (a) The pair (E_5, Q_1) is considered. $Q_1 \notin \mathcal{V}(E_5)$ since Q_1 is above $\pi(P_5, -e_5)$. (b) The pair (E_1, Q_1) is considered. $Q_1 \notin \mathcal{V}(E_1)$ since Q_1 is beneath E_1. (c) The pair (E_5, Q_1) is considered. $Q_1 \in \mathcal{V}(E_5)$, but $S_5 \notin \mathcal{V}(Q_1)$.

If $\alpha \in (0, 1)$, then the projection lies in E_a and S_a is set to S. This case also implies that the vertex P_b is located between the lower half-planes $\pi(P_a, -e_a)$ and $\pi(P_{fo(a)}, e_a)$. Therefore, the first two relations for P_b to be an element of $\mathcal{V}(E_a)$ are checked (compare with (17)-(19)). The last relation involves verifying whether P_b lies above the edge E_a. Let n_a be the outward normal vector to E_a. Then P_b lies above E_a if

$$\overrightarrow{P_a P_b} \cdot n_a > 0. \tag{21}$$

If this inequality is not satisfied, then P_b does not belong to $\mathcal{V}(E_a)$ and a new pair of features must be found. This new pair is composed of P_b and the closest feature of R_1 to P_b, so that the new pair is closer in distance than the previous one. Let us see how to

find such a feature. Since the Voronoi regions define a partition of $\mathbb{R}^2 \setminus R_1$, P_b lies either in the Voronoi region of an edge of R_1 or in the Voronoi region of a vertex of R_1. On one hand, P_b lies in the Voronoi region of the edge E_i of R_1 if and only if

- the projection of P_b on E_i lies in E_i, i.e.:

$$\alpha_{E_i} \in (0, 1), \tag{22}$$

- and P_b lies above the line containing E_i, i.e.:

$$\overrightarrow{P_i P_b} \cdot n_i > 0. \tag{23}$$

On the other hand, P_b lies in the Voronoi region of the vertex P_i of R_1 if and only if

- the projection parameter of P_b onto $E_{pr(i)}$, whose endpoint is P_i is larger than 1, i.e:

$$\alpha_{E_{pr(i)}} \geq 1, \tag{24}$$

- and the projection parameter of P_b onto E_i, whose startpoint is P_i is negative, i.e.:

$$\alpha_{E_i} \leq 0. \tag{25}$$

Therefore, to determine in which Voronoi region the point P_b belongs to, we enumerate the edges on R_1 until an edge satisfies (22)-(23) or two successive edges satisfy (24)-(25). Figure 9-(b) illustrates such a case. The original pair of features is (E_1, Q_1). The point Q_1 does not belong to $\mathcal{V}(E_1)$ since Q_1 is beneath E_1. The new pair of features is (E_4, Q_1), E_4 being the first edge of R_1 that verifies (22)-(23).
It may happen that no edge satisfies (22)-(23) and no pair of successive edges verifies (24)-(25). This case means that the point P_b lies inside the polyhedron R_1. Thus, a collision between the robots is detected. The algorithm to find the feature that is the closest to P_b is summarized in Table 3.
Suppose now that $P_b \in \mathcal{V}(E_a)$. The last check of the case Edge-Vertex is to verify whether $S_a \in \mathcal{V}(P_b)$. This test corresponds to the case Vertex-Vertex since S_a and P_b are both vertices. Thus, if $S_a \notin \mathcal{V}(P_b)$, the feature P_b is replaced by E_b or $E_{pr(b)}$ depending on which inequality in (16) is violated. In Figure 9-(c), an example is represented where $Q_1 \in \mathcal{V}(E_5)$, but the orthogonal projection $S_5 \notin \mathcal{V}(Q_1)$. The first inequality in (16) is here not satisfied. Then, the new candidate pair is (E_5, E_1).
As in the case Vertex-Vertex, we relax the tests on the value of α and of (21) with the positive parameter δ_c to avoid numerical problems. Eventually, the case Edge-Vertex is summarized in Table 4.

5.3 Case Edge-Edge

Let us consider the pair of features (E_a, E_b) where E_a is an edge of R_1 and E_b is an edge of R_2. The case when the edges are parallel must be set apart from the non-parallel case.

Table 3: Algorithm "Find the Closest Feature to P_b"

```
Compute α_pr = (P_m P_b · e_m) / ||e_m||_2^2
For I = 1, ..., m
    Compute α = (P_I P_b · e_I) / ||e_I||_2^2
    If α ∈ [δ_c, 1 − δ_c] and P_I P_b · n_I > δ_c ||n_I||_2 ||P_I P_b||_2 then
      return new pair (E_I, P_b)
    Else if α_pr > 1 − δ_c and α < δ_c then
      return new pair (P_I, P_b)
    Else
      Set α_pr = α
    End if
End for
return collision
```

Table 4: Algorithm "Edge (E_a) - Vertex (P_b)"

```
Compute α = (P_a P_b · e_a) / ||e_a||_2^2
If α > 1 − δ_c then
    return new pair (P_fo(a), P_b)
Else if α < δ_c then
    return new pair (P_a, P_b)
Else
    If P_a P_b · n_a < δ_c ||n_a||_2 ||P_a P_b||_2 then
      f_a ← Find the Closest Feature to P_b
      return new pair (f_a, E_b)
    Else
      Compute S_a = P_a + α e_a
      If S_a ∉ π(P_b, e_b) then
        return new pair (E_a, E_b)
      Else if S_a ∉ π(P_b, e_pr(b)) then
        return new pair (E_a, E_pr(b))
      Else
        return (E_a, P_b) is the closest pair
      End if
    End if
End if
```

5.3.1 Parallel edges

Let us assume here that the edges E_a and E_b are parallel. Two possibilities exist. Either the edges overlap. In that case, there are several pairs of points that achieve the minimal distance between the edges. The edges may be the closest pair of features. Or, the edges do not overlap. Then, there exists a pair of vertices that are closer in distance than the pair (E_a, E_b).
To determine if the edges overlap, the smallest edge is projected onto the largest one. Let us assume that $\|e_b\|_2 \leq \|e_a\|_2$. Then, the vertices P_b and $P_{fo(b)}$ are projected onto E_a. Following (14), the projection coefficient are given by

$$\alpha_b = \frac{\overrightarrow{P_aP_b} \cdot e_a}{\|e_a\|_2^2} \quad \text{and} \quad \alpha_{fo(b)} = \frac{\overrightarrow{P_aP_{fo(b)}} \cdot e_a}{\|e_a\|_2^2}. \tag{26}$$

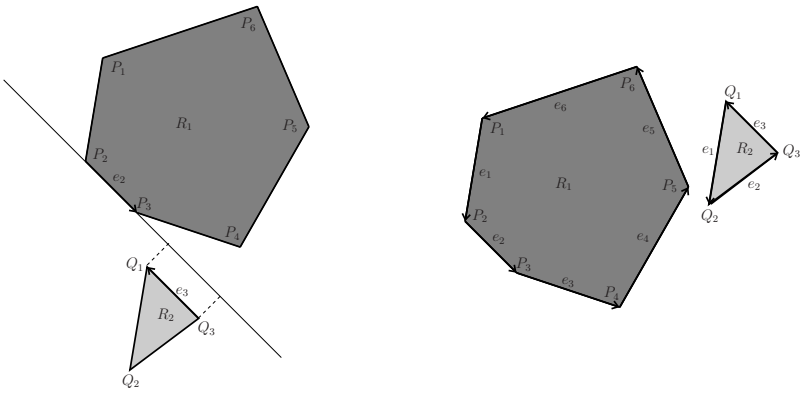
The edges do not overlap when $\alpha_b, \alpha_{fo(b)}$ are both negative or both larger than 1. In that case, a new pair of features must be determined. If $\alpha_b, \alpha_{fo(b)}$ are both negative, then E_b is closer to P_a than to $P_{fo(a)}$. Consequently, the first component of the new pair is P_a. For the second component, the distance between P_a and the vertices of E_b is tested. The new pair of features is

$$\begin{cases} (P_a, P_b), & \text{if } \|\overrightarrow{P_aP_b}\|_2 \leq \|\overrightarrow{P_aP_{fo(b)}}\|_2, \\ (P_a, P_{fo(b)}), & \text{otherwise.} \end{cases}$$

Similarly, if $\alpha_b, \alpha_{fo(b)}$ are both larger than 1, the new pair of features is equal to

$$\begin{cases} (P_{fo(a)}, P_b), & \text{if } \|\overrightarrow{P_{fo(a)}P_b}\|_2 \leq \|\overrightarrow{P_{fo(a)}P_{fo(b)}}\|_2, \\ (P_{fo(a)}, P_{fo(b)}), & \text{otherwise.} \end{cases}$$

An illustration is given in Figure 10-(a). The edge E_2 of R_1 is parallel to E_3 of R_2. The edge E_3 is projected onto E_2 and the resulting projection coefficients α_1, α_3 are both larger than 1. The next pair of features is the pair of vertices (P_3, Q_1) since the vertex P_3 is closer to P_1 than to Q_3.



(a) (b)

Figure 10: (a) The edges E_2 of R_1 and E_3 of R_2 are parallel and do not overlap. The pair (P_3, Q_1) is closer than (E_2, E_3). (b) The edges E_1 of R_1 and E_1 of R_2 are parallel and overlap, but (E_1, E_1) is not the closest pair since E_1 of R_2 is beneath E_1 of R_1.

The edges overlap when at least one of the projection coefficients is in $(0,1)$. All the points in the overlapping region achieve the minimum distance between the edges. However, the edges are not necessarily the closest pair of features. The edges are the closest pair, if each edge is located above the other edge. For instance, let us consider the pair of edges (E_1, E_1) in Figure 10-(b). These edges are parallel and overlap. However, they are not the closest pair of features, the edge E_1 of R_2 being located beneath E_1 of R_1.
Let us assume that $\alpha_b \in (0,1)$. The point S_a is defined as the projection of P_b onto E_a, i.e. $S_a = P_a + \alpha_b \overrightarrow{P_a P_{fo(a)}}$. The pair of points (S_a, P_b) achieves the minimum distance between the edges, that is, S_b is equal to P_b. According to (21), the edge E_a is above E_b if and only if the following inequality is satisfied

$$\overrightarrow{P_b P_a} \cdot n_b > 0.$$

If the inequality is not satisfied, then a new pair of features is searched. This new pair is such that the feature on R_1 is the closest to P_b. The pair is obtained by applying the algorithm "Find Closest Feature to P_b". Similarly, we check if E_b is above E_a. If E_b is not above, then we look for the feature on R_2 that is the closest to S_a. The algorithm to handle the case Edge-Edge with paralle edges is sketched in Table 5.

5.3.2 Non-parallel edges

Let us consider here the non-parallel case. If the edges are not parallel, then this pair of features can not be the closest pair. At least one vertex of the edges achieves the minimum distance between the edges. The goal of this section is then to find the next pair of features. This pair is composed by the features that contain the points minimizing the distance between the edges. So, let us compute the minimum distance.
Let us consider the lines that support the edges E_a and E_b:

$$\begin{aligned} \ell(e_a, P_a) &= \{x \in \mathbb{R}^2 \,|\, x = P_a + k_a\, e_a,\ k_a \in \mathbb{R}\}, \\ \ell(e_b, P_b) &= \{x \in \mathbb{R}^2 \,|\, x = P_b + k_b\, e_b,\ k_b \in \mathbb{R}\}. \end{aligned}$$

Let d_E be the distance function between any pair of points (S_a, S_b), with $S_a = P_a + k_a\, e_a \in \ell(e_a, P_a)$ and $S_b = P_b + k_b\, e_b, \in \ell(e_b, P_b)$. The function d_E is defined as follows:

$$\begin{aligned} d_E : \mathbb{R} \times \mathbb{R} &\rightarrow [0, +\infty[\\ (k_a, k_b) &\mapsto d_E(k_a, k_b) = \|P_a + k_a\, e_a - P_b - k_b\, e_b\|_2^2. \end{aligned}$$

The function d_E can be rewritten as

$$\begin{aligned} d_E(k_a, k_b) &= (P_a + k_a\, e_a - P_b - k_b\, e_b)^T (P_a + k_a\, e_a - P_b - k_b\, e_b), \\ &= \|e_a\|_2^2\, k_a^2 - 2\, e_a^T e_b\, k_a k_b + \|e_b\|_2^2\, k_b^2 + 2\, e_a^T \overrightarrow{P_b P_a}\, k_a - 2\, e_b^T \overrightarrow{P_b P_a}\, k_b \\ &\quad + \|\overrightarrow{P_b P_a}\|_2^2. \end{aligned}$$

Hence, d_E is a quadratic function in k_a and k_b. Furthermore, the Hessian matrix of d_E is positive definite. Indeed, the Hessian matrix is given by

$$H(d_E)(k_a, k_b) = 2 \begin{pmatrix} \|e_a\|_2^2 & -e_a^T e_b \\ -e_a^T e_b & \|e_b\|_2^2 \end{pmatrix}.$$

Table 5: Algorithm "Edge (E_a) - Edge (E_b) parallel", when $\|e_b\|_2 \leq \|e_a\|_2$

```
Compute α_b and α_{fo(b)} as in (26)
If α_b < δ and α_{fo(b)} < δ then
    If ‖P_a P_b‖_2 ≤ ‖P_a P_{fo(b)}‖_2 then
      return new pair (P_a, P_b)
    Else
      return new pair (P_a, P_{fo(b)})
    End if
Else if α_b > 1 − δ and α_{fo(b)} > 1 − δ then
    If ‖P_{fo(a)} P_b‖_2 ≤ ‖P_{fo(a)} P_{fo(b)}‖_2 then
      return new pair (P_{fo(a)}, P_b)
    Else
      return new pair (P_{fo(a)}, P_{fo(b)})
    End if
Else
    If α_b ∈ [δ, 1 − δ] then
      Set S_a = P_a + α_b P_a P_{fo(a)} and S_b = P_b
    Else
      Set S_a = P_a + α_{fo(b)} P_a P_{fo(a)} and S_b = P_{fo(b)}
    End if
    If P_a P_b · n_a > δ‖n_a‖_2 ‖P_a P_b‖_2 then
      If P_b P_a · n_b > δ‖n_b‖_2 ‖P_b P_a‖_2 then
        return (E_a, E_b) is the closest pair
      Else
        f_b ← Find the Closest Feature to S_a
        return new pair (E_a, f_b)
      End if
    Else
      f_a ← Find the Closest Feature to S_b
      return new pair (f_a, E_b)
    End if
End if
```

On one hand, we have $\|e_a\|_2^2 > 0$ since the edges are supposed to be not degenerate. On the other hand, the determinant of the Hessian matrix is

$$\det H(d_E)(k_a, k_b) \;=\; 4(\|e_a\|_2^2\|e_b\|_2^2 - (e_a^T e_b)^2) \;=\; 4\|e_a\|_2^2\|e_b\|_2^2(1-\cos(\beta)),$$

where β is the angle between e_a and e_b. Since the edges are not parallel, $\cos(\beta) < 1$ and we deduce

$$\det H(d_E)(k_a, k_b) > 0.$$

Sylvester's criterion implies that $H(d_E)(k_a, k_b)$ is positive definite. Consequently, the quadratic function d_E is convex.
By assumption, the lines $\ell(e_a, P_a)$ and $\ell(e_b, P_b)$ are not parallel. Therefore, the lines are secant and the minimum distance is equal to 0. This minimum is global since d_E is convex and quadratic. The minimum is given by solving

$$\nabla d_E(k_a, k_b) = \begin{pmatrix} 2\,\|e_a\|_2^2\,k_a - 2\,e_a^T e_b\,k_b + 2\,e_a^T\overrightarrow{P_bP_a} \\ 2\,\|e_b\|_2^2\,k_b - 2\,e_a^T e_b\,k_a - 2\,e_b^T\overrightarrow{P_bP_a} \end{pmatrix} \;=\; \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

The solution of the above linear system is

$$k_a^* \;=\; \frac{(e_a^T e_b)\,e_b^T\overrightarrow{P_bP_a} \;-\; \|e_b\|_2^2\,e_a^T\overrightarrow{P_bP_a}}{\|e_a\|_2^2\,\|e_b\|_2^2 \;-\; (e_a^T e_b)^2}, \tag{27}$$

$$k_b^* \;=\; \frac{\|e_a\|_2^2\;e_b^T\overrightarrow{P_bP_a} \;-\; (e_a^T e_b)\,e_a^T\overrightarrow{P_bP_a}}{\|e_a\|_2^2\,\|e_b\|_2^2 \;-\; (e_a^T e_b)^2}. \tag{28}$$

The denominator of the above fractions is always positive since it is equal to the quarter of the determinant of the Hessian matrix, which is positive definite.
In conclusion, the lines $\ell(e_a, P_a)$ and $\ell(e_b, P_b)$ crosses at $P_a + k_a^*\,e_a \;=\; P_b + k_b^*\,e_b$ and it follows

$$d_E(k_a^*, k_b^*) = 0.$$

We look for the pair of points that minimizes the distance between the edges E_a and E_b. Therefore, the minimum is no longer searched in $\mathbb{R}^2$, but only in $[0,1]^2$. This minimum is obtained by solving the following problem

$$\begin{array}{lll} (Pd_E) & \min & d_E(k_a, k_b) \\ & \text{s.t.} & 0 \le\; k_a \;\le 1, \\ & & 0 \le\; k_b \;\le 1. \end{array}$$

Let $(\bar{k}_a, \bar{k}_b)$ be the solution of (Pd_E). If $(\bar{k}_a, \bar{k}_b)$ lies in $(0,1)^2$, then the edges intersect and a collision is detected. Indeed, $(\bar{k}_a, \bar{k}_b) \in (0,1)^2$ means that the minimum of (Pd_E) is achieved by points located in the edges. Since the edges are not parallel and we work in a two-dimensional space, these points necessarily coincide.
If, on the other hand, $(\bar{k}_a, \bar{k}_b) \notin (0,1)^2$, then the solution of (Pd_E) is on the boundary of $[0,1]^2$. To find this solution, let us consider the location of the global optimum (k_a^*, k_b^*) in $\mathbb{R}^2$. The value of d_E at (k_a^*, k_b^*) is null. Furthermore, the contour lines of d_E are ellipses centered at (k_a^*, k_b^*). The value of the contour lines increases as one moves away from (k_a^*, k_b^*).
Now, two cases must be distinguished, according to the values of k_a^* and k_b^*:

- case 1: either k_a^* or k_b^* is in $[0,1]$,
- case 2: k_a^* and k_b^* are both outside $[0,1]$.

Let us start with case 1. An instance of this case is $k_a^* > 1$ and $k_b^* \in [0,1]$. For this instance, there exists a contour line that is tangent to the line $k_a = 1$. Let P_I be the intersection point and let k_b^I be the second coordinate of P_I, that is $P_I = (1, k_b^I)$. By definition of the contour lines, the value of d_E along $k_a = 1$ is such that

$$d_E(1, k_b^I) \, < \, d_E(1, k_b), \; \forall k_b \in \mathbb{R} \setminus \{k_b^I\}.$$

If k_b^I lies in $[0,1]$, then $(1, k_b^I)$ is the solution of (Pd_E). If k_b^I is larger than 1, then P_I does not belong to $\partial[0,1]^2$. Since the value of d_E increases from P_I to 0 along the line $k_a = 1$, the minimum of (Pd_E) is located at $(1,1)$. Similarly, if k_b^I is smaller than 0, the minimum of (Pd_E) occurs at $(1,0)$.
An illustration of the contour lines is given in Figure 11-(a). This example corresponds to the pair of edges (E_4, E_1) given in Figure 10-(b). The x-axis is the value of k_a and k_b is represented in the y-axis. The intersection point P_I is located on the boundary of $[0,1]^2$. P_I achieves then the minimum of (Pd_E).
From a mathematical point of view, the solution $(\bar{k}_a, \bar{k}_b)$ of (Pd_E) is computed in the following manner. Because $k_a^* > 1$, $\bar{k}_a$ is set to 1. The problem now is to find a value for $\bar{k}_b$. The variable $\bar{k}_b$ is found by solving

$$\min_{0 \leq k_b \leq 1} g_1(k_b) \tag{29}$$

where

$$g_1(k_b) \, = \, d_E(1, k_b) \, = \, \|e_b\|_2^2 \, k_b^2 - 2 \, e_b^T (e_a + \overrightarrow{P_b P_a}) \, k_b + (e_a + \overrightarrow{P_b P_a})^2.$$

The function g_1 is a second-order polynomial. The minimum is obtained at the root of the first derivative of g_1:

$$g_1'(k_b) \, = \, 0 \; \Leftrightarrow \; k_b \, = \, \frac{e_b^T (e_a + \overrightarrow{P_b P_a})}{\|e_b\|_2^2}.$$

Thus, the intersection point P_I is $(1, k_b^I)$, where k_b^I is given by the above expression. The minimum of (Pd_E) is then obtained at

$$(\bar{k}_a, \bar{k}_b) \, = \, \begin{cases} (1,0) & \text{if} \quad k_b^I \, < \, 0, \\ (1, k_b^I) & \text{if} \quad k_b^I \, \in \, [0,1], \\ (1,1) & \text{if} \quad k_b^I \, > \, 1. \end{cases}$$

The pair $(\bar{k}_a, \bar{k}_b)$ defines the pair of points $(P_a + \bar{k}_a e_a, P_b + \bar{k}_b e_b)$. These points indicate the next pair of features. For instance, if $P_a + \bar{k}_a e_a$ is in the edge E_a, then the next feature is E_a. If, in the other hand, $P_a + \bar{k}_a e_a$ is an endpoint of the edge, then the next feature is the endpoint itself. Finally, the next pair of features is defined as follows

$$\begin{cases} (P_{fo(a)}, P_b) & \text{if} \quad (\bar{k}_a, \bar{k}_b) \, = \, (1,0), \\ (P_{fo(a)}, E_b) & \text{if} \quad (\bar{k}_a, \bar{k}_b) \, = \, (1, k_b^I), \\ (P_{fo(a)}, P_{fo(b)}) & \text{if} \quad (\bar{k}_a, \bar{k}_b) \, = \, (1,1). \end{cases}$$

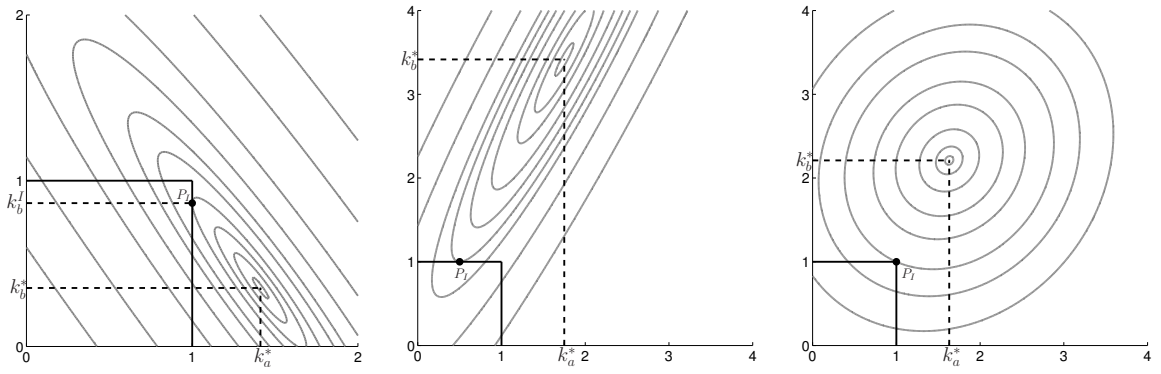


Figure 11: Contour lines of the minimum distance d_E for the pairs (a) (E_4, E_1), (b) (E_5, E_3) and (c) (E_3, E_1) represented in Figure 10-(b). For these examples, the point P_I achieves the minimum of (Pd_E).

The other instances of case 1 are $(k_a^*, k_b^*) \in]-\infty, 0[\times[0,1]$, $(k_a^*, k_b^*) \in [0,1]\times]1, +\infty[$ and $(k_a^*, k_b^*) \in [0,1]\times]-\infty, 0[$. They are handled similarly. For example, the instance $(k_a^*, k_b^*) \in]-\infty, 0[\times[0,1]$ is solved by setting $\bar{k}_a$ to 0 and $\bar{k}_b$ to the solution of

$$\min_{0 \le k_b \le 1} g_0(k_b) \;=\; d_E(0, k_b).$$

Now, let us consider case 2. This case contains the instances $k_a^*, k_b^* > 1$, $k_a^*, k_b^* < 0$, $k_a^* > 1, k_b^* < 0$ and $k_a^* < 0, k_b^* > 1$. Let us study the first instance, i.e. $k_a^*, k_b^* > 1$. The Figures 11-(b) and 11-(c) illustrate such a case. Figure 11-(b) corresponds to the pair (E_5, E_3). The contour line hits first the segment line $[0,1] \times \{1\}$. The minimum is then reached at $(\bar{k}_a, 1)$ with $\bar{k}_a \in (0,1)$. The next candidate pair of features is (E_5, P_1). Figure 11-(c) illustrates the contour lines of the distance function d_E for the pair (E_3, E_1) in Figure 10-(b). The contour line hits the corner $(1,1)$ of the square $[0,1]^2$. Hence, $(1,1)$ achieves the minimum of d_E on $[0,1]^2$ and the next candidate pair is (P_4, P_2).
Mathematically, the minimum of (Pd_E) is computed in a similar way to case 1. The additional difficulty here is how to determine which from $\bar{k}_a$ and $\bar{k}_b$ must be initially set to 1. One possibility is to evaluate the sign of the partial derivatives at the corner $(1,1)$. However we prefer to simply apply the above computations to $\bar{k}_a = 1$, then to $\bar{k}_b = 1$ and compare the resulting pairs of points. More precisely, let us first set the parameter $\bar{k}_{a,1}$ to 1. We associate here the index 1 to indicate that we compute the first pair. Then, by solving (29), the pair $(\bar{k}_{a,1}, \bar{k}_{b,1})$ is obtained. To this pair of parameters corresponds the pair of points $(P_a + \bar{k}_{a,1} e_a, P_b + \bar{k}_{b,1} e_b)$.
In a second step, the parameter $\bar{k}_{b,2}$ is set to 1 and $\bar{k}_{a,2}$ is the solution of the following minimization problem

$$\bar{k}_{a,2} = \arg \min_{0 \le k_a \le 1} d_E(k_a, 1).$$

The second pair of points is $(P_a + \bar{k}_{a,2} e_a, P_b + \bar{k}_{b,2} e_b)$. The minimum of (Pd_E) occurs at the closest pair of points, i.e. at

$$(\bar{k}_a, \bar{k}_b) \;=\; \begin{cases} (\bar{k}_{a,1}, \bar{k}_{b,1}) & \text{if } d_E(\bar{k}_{a,1}, \bar{k}_{b,1}) \le d_E(\bar{k}_{a,2}, \bar{k}_{b,2}), \\ (\bar{k}_{a,2}, \bar{k}_{b,2}) & \text{otherwise.} \end{cases} \tag{30}$$

The next pair of features is deduced from $(\bar{k}_a, \bar{k}_b)$, as previously:

$$\begin{cases} (P_a + \bar{k}_a e_a, P_b + \bar{k}_b e_b) & \text{if} \quad \bar{k}_a, \bar{k}_b \in \{0,1\}, \\ (E_a, P_b + \bar{k}_b e_b) & \text{if} \quad \bar{k}_a \in (0,1),\ \bar{k}_b \in \{0,1\}, \\ (P_a + \bar{k}_a e_a, E_b) & \text{if} \quad \bar{k}_a \in \{0,1\},\ \bar{k}_b \in (0,1). \end{cases}$$

We can observe that the next pair of features always contains at least one vertex. The case Edge-Edge when the edges are not parallel is summarized in the algorithm of Table 6.

Table 6: Algorithm "Edge (E_a) - Edge (E_b) not parallel"

```
Compute k_a^* and k_b^* according to (27)-(28)
If (k_a^*, k_b^*) ∈ [0,1]^2 then
   return collision
Else if k_a^* ∈ [0,1] then
   If k_b^* > 1 then
      Set k̄_b = 1 and k̄_a = arg min_{0≤k_a≤1} d_E(k_a, 1)
   Else
      Set k̄_b = 0 and k̄_a = arg min_{0≤k_a≤1} d_E(k_a, 0)
   End if
Else if k_b^* ∈ [0,1] then
   If k_a^* > 1 then
      Set k̄_a = 1 and k̄_b = arg min_{0≤k_b≤1} d_E(1, k_b)
   Else
      Set k̄_a = 0 and k̄_b = arg min_{0≤k_b≤1} d_E(0, k_b)
   End if
Else
   Compute (k̄_a, k̄_b) as in (30)
End if
If k̄_a, k̄_b ∈ {0,1} then
   return new pair (P_a + k̄_a e_a, P_b + k̄_b e_b)
Else if k̄_a ∈ (0,1) then
   return new pair (E_a, P_b + k̄_b e_b)
Else
   return new pair (P_a + k̄_a e_a, E_b)
End if
```

5.4 Convergence

The convergence of Lin and Canny's algorithm is guaranteed since the new pair of features is always closer in distance than the previous pair. Furthermore, the test algorithms "Vertex (P_a) - Vertex (P_b)","Edge (E_a) - Edge (E_b)","Edge (E_a) - Edge (E_b) parallel" and "Edge (E_a) - Edge (E_b) not parallel" are constant in time. Only the convergence of the algorithm "Find the closest feature to P_b is of order m, the number of vertices in the robot. In the worst case, the $4m^2$ possible pairs of features are visited. Consequently, Lin and Canny's algorithm is $O(m^3)$. However, the order of the method can be drastically reduced, when the initial pair is well chosen. In our particular application, the robots

move slightly between two consecutive time steps t_k and t_{k+1}. In this case, a good initial pair at t_{k+1} is the closest pair of features found at t_k. The closest pair at t_{k+1} is then either the initial pair itself or a neighboring pair, that is obtained by applying the constant in time test algorithms.

6 Numerical results

The collision detection algorithm is applied to the two-dimensional work-cell presented in Figure 12. The work-cell is composed of four obstacles (black quadrilaterals) and two robots R_1 and R_2. The robot R_1 is depicted at several time steps by a light grey square. The robot starts from V_I^1, follows the light grey trajectory and ends at V_F^1 (black triangle). The markers on the trajectory indicate the position of the center of gravity of R_1 at the time steps τ_k, $k = 1, \ldots, n$. Similarly, the robot R_2 is the dark grey square and its associated trajectory is the dark grey curve. The robot R_2 moves from V_I^2 to V_F^2.

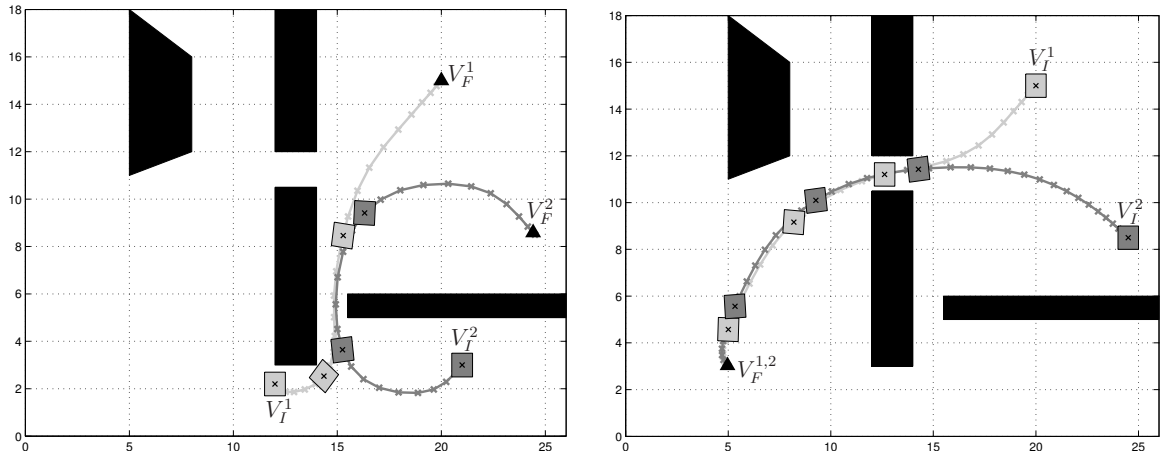


Figure 12: A two-dimensional work-cell. The black quadrilaterals are the obstacles. The grey squares are the robots, represented at several time steps. The robot R_1 is light grey and the robot R_2 is dark grey. The associated trajectories follow the same color code. The black diamonds are the end point of the trajectories. (a) the trajectories are collision-free. (b) the robots intersect just before reaching the end position $V_F^{1,2}$.

For the numerical example given in Figure 12-(a), the collision detection algorithm correctly returns that the robots do not intersect. The example is composed of 52 time intervals of the form $[t_l, t_u]$. Only one interval is split into two subintervals while checking Schwarzer et al.'s inequality. The upper bound η of the distance between the robots is mostly computed in the broad phase, even if the robots are close to each other while travelling along the trajectories. The narrow phase is called only 7 times. For these 7 instances, only 14 test algorithms of Lin and Canny's method are applied.
In the example in Figure 12-(b), both robots have the same end point, denoted by $V_F^{1,2}$. Hence, the robots collide. Moreover, the trajectories are such that the robots follow each other very closely. With these trajectories, the detection algorithm must consider 71 time intervals $[t_l, t_u]$. The collision is detected while checking Schwarzer et al.'s inequality for the 56^{th} time interval. The robots at the end time of this interval are depicted in

Figure 12-(b). We can observe that the robots intersect just before reaching $V_F^{1,2}$. Before the collision is detected, about 30 time intervals and subintervals are bisected. Lin and Canny's algorithm is called 9 times and the test algorithms are applied 12 times.
These two examples illustrate the good running of our collision detection algorithm. The algorithm returns the good solution and always almost instantaneously. Indeed, the CPU time is equal to $4 \cdot 10^{-3}\, s$ for the first example, and to $5 \cdot 10^{-3}\, s$ for the second example. Lin and Canny's algorithm is also well adapted to our two-dimensional examples since the number of calls to the test algorithms is small. Finally, the few calls to the narrow pahse shows the good estimation of the lower bound λ_1 and λ_2 in Schwarzer et al.'s inequality. We conclude, that the collision detection algorithm is perfectly suited for the second step of the iterative method to solve (WCP).

7 Conclusions

An algorithm to detect a collision between robots moving along specified trajectories has been presented. This algorithm is used to solve the work-cell problem (WCP) described in [13]. Therefore, the robots are two-dimensional convex polyhedra.
The collision detection algorithm follows the dynamic method developed by Schwarzer et al. in [24]. The method involves comparing lower bounds of the distance travelled by the robots during a time interval, with an upper bound of the distance between the robots. We have established a formula of the lower bounds that perfectly suits the robots and the trajectories issued from (WCP). To compute the upper bound of the distance, when the robots are close to each other, we have adapted Lin and Canny's algorithm to our two-dimensional case. The resulting algorithm was applied to the second step of the iterative method to solve (WCP). Numerical examples have shown the efficiency of the algorithm: a correct output is given almost instantaneously.
These good results encourage us to use the same strategy for a three-dimensional work-cell that contains industrial robots. However, due to the complexity of the geometry of such robot, a bounding volume hierarchy method must be included to compute the right hand side of Schwarzer et al.s inequality efficiently.

References

[1] J. W. Boyse. Interference detection among solids and surfaces. *Commun. ACM*, 22(1):3–9, 1979.

[2] S. Cameron. A study of the clash detection problem in robotics. In *Int. Conf. Robotics & Automation*, pages 488–493, 1985.

[3] S. Cameron. Enhancing GJK: Computing minimum and penetration distances between convex polyhedra. In *Proceedings of International Conference on Robotics and Automation*, pages 3112–3117, 1997.

[4] J. Canny. Collision detection for moving polyhedra. Technical report, Massachussetts Institute of Technology, October 1984.

[5] J. D. Cohen, M. C. Lin, D. Manocha, and M. K. Ponamgi. I-collide: An interactive and exact collision detection system for large-scaled environments. In *Symposium on Interactive 3D Graphics*, pages 189–196. ACM Siggraph, April 1995.

[6] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Journal of Algorithm*, 6, issue 3:381–392, 1985.

[7] M. Gerdts, R. Henrion, D. Hömberg, and C. Landry. Path planning and collision avoidance for robots. *Numerical Algebra, Control and Optimization*, 2(3):437 – 463, 2012.

[8] E.G. Gilbert and S.M. Hong. A new algorithm for detecting the collision of moving objects. In *IEEE Proc. Int. Conf. Robotics Automat.*, volume 1, pages 8–14, 1989.

[9] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, April 1988.

[10] S. Gottschalk, M. C. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In ACM SIGGRAPH, editor, *Computer Graphics Proceedings, Annual Conference Series*, 1996.

[11] P. M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15:179–210, 1996.

[12] P. Jiménez, F. Thomas, and C. Torras. 3d collision detection: A survey. *Computers and Graphics*, 25:269–285, April 2000. Preprint submitted to Elsevier Preprint.

[13] C. Landry, R. Henrion, D. Hömberg, M. Skutella, and W. Welz. Task assignment, sequencing and path-planning in robotic welding cells. In *Methods and Models in Automation and Robotics (MMAR), 2013 18th International Conference on*, pages 252–257, 2013.

[14] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha. Fast distance queries with rectangular swept sphere volumes. In *Proc. of IEEE Int. Conference on Robotics and Automation*, pages 3719–3726, 2000.

[15] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

[16] M. C. Lin. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1993.

[17] M. C. Lin and J. F. Canny. A fast algorithm for incremental distance calculation. *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, page 1008, 1991.

[18] M. C. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *In Proc. of IMA Conference on Mathematics of Surfaces*, pages 37–56, 1998.

[19] K. Maruyama. A procedure to determine intersections between polyhedral objects. *International Journal of Computer & Information Sciences*, 1:255–266, 1972.

[20] B. Mirtich. V-clip: Fast and robust polyhedral collision detection. Technical report, Mitsubishi Electronics Research Laboratory, 1997.

[21] B. Mirtich. Efficient algorithms for two-phase collision detection. In K. Gupta and A.P. del Pobil, editors, *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, pages 203–223. Wiley, New York, 1998.

[22] S. Quinlan. Efficient distance computation between nonconvex objects. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 3324–3329, 1994.

[23] S. Redon, A. Kheddar, and S. Coquillart. An algebraic solution to the problem of collision detection for rigid polyhedral objects. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation, ICRA 2000, April 24-28, 2000, San Francisco, CA, USA*, pages 3733–3738. IEEE, 2000.

[24] F. Schwarzer, M. Saha, and J. Latombe. Adaptive dynamic collision checking for single and multiple articulated robots in complex environments. *IEEE Tr. on Robotics*, 21:338–353, 2005.

[25] P. Toth and D. Vigo. The vehicle routing problem. pages 1–26. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.