Path Curve Optimization for Driven Robot in Matlab® Software

Radek DOSKOČIL[a] and Thibault TALHOUËT[b]

Abstract

The paper deals with the path planning for the drive robots. The paper presents the method of finding locally optimal path with respect to some physical conditions. The polyline and circular segments represent the path. Our main tool of solving this task is Matlab® software. Thanks to this Matlab scripts we found a trajectory more suited to drive robots and faster. Used with an adapted trajectory planning method, this algorithm can be useful for finding a path between two points. Depending on the stability constraints of the robot the speed can be adapted in turns, looking at the centrifugal force that undergoes the robot. Nevertheless, the process of erasing unnecessary points is still largely perfectible. Being the basis of the optimization process, it is necessary that this erasure is optimal, that is to say that there remains only the strict number of points necessary to go from one point to another.

Key words: Path Planning, Trajectory Optimization, Road Map Method, Cell Decomposition Method, Driven Robot, Matlab®.

1. Introduction

There are many applications for military robots. Robots are increasingly automated and this requires controlling the movement of these. Many trajectory planning methods have been created but they are not necessarily suitable for differential driven robots. Our objective was to optimize the trajectory for this type of robot.

The vehicle locomotion is what enables the robot to move. It transfers the energy given by the motors to a kinetics energy. There are three major means of locomotion: continuous tracks, wheels and legs. E.g. [1] deals more precisely with this robot components.

The path planning is a process that aims at finding a path from a start point to a goal point avoiding the obstacles that are in the environment. This process raises two problems. The first one is that we are not sure to find a collision admissible free path. This problem is called the “decision problem”. The second one is the computation of the path, also known as the “complete problem” [2].

Path planning has many applications: artificial intelligence in video games, robotic surgery, architectural design and robot motion. Taking the example of the robot motion, the difficulty of such a process seems obvious. Indeed, it is quite difficult to find a path from one point to another considering the inner motion constraints of the robot, its dimensions, the obstacles and the uncertainties.

The path can be found by various methods depending on the data that we have. Indeed, it is not the same problem if we know, not know or partially know the environment the robot will deal with. For the mobile robots the problem is well documented. We find three major categories for path planning of ground robots: the road map method, the cell decomposition method and the potential field method [2].

The environment is thus a key data in this process. It is defined by a map that can contain known or/and unknown obstacles. To simplify the finding of the path from one point to another the process could either discretise the map into small areas or maps or convert the map into a continuous field. For example, the potential field method converts the map into a continuous field. It is said that the potential field method is continuous path planning method. On the contrary, the road map method discretise the map into a small area and so it is said to be a discrete path planning method.

[a] Department of Military Robotics, University of Defence, Brno, Czech Republic, e-mail: radek.doskocil@unob.cz
[b] Écoles de Saint-Cyr, Coëtquidan, France, e-mail: thib.talhouet@hotmail.fr

As soon as the process has created the map, an adapted search algorithm look for a feasible path and then the right command are sent to the robot.

The aim of this presented work is to find the best way for driven robot to move from a start point to an end position, with consider some robots limitation, e.g. some max. acceleration. The main tool of solving this task is Matlab® software.

Motivation to use Matlab was to use a tool that is used for teaching purposes at universities and to involve students in solving defined tasks.

The constraints were that we have to rely on the paths obtained by planning methods already known and adapt such trajectory for differential drive robots. This implies respecting the kinematic constraints of such robots.

The optimization obtained transforms the path formed by a succession of segments into a succession of segments and arcs. This new path is faster and better match a robot of the differential drive type.

2. Problem Formulation

We assume that the environment is entirely known and that the obstacles are rectangular. We also assume that the motors used to change their velocity instantaneously. We will only study the case of the road map method and the cell decomposition method. In the case of those two methods, the path obtained is a series of nodes and lines the robot must follow. Is this the shortest path possible? Is this the best-adapted path for differential drive robots? [2, 3]

If the robot follows the lines created, it will consume a lot of energy. In fact, when the robot arrives on a node, it will have to do three steps. First the robot will have to stop, then to turn on site in order to be on the right direction and then to advance. Furthermore the multiplication of tasks the robot has to do multiplies the number of errors the robot could do. We have to give to the robot a more fluent path. The approach was to transform a series of segments in a series of segments and arcs to have the most fluent path possible.

3. Base functions created

In text of chapter 3 was generally used sources [4 - 10]. The downloaded information has been included in these links. The specific using is indicated in the text directly.

3.1. The map functions

The map functions such as “mapsquare” aim at creating a map with rectangles (Fig. 1). This was the first map we were working on. All the rectangles created are created thanks to the Matlab function Rectangle. This function return an array with all the properties of the rectangle. Thanks to this array, we are able to easily find the position of one rectangle, for example. The map functions create a vector with the array of the entire rectangle created. This vector is the output of the map functions and so could be used in the main script.

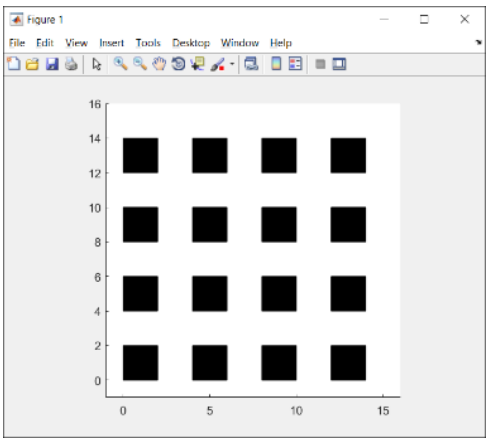


Fig. 1. First map created

3.2. Function “angle”

This function calculate the angle θ between two vectors different from zero, AB and AC, by using the scalar product and the determinant of those vectors to get the sine of the angle. As shown in the Fig. 2 aside if the point C is in the same side of the line than C’ then theta will be positive, if it is in the same side as C“ then theta will be negative.

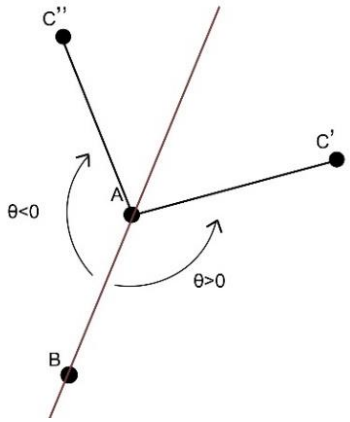


Fig. 2. Sign of the angle

Indeed the scalar product is defined by:

$$\overrightarrow{AB} \cdot \overrightarrow{AC} = AB \cdot AC \cdot \cos\left(BAC\right). \quad (1)$$

And if $\overrightarrow{AB}=(x_1,y_1)$ and $\overrightarrow{AC}=(x_2,y_2)$ then

$$\overrightarrow{AB}\cdot\overrightarrow{AC}=x_1\cdot x_2+y_1\cdot y_2. \quad (2)$$

However, this is not enough because the function arccosine is not a bijection. We need to have the sinus too. The determinant of AB and AC is defined by:

$$\det\left(\overrightarrow{AB},\overrightarrow{AC}\right)=(x_1\cdot y_2-x_2\cdot y_1)=AB\cdot AC\cdot\sin\left(BAC\right). \quad (3)$$

The angle obtained is contained between $[-\pi;\pi]$.

3.3. *Function "distance"*

This function gives the distance between two points.

3.4. *Function "dessiner_arc"*

This function draws the arc between two points, M1 and M2, knowing the coordinates of these two points and the coordinates of the center of the circle, O. M3 is the image of the point O translated by the vector (4,0).

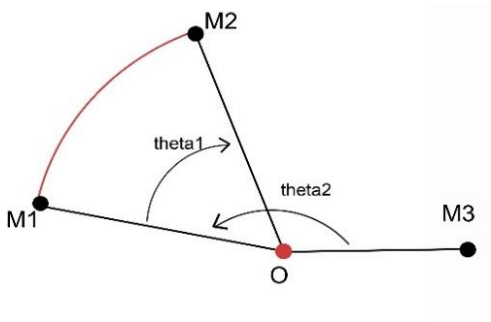


Fig. 3. Method for drawing arc

To do that we used the Cartesian parametric equation of the circle (written below) whom center is $O(x_0,y_0)$. The parameter is the angle, and it varies to one value to another in order to draw the right arc.

$$\begin{cases}x=x_0+R\cdot\cos(\theta)\\ y=y_0+R\cdot\sin(\theta)\end{cases},\qquad \theta\in[0,2\pi]. \quad (4)$$

In order to draw the right arc, we set $\theta=\theta_2+k\cdot\theta_1$, where k vary from 0 to 1 and where

$\theta_1=\left(\overrightarrow{OM3},\overrightarrow{OM2}\right)$ and $\theta_2=\left(\overrightarrow{OM1},\overrightarrow{OM2}\right)$. The function returns the length of the arc.

3.5. *Function "distrd"*

This function aims at giving the distance between one line defined by two points A and B and a rectangle, assuming that the line does not go through the rectangle. The distance between two sets is defined to be the shortest distance between one point of the first set and one point of the other. However, it seems very hard to find the shortest distance between those two sets. In fact, the problem is easier than that the shortest distance between a line and a rectangle is the shortest distance between the line and each corner of the rectangle. A distance between one point and a line is easy to calculate: it is the distance between this point and its orthogonal projection on the line.

The function "distrd" evaluates the four distances and returns the shortest one. It also returns the point M that is at the two third of the distance from the orthogonal projection as shown in the picture aside, see Fig. 4.

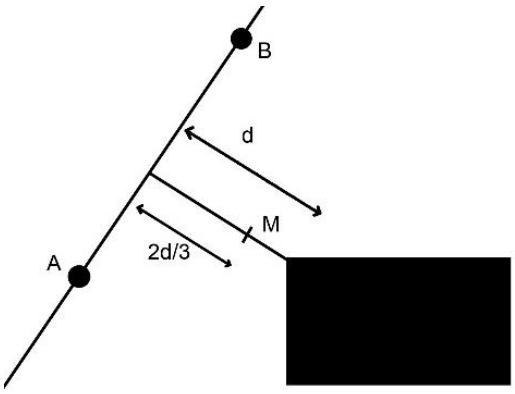


Fig. 4. Illustration of the "distrd" function

3.6. *Function "distdp"*

This function looks alike the latest one. Nonetheless, this function only evaluates the distance between one point V and a line defined by two points. Thus between V and its orthogonal projection on the line, H. It returns the distance and the point M which is at a distance of two third of the distance between V and H from H, alike the point M in the latest function.

3.7. *Function "eqdroite"*

This function returns the coefficients a and b that verify $y=a\cdot x+b$ for two points given $A(x_A,y_A)$ and $B(x_B,y_B)$.

3.8. *Function "centre"*

Let's assume that we know the coordinate of two points A and B and an equation of a line D that goes through B. We consider the circle that is tangent to (AB) in A and which centre is on the line D. h is the perpendicular to AB in A. Given that the circle is tangent to AB in A, the centre is on the intersection between h and D. The function returns the coordinate of such centre. Note that if we put a third point C such as D is the bisector of ABC then the circle is tangent also to (BC).

3.9. *Function "velocity"*

This function returns the variation of the angular rotation of a wheel in function of the time using the relations [2, 3]

$$\omega_L = \frac{v}{r}\cdot\left(1-\frac{L}{2\cdot R}\right) \text{ and } \omega_R = \frac{v}{r}\cdot\left(1+\frac{L}{2\cdot R}\right). \quad (5)$$

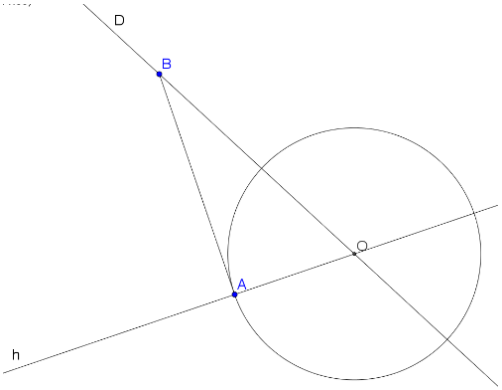


Fig. 5. Geometrical construction of the centre

4. First approach

4.1. *Description of the optimization*

The aim of our optimization is to transform a series of segments into a series of arcs and segments. The arcs must replace every turn on site, so they have to be created for each two successive segments and have to be tangent to them. In order to do that the algorithm takes every three successive points and transform the two successive segments that the points form in a segment, "an appropriate curve" and a segment.

The arcs must check some condition:

Firstly, we have seen that in a turn the differential drive robot is submitted to the centrifugal force. Yet, the centrifugal force is proportionate to the square of the speed and the inverse of the radius. As much as we can, we do not want to influence the velocity of the robot. Thus, we have to optimise the radius in finding the greater radius we can.

Secondly, the path must be smooth and continuous. That is why the arcs must be tangent to the arcs that is before and after it. We will see that this includes another condition on the arc. The third condition is that the arc must not collide with any obstacles on the map. To understand what the appropriate curve is, let is consider three points P, N and Q from the plane and a rectangle. We assume that the segment PN and NQ does not go through the rectangle, such a condition is ensured by the restriction to the road map and cell decomposition methods of my studies.

We draw the centre of the circle that goes through the point P and that is tangent to PN and NQ. So the orthogonal projection of its centre on the line (PN) is the point P. Given that a circle that is tangent to two lines has its centre on the bisector of those two lines, the centre is easy to find. We named it O1 and called the corresponding circle C.

4.2. *First definition of the no collision constraint*

This was the first definition I set to avoid a collision. We will see further that this solution is not adapted to all circumstances.

The part of the plane that interests us is the set of point S defined by all the points $M(r,\theta)$ with $r\rangle 0$ and $\theta \in \left[0,\left(\overrightarrow{NP},\overrightarrow{NQ}\right)\right]$ where $\theta = \left(\overrightarrow{NP},\overrightarrow{NQ}\right)$ and r is the distance NM. The set is drawn in orange on the picture aside, see Fig. 6.

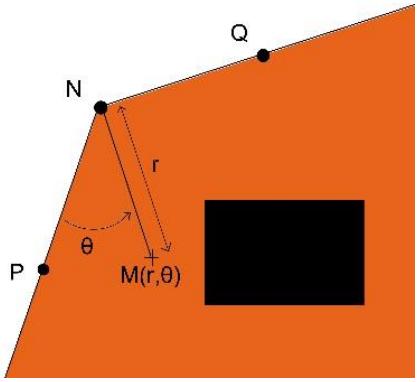


Fig. 6. Domain of study of the first condition

We restrained our domain of study to every rectangle that is contained in this set. To find the rectangle that might collide with the arc we want to create, that is to say the closest rectangle, we just have to evaluate all the distances between the point N and the centre of each rectangle. The closest rectangle is the one whom centre is the closest to N.

We have looked for the closest rectangle in order to find a point where the circle must go through the closest possible to this rectangle.

We apply the function "distrd" presented in the part 3.5, to the closest rectangle that we found and each segment NP and NQ. We obtain two points that are very close to the rectangle and we keep the one that is the most restrictive. That is to say the point that corresponds to the shortest distance between the lines (NP) and (NQ) and the rectangle.

Finding the radius of a circle that is tangent to two lines and that goes through one point, M that does not belong to

one of those lines is a known issue. The Fig. 7 illustrates its solution.

First, we find the intersection between the lines (NM) and the circle C (the orange circle on the map). We get the point M' and M". We create the point O1' and O" that are the image of O1 by the homothetic transformation of centre N and with a ratio respectively equal to $\frac{NM'}{NM}$ and $\frac{NM''}{NM}$. The circles whose centre are O1' and O1" and that goes through M are the image of C by the homothetic transformation described earlier, thus they are tangent to the two lines (NP) and (NQ). Furthermore, we have the following expressions:

$$\begin{cases} O1''M = \frac{NO1''}{NO1} \cdot 01M'' \\ O1'M = \frac{NO1'}{NO1} \cdot 01M' \end{cases} . \qquad (6)$$

Thus, we have the radius of the two circles that are tangent to the two lines and that goes through M.

At the end, we only keep the arc that concerns us. That is to say the arc that goes through M and whose convex part contained N. Here this is the arc HG that is drawn in blue on the Fig. 7.

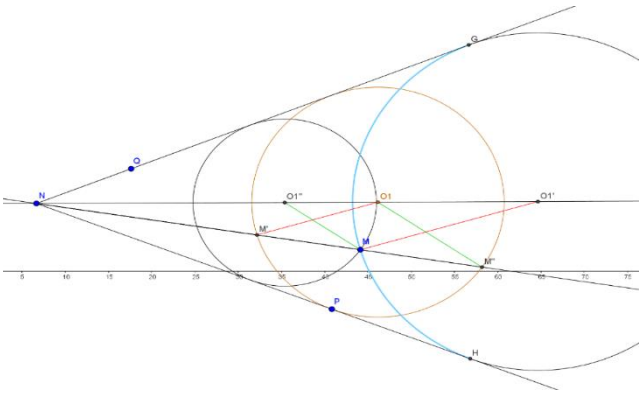


Fig. 7. Solution of the problem of the circle tangent to two lines and passing by one point that does not belong to those lines

4.3. *Second condition: continuous path*

The second condition is that two successive arcs must not intersect or at least in the point where the two arcs are tangent to the common segment. Let is add one point O on the plane such as PN, NQ and QO is an admissible path, and one rectangle as drawn in the picture aside, see Fig. 8.

Thanks to the method presented previously we drew the two arcs HG and IJ, considering successively the ordered set of points {P,N,Q} and {N,Q,O}. In this situation, we will first follow the segments PH and then the arc HG. However, when arriving on G we want to be able to follow the arc IJ unless we go backward following the segment GI.

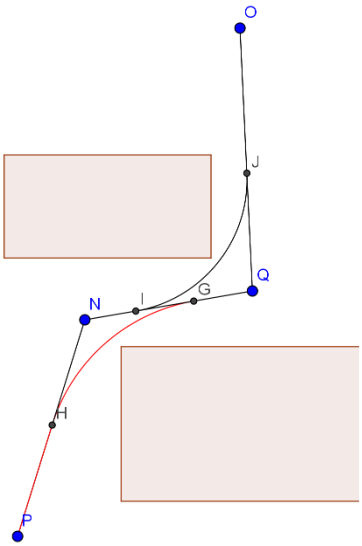


Fig. 8. Illustration of the cause of the second condition

To eliminate the problem we have to set a condition. The condition is that the point where the arc is tangent to the segments should not be more distant from the second point of a set of three ordered points (N for {P,N,Q}) than half of the distance of the respective segment. For the set of points that we considered we obtained the following conditions:

$$NH \leq NP/2 \text{ and } NG \leq NQ/2. \qquad (7)$$

This problem is most common if the length of the segments are too short compared to the distance between the segments and the rectangles. Note that by doing this we also force the points where the arc begins and ends to be contained in the corresponding segments.

5. Limits

5.1. *The set of points*

The major limit of such an algorithm is the configuration of the points in the plane. When using either a cell decomposition method or a road map method we are sure that the series of a point will form a series of successive segments that will be contained in the free space. With a cell decomposition method the points could be set wherever we want in the different cells. In the contrary, the road map methods place the points on a particular situation. Some of the road map method as the visibility graph method could not be used with the algorithm, because the points created are the vertices of the polygonal form.

Moreover, with some road map method we are not sure that the path is the shortest possible using all the points as with the probabilistic road map method. It depends on the 'roads' generated. For example, if the path obtained is the red one on the picture aside (Fig. 9), we would prefer to green one. Somehow, we have to be sure that the green one is in the free path.

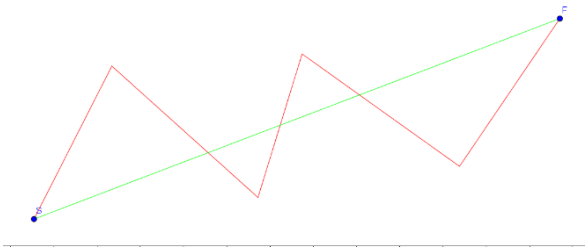


Fig. 9. Illustration of the limit of the point's configuration

We did not manage to find conditions that check if a segment collide or not with an obstacle. Somehow, we manage to erase some of the 'useless' points by evaluating the angle between three successive points. If the angle is near 180 degrees then we erase the second point. The angle has to be contained between 180-t and 180+t where *t* is a value previously fixed, called the tolerance.

5.2. First definition of the non-collision constraint

In the chapter 4 we have restrained the test of the distance to one particular rectangle that is the closest to the point N.

As shown on the Fig. 10, this restriction could not be appropriate. In the drawing the distance NE is shorter than the distance NF. Thus the rectangle that the algorithm will study is the one whose centre is E. The arc that is drawn with the process explained in part 4.1 is the blue arc. We notice that the problem is that the arc collides with the second rectangle.

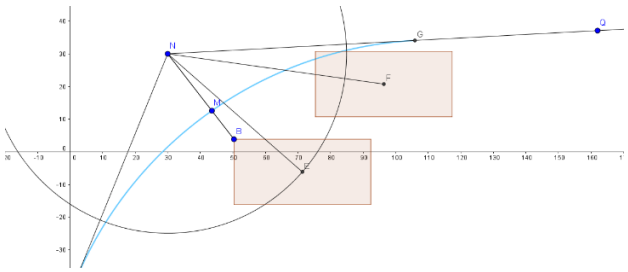


Fig. 10. Limit of the no-collision condition

To avoid such problem the solution would be to not restrain the process to one rectangle but to all the rectangles that are contained in the set S described in part 4.1. The process will have to evaluate the distance between the four corners of all rectangle of the set S and the two lines NP and NQ. The problem is that it means to calculate $8 \cdot n$ distances instead of $n+8$, if n is the number of rectangles.

Furthermore, if we just pick the shortest distance and then apply the rest of the process of the part 4.1 then we could also obtain an arc that collides a second rectangle as shown in the Fig. 11. The segment IB is the shortest distance. We deduce the point M. The arc that goes through M and is tangent to the lines NP and NQ, collides with the rectangle whose centre is F.

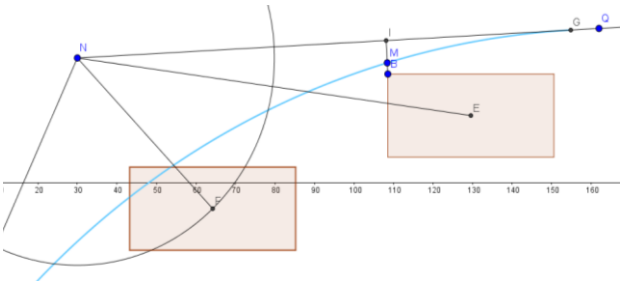


Fig. 11. Counter example of the shortest distance condition

6. Second approach

We have seen that the no collision condition developed in the part 4.1 works in some conditions but not all. We have to find a new approach. Thanks to the second condition found, expressed by the condition (7) we know the largest radius possible. If the arc created with this radius collides with a rectangle then we have to find a shorter radius. Yet how can we know that the arc collide?

6.1. Second definition of the no collision condition

a) <u>Restriction of study</u>

Once again, the only rectangles that could collide with the largest arc are in the set S, described in the part 4.1. This is because the arc that concerns us is an arc that linked the segments NP and NQ and whose convex part contains N. We name A and B the points where respectively the arc (in blue in the Fig. 12) begins and ends.

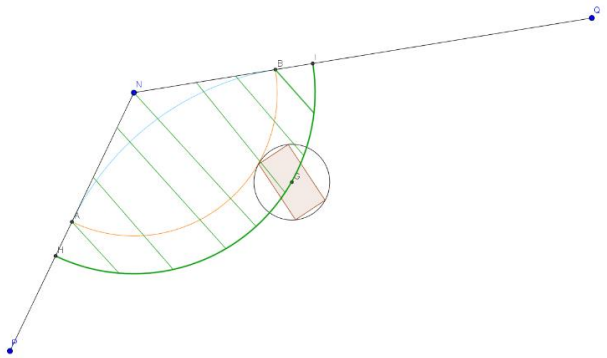


Fig. 12. Representation of the restriction set

If a rectangle collides with the arc thus, it is contained in S and it has at least one point that is contained in the disc with centre N and radius NA=NB. Let is model the rectangles by discs such as their centre are the centre of the rectangles and their radius are half of the length of the diagonal. Thus, the rectangles are fully contained in the discs. Let is consider the diagonal of all the rectangles contained in S, we called $L_{\max}$ the maximum length of those diagonals.

If a rectangle collides with the arc, its centre necessarily verify the following expression:

$$d > L_{\max} + NA, \qquad (8)$$

where d is the distance between the centre and N.

A rectangle that collides with the arc is contained in S and verify the latest condition. So the centre of the rectangle has to be contained in the new set called S' defined by all the points $M(r,\theta)$ with $L_{\max} + NA > r > 0$ and $\theta \in \left[0, \left(\overrightarrow{NP}, \overrightarrow{NQ}\right)\right]$ where $\theta = \left(\overrightarrow{NP}, \overrightarrow{NM}\right)$ and r is the distance NM. The set is hatched in green on the Fig. 12.

b) <u>The new definition</u>

Let is consider the arc AB and O its centre. We name T the part of the plane delimited by the segments AO, OB, BN and NA which is coloured in orange on the Fig. 13. We name T1 the part of T, which is delimited by the segment [OA] and [OB] and the arc AB (the part is hatched in blue on the Fig. 13). We name T2 the part of the plane defined by T\T1.

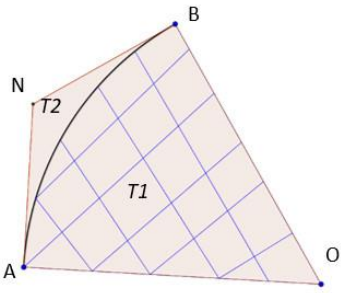


Fig. 13. Representation of the part of the plane

If a rectangle collides with the arc AB then it has at least one point that is contained in T2. If all the points of the rectangle are in T2, which is equal to the four vertices of the rectangle belong to T2, then the rectangle does not collide. However, we will see later that it is not important to consider that case. Given that the rectangles do not collide with NP and NQ, even more so with NA and NB then if a rectangle collide one of its vertex belongs to T2. To conclude, a rectangle that is contained in S' does not collide with the arc if it has no vertex in T2, so if none of its vertices is at a bigger distance from O than the radius of the arc.

6.2. The process

In this part, we will explain the successive steps of the algorithm:

1. The map is charged using one of the maps available.
2. The user puts as many points as he wants on the map by clicking. Note that the segment linking two successive points must be on the free space.
3. The algorithm erases all the "useless" points defined in part 3a.
4. The algorithm picks every three successive points and find the "best" arc possible. The "best" arc is the arc with the largest radius that satisfies the conditions of no-collision and continuous path.
5. The algorithm draws the variation of the angular velocity of each wheel.

To find the "best" arc for the three successive points P, N and Q the algorithm follows the steps that are computed in the subscript "virage":

1. Calculate d the minimum between half of the distance PN and NQ. M is the point on PN that satisfies NM=d.
2. Find the radius, R, and the centre, O, of the arc that is tangent to the two segments and goes through M.
3. Find the point M1 where the circle with radius R and centre O is tangent to NP. Deduce the point M2 where the circle is tangent to NQ.
4. Erase all the rectangles whose centre does not belong to S1.
5. Erase all the vertices of the remaining rectangles that does not belong to T.
6. Find the maximum of the distance between those points and N. The vertex that corresponds to this maximum is called V.
7. Test the value. If the value is inferior to R then we found the 'best' circle. If the value is superior to R then the sub-algorithm continues.
8. Find the minimum of the distance between V and the two segments PN and NQ. H is the orthogonal projection of V on the corresponding segment.
9. The new point M is the point belonging to [HV] that satisfies $HM = 2/3 \cdot HV$. Then sub-algorithm repeats the steps 2 and 3. The appropriate curve is found.
10. The steps 4, 5 and 6 are computed in a subscript called 'procherect'.

7. Results

7.1. A shorter and faster path

The path we wanted to create is a path that is shorter and faster than the path created by the successive segments. We have compared the path obtained by our algorithm and the original path in different situations. The paths obtained are illustrated in the Fig. 14.

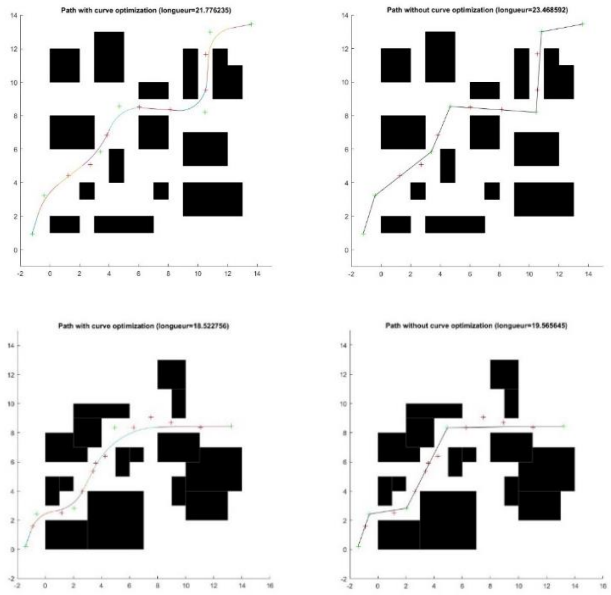
It appears that each time, the shortest path is the path obtained by the optimization algorithm. If we assume that the robot does not have to slow down in the different turns then the path obtained is also the faster. Somehow, this last statement depends on the robot we work with. Indeed the speed in the turn is limited by the threshold value of the centrifugal force, as explained in [2, 3].

7.2. The importance of the tolerance angle

We have set a tolerance angle to erase all the "useless" points, see part 5.1. This tolerance angle has a huge importance as we see on the picture below. First, because it enables us to have larger segments and that directly influence the way the different arcs are drawn. Indeed the larger the segments are the larger the largest radius possible is, cause of the condition presented in the part 4.2b. Besides, it reduces the length of the path.

Furthermore, given that this tolerance angle erases some points, it decreases the number of sets of three successive points that have to be treated by the algorithm.

Somehow, when setting this tolerance angle, we have to take care that the new set of points created could be linked without colliding with any obstacles. The greater the tolerance angle is the more likely collisions will occur.



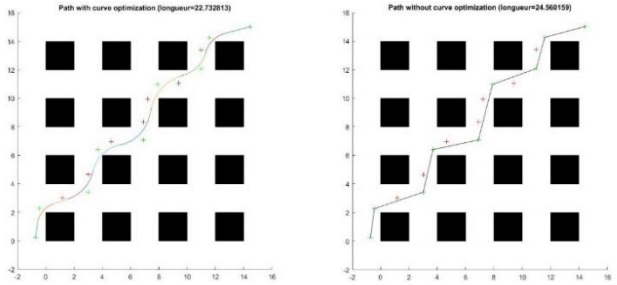


Fig. 14. Comparison between the path with and without optimization using different maps (upper pair - maprec1, middle pair - maprec2, lower pair - mapsquare)

In the Fig. 15, we have two situations with the same set of points in the map maprec2. The drawing on the right is the result of a tolerance angle equal to 0, and on the left the tolerance angle is set to 60. The path obtained in the Fig. 14B was obtained with a tolerance angle equal to 40.

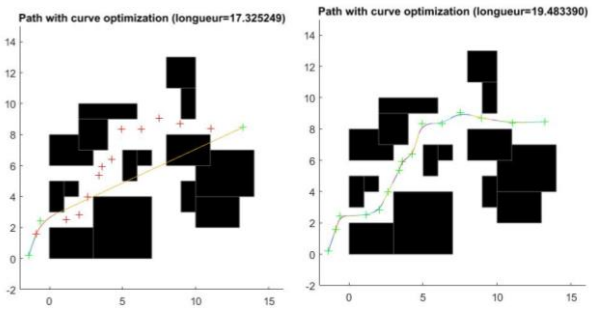


Fig. 15. Influence of the tolerance angle

7.3. The control of the robot

The Fig. 16 shows the variation of the velocity of each wheel for the path obtained with optimization shown on the Fig. 14, upper pair, (the blue curve is for the left wheel and the red one for the right wheel). The dimension used for the robot are L=15 cm, r=5 cm and the constant speed of the robot has been set to 25 cm/s.

We see the problem mentioned in the part 2. It is hard to get a signal response looking like that.

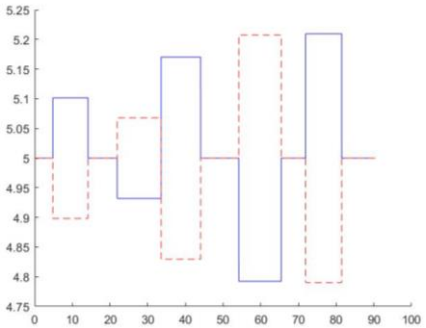


Fig. 16. Variation of the angular velocity of both wheels

8. Conclusion

The aim of our task was to find the best trajectory possible for a differential drive robot, between two points avoiding the obstacles [9, 10].

After having grasped the constraints of such a study, we have chosen to focus on the optimization of a path obtained by a cell decomposition method or a road map method. The result that we got reach the goal we have fixed for ourselves. The algorithm “curve_optimization” find a shorter path and better suited to a drive robot. It has been developed in the case of rectangular obstacles. However, any obstacles could be modelled by rectangles (the more numerous they are the more precise will be the model). The map that we can see on the Fig. 17 illustrates this case.

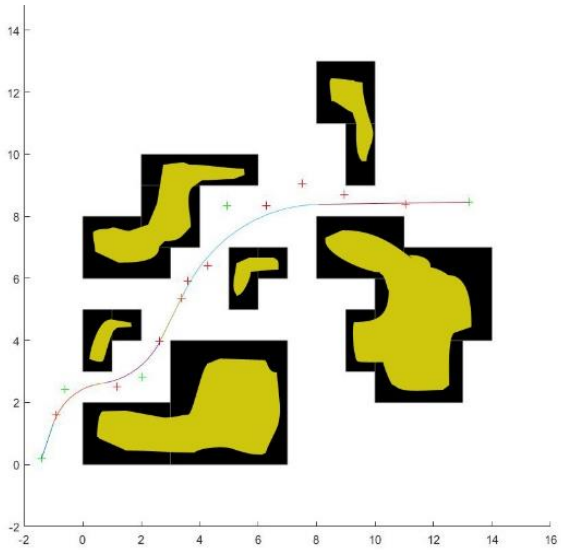


Fig. 17. Modelling of obstacles with rectangles

It was not easy to get such an algorithm.

Firstly, Matlab is a program with a huge number of functions implemented, therefore it needs years to completely master it and to obtain such algorithm we have spent times looking for the solution on the net.

Secondly, the conditions we have were difficult to be mathematically defined. Moreover, the geometric problem we have to face have several solutions, but we have to find the best way to translate it in the Matlab language.

To conclude, the algorithm we have created it is not perfect: the major part that could be improved is the deletion of the “useless” points. The algorithm could be applied to a robot, working jointly with a cell decomposition or a road map method and a path-finding algorithm if we add some things to this algorithm. Indeed the algorithm does not take into account the size of the robot in the no collision detection, neither the speed limitation in a turn and the initial pose of the robot.

Acknowledgements

The work presented in this article has been supported by the Czech Republic Ministry of Defence - University of Defence development program "Research of sensor and control systems to achieve battlefield information superiority".

References

[1] Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). Introduction to autonomous mobile robots. *MIT press*, [Ch. 1-3].

[2] de Wit, C. C., Siciliano, B., & Bastin, G. (Eds.). (2012). Theory of robot control. *Springer Science & Business Media*. G. Bastin (Éds.), [Ch. 7]

[3] Siciliano, B., & Khatib, O. (Eds.). (2008). Springer handbook of robotics. *Springer Science & Business Media*. [Ch. 35]

[4] Kavraki, L. E., Svestka, P., Latombe, J. and Overmars, M. H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces, in *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566-580, Aug. 1996, doi: 10.1109/70.508439

[5] Cook, G.; Zhang, F. Kinematic Models for Mobile Robots, in *Mobile Robots: Navigation, Control and Sensing, Surface Robots and AUVs* , IEEE, 2020, pp.5-12, doi: 10.1002/9781119534839.ch1.

[6] Lingelbach, F. (2005). Path planning using probabilistic cell decomposition, *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, New Orleans, LA, USA, 2004, pp. 467-472 Vol.1, doi: 10.1109/ROBOT.2004.1307193

[7] S.M. LaValle, Planning Algorithms, *Cambridge University Press*, New York, 2006.

[8] Štefek A., Křivánek V., Bergeon Y.T., Motsch J. (2016) Differential Drive Robot: Spline-Based Design of Circular Path. In: Awrejcewicz J. (eds) *Dynamical Systems: Theoretical and Experimental Analysis*. Springer Proceedings in Mathematics & Statistics, vol 182. Springer, Cham. https://doi.org/10.1007/978-3-319-42408-8_26

[9] Krejsa J., Vechet S. (2018) Determination of Optimal Local Path for Mobile Robot. In: Březina T., Jabłoński R. (eds) *Mechatronics 2017. MECHATRONICS 2017. Advances in Intelligent Systems and Computing*, vol 644. Springer, Cham. https://doi.org/10.1007/978-3-319-65960-2_79

[10] J. Krejsa and S. Vechet, "Finding optimal local path for mobile robot with kinematic constraints: How to find the safest path for nonholonomic robot," 2016 17th *International Conference on Mechatronics - Mechatronika (ME)*, Prague, 2016, pp. 1-6.

Appendix A. Functions Script

```
function theta=angle(B,A,C)
% This function calculate the value of the oriented angle between the
% vectors AB and AC using the scalar product and the determinant
AB=B'-A';
AC=C'-A';
Na=realsqrt(AB (1,: )^2+AB (2,:)^2);
Nb=realsqrt(AC( 1 ,:)A2+AC(2,:)^2);
C=(AB(1,:)*AC(1,:)+AB(2,:)*AC(2,:))/(Na*Nb);
S=(AB(1,:)*AC(2,:)-AB(2,:)*AC(1,:));
if (abs(C+1)<10A-6)
theta=pi;
else
theta=sign(S)*acos(C);
end
end
```

```
function O=centre(P,N,D)
% Function that give the coordinate of the centre O of the circle which is
% tangent to PN in P. The centre must belong to the line D
M1=[D(1)-1;P(1)-N(1) P(2)-N(2)];
M2=[-D(2);P(1)*(P(1)-N(1))+P(2)*(P(2)-N(2))];
O=M1\M2;
O=O';
end
```

```
function r=mapsquare()
% Map function with squares only called mapsquare
taille=2; ecart=2;
axis([-2 16 -2 16]); axis square; hold on;
A1=[0 0];
r(1)=rectangle('Position',[A1(1) A1(2) taille taille],'FaceColor','k');
A2=A1;
for k=1:3
A2(2)=A2(2)+ecart+taille;
r(1+k)=rectangle('Position',[A2(1) A2(2) taille taille],'FaceColor','k');
end
for i=0:2
A1( 1)=A 1(1)+ecart+taille;
r(5+4*i)=rectangle('Position',[A1(1) A1(2) taille taille],'FaceColor','k');
A2=A1;
for k=1:3
A2(2)=A2(2)+taille+ecart;
r(5+4*i+k)=rectangle('Position',[A2(1) A2(2) taille taille],'FaceColor','k');
end
end
end
```

```
function d=dessiner_arc(M1,M2,O)
% This function draws the arc with centre O knowing the point where it
% begins and ends. It also returns the length of the arc.
R=distance(M1,O);
theta1=angle(M1,O,M2);
M3=[O(1)+4 O(2)];
theta2=angle(M3,O,M1);
k=0:0.1:1;
x=O( 1)+R*cos(theta2+k*theta1);
y=O(2)+R*sin(theta2+k*theta1);
plot(x,y);
d=R*abs(theta1);
end
```

```
function dist=distance(A,B)
% This function gives the distance between two points.
AB=B'-A';
dist=realsqrt(AB(1,:)^2+AB(2,:)^2);
end
```

```
function [d,M]=distdp(V,A,B)
% This fucntion calculate the distance between a line and a rectangle
% assuming that the line does not go through the rectangle. V is the vector
% that contains all the coordinates of the vertices.

% Calculation of the distance between the line and each vertices
d=abs(det( [B-A;V-A] ))/norm(B -A);

% Calculation of its orthogonal projection
D=eqdroite(A,B);
M1=[-D(1) 1;A(1)-B(1) A(2)-B(2)];
M2=[D(2);(A( 1)-B(1))*V (1)+(A(2)-B(2))*V (2)];
H=M1\M2;

% Calculation of the coordinates of the point M
HV=V-H';
HM=2/3*HV;
M=HM+H';
end
```

```
function D=eqdroite(A,B)
% Function that gives the coefficients a and b that satisfy the following
% equation of the line (AB) y=a*xb
a=(B(2)-A(2))/(B(1)-A(1));
b=A(2)-a*A(1);
D=[a b];
end
```

```
function [d,M]=distrd(rectangle,A,B)
```

```
% This fucntion calculate the distance between a line and a rectangle
% assuming that the line does not go through the rectangle.

%Calculation the coordinates of each vertice
Xr=rectangle.Position;
X1=[Xr(1) Xr(2)];
X2=[Xr(1)+Xr(3) Xr(2)];
X3=[Xr(1)+Xr(3) Xr(2)+Xr(4)];
X4=[Xr(1) Xr(2)+Xr(4)];
X=[X1;X2;X3;X4];

% Calculation of the distance between the line and each vertices
d1=abs(det( [B-A;X1 -A] ))/norm(B -A);
d2=abs(det([B-A;X2-A]))/norm(B-A);
d3=abs(det([B-A;X3-A]))/norm(B-A);
d4=abs(det([B-A;X4-A]))/norm(B-A);
[d,i]=min([d1 d2 d3 d4]); % distance and index of the nearest vertex

%Calcululation of its orthogonal projection
D=eqdroite(A,B);
C=X(i,:);
M1=[-D(1) 1;A(1)-B(1) A(2)-B(2)]; M2=[D(2);(A(1)-B(1))*C(1)+(A(2)-B(2))*C(2)];
H=M1\M2;

%Calculation of the coordinates of the point M
HC=C-H';
HM=2/3*HC;
M=HM+H';
end
```

```
function r=maprec1()
% Map function named maprec1
axis([-2 15 -1 14]); axis square; hold on;
r(1)=rectangle('Position',[0 1 2 1],'FaceColor','k');
r(2)=rectangle('Position',[3 1 4 1],'FaceColor','k');
r(3)=rectangle('Position',[9 2 4 2],'FaceColor','k');
r(4)=rectangle('Position',[2 3 1 1],'FaceColor','k');
r(5)=rectangle('Position',[4 4 1 2],'FaceColor','k');
r(6)=rectangle('Position',[7 3 1 1],'FaceColor','k');
r(7)=rectangle('Position',[0 6 3 2],'FaceColor','k');
r(8)=rectangle('Position',[6 6 2 2],'FaceColor','k');
r(9)=rectangle('Position',[9 5 3 2],'FaceColor','k');
r(10)=rectangle('Position',[0 10 2 2],'FaceColor','k');
r(11)=rectangle('Position',[3 10 2 3],'FaceColor','k');
r(12)=rectangle('Position',[6 9 2 1],'FaceColor','k');
r(13)=rectangle('Position',[9 9 1 3],'FaceColor','k');
r(14)=rectangle('Position',[11 9 1 3],'FaceColor','k');
r(15)=rectangle('Position',[12 9 1 2],'FaceColor','k');
end
```

```
function r=maprec2()
```

```
% Map function named maprec2
axis([-2 16 -2 15]); axis square; hold on;
r(1)=rectangle('Position',[0 0 5 2],'FaceColor','k','EdgeColor',[0.4 0.4 0.4]);
r(2)=rectangle('Position',[3 0 4 4],'FaceColor','k','EdgeColor',[0.4 0.4 0.4]);
r(3)=rectangle('Position',[9 3 2 2],'FaceColor','k','EdgeColor',[0.4 0.4 0.4]); r(4)=rectangle('Position',[10 2 3
2],'FaceColor','k','EdgeColor',[0.4 0.4 0.4]); r(5)=rectangle('Position',[10 4 4 3],'FaceColor','k','EdgeColor',[0.4 0.4 0.4]);
r(6)=rectangle('Position',[8 6 3 2],'FaceColor','k','EdgeColor',[0.4 0.4 0.4]);
r(7)=rectangle('Position',[5 6 2 1],'FaceColor','k','EdgeColor',[0.4 0.4 0.4]);
r(8)=rectangle('Position',[5 5 1 2],'FaceColor','k','EdgeColor',[0.4 0.4 0.4]);
r(9)=rectangle('Position',[0 6 3 2],'FaceColor','k','EdgeColor',[0.4 0.4 0.4]); r(10)=rectangle('Position',[2 7 2
3],'FaceColor','k','EdgeColor',[0.4 0.4 0.4]); r(11)=rectangle('Position',[2 9 4 1],'FaceColor','k','EdgeColor',[0.4 0.4 0.4]);
r(12)=rectangle('Position',[9 9 1 3],'FaceColor','k','EdgeColor',[0.4 0.4 0.4]); r(13)=rectangle('Position',[8 11 2
2],'FaceColor','k','EdgeColor',[0.4 0.4 0.4]); r(14)=rectangle('Position',[0 3 1 2],'FaceColor','k','EdgeColor',[0.4 0.4 0.4]);
r(15)=rectangle('Position',[1 4 1 1],'FaceColor','k','EdgeColor',[0.4 0.4 0.4]);
end
```

```
function rec=procherect(r,P,N,Q)
% First approach of the no-collision condition
n=length(r);
theta=angle(P,N,Q);
k=1;
anglerec=ones( 1,n);
indice=zeros(1,n);
for i=1:n
Q2=r(i).Position;
anglerec(i)=angle(P,N,[Q2(1)+Q2(3)/2 Q2(2)+Q2(4)/2]);
if (abs(anglerec(i))<abs(theta) && sign(anglerec(i))==sign(theta))
indice(k)=i; k=k+1;
end
end

indice(indice==0)=[];

I=length(indice);
d=zeros(1,I);

for p=1:I
A=r(indice(p)).Position;
d(p)=distance(N,[A( 1)+A(3)/2 A(2)+A(4)/2]);
end

[~,m]=min(d);
rec=r(indice(m));
end
```

```
function R1=rayon_pt(P,N,O1,M)
% This function calculate the radius of the circle tangent to PN that
% goes through M, knowing the centre of the circle tangent to PN in P
Da=eqdroite(N,M);
r=distance(O1,P);
```

```
[xout,yout]=linecirc(Da( 1),Da(2),O1(1),O1(2),r);
M1=[xout(1) yout(1)];
M2=[xout(2) yout(2)];
R(1)=distance(N,M) *r/distance(N,M1);
R(2)=distance(N,M) *r/distance(N,M2);
R1=max(R);
end
function y=velocity(x,w_L,V,r,t2,t3)
```

```
% This function create the variation of the angular speed of a wheel
% knowing the different values over time
y=V/r;
for k=1: length(w_L)
y=y+(w_L(k)-V/r)*(heaviside(x-t2(k+1))-heaviside(x-t3(k+1)));
end
end
```

```
function [dmax,V]=restriction(r,P,N,Q,O,M1,M2)
% The function returns the coordinates of the more distant vertex that
% belongs to T and its distance from O

%% Initialisation
n=length(r);
theta=angle(P,N,Q);
theta2=angle(M1,O,M2);
k=1;
anglerec=zeros( 1,n);
halfdiag=zeros( 1 ,n);
indice=zeros(1,n);

%% We erase all the rectangles whose centre is not in S
for i=1:n
Q2=r(i).Position;
anglerec(i)=angle(P,N,[Q2(1)+Q2(3)/2 Q2(2)+Q2(4)/2]);
halfdiag(i)=distance([Q2(1)+Q2(3)/2 Q2(2)+Q2(4)/2],[Q2(1) Q2(2)]);
if (abs(anglerec(i))<abs(theta) && sign(anglerec(i))==sign(theta))
indice(k)=i;
k=k+1;
end
end

indice(indice==0)=[];
% If S is empty then we stop then it is finished
if isempty(indice)==1
dmax=0;
V=0;
else

%% We erase all the rectangles whose centre is not in S'
indice2=zeros( 1,length(indice));
```

```
halfdiagmax=max(halfdiag);
drec=zeros(1,length(indice));
drecmax=halfdiagmax+distance(N,M1);
u=1;
for m=1:length(indice)
Q2=r(indice(m)).Position;
drec(m)=distance(N,[Q2(1)+Q2(3)/2 Q2(2)+Q2(4)/2]);
if drec(m)<drecmax
indice2(u)=indice(m);
u=u+1;
end
end
indice2(indice2==0)=[];

% If S' is empty then we can stop
if isempty(indice2)==1
dmax=0;
V=0;
else

% Else we have to find all the vertices that are in T
vertex=zeros(4,2);
t=1;
for u=1:length(indice2)
Q2=r(indice2(u)) .Position;
vertex(1,:)=[Q2(1) Q2(2)];
vertex(2,:)=[Q2(1)+Q2(3) Q2(2)];
vertex(3,:)=[Q2(1)+Q2(3) Q2(2)+Q2(4)];
vertex(4,:)=[Q2(1) Q2(2)+Q2(4)];
for v=1:4
anglevertex=angle(M1,O,vertex(v,:));
if (abs(anglevertex)<abs(theta2) && sign(anglevertex)==sign(theta2))
goodvertex(t,: )=vertex(v,:);
t=t+1;
end
end
end

% If T is empty then we can stop
if t==1
dmax=0;
V=0;
else

%% We have to find the more distant from O
d=zeros(1,t-1);
for w=1:t-1
d(w)=distance(O,[goodvertex(w,1) goodvertex(w,2)]);
end
[dmax,I] =max(d);
V=goodvertex(I,:);
end
end
```

```
end
end
```

```
function [M1,M2,O,w_L,w_R]=virage(P,N,Q,rec,V,r,L)

%% Initialisation of the variables
theta=angle(P,N,Q);
Rtheta=[cos(theta/2) -sin(theta/2) 0 ;sin(theta/2) cos(theta/2) 0; 0 0 1];
Rtheta2=[cos(theta) -sin(theta) 0 ;sin(theta) cos(theta) 0; 0 0 1];
T1=[1 0 N(1);0 1 N(2); 0 0 1];
T2=[1 0 -N(1);0 1 -N(2); 0 0 1];
T=T1*Rtheta*T2;
T3=T1*Rtheta2*T2;

%% Preliminary study % Creation of the bisector
B=T*[P(1) P(2) 1]';
B=[B(1) B(2)];
D=eqdroite(N,B);

% Intersection point of the perpendicular in P and the bisector
O1=centre(P,N,D);

%% Finding the largest arc possible
% Calculate dmin the minimum between half of the distance PN and NQ
d1=distance(N,P);
d2=distance(N,Q);
d=[d1 d2];
dmin=1/2*min(d);

% M is the point on PN that satisfies NM=dmin
NP=P-N;
NM=dmin*NP/d1;
M=NM+N;

%% Find the radius and the centre of the arc that is tangent NP and NQ and goes through M
R=(dmin*distance(P,O1))/(distance(N,P));

%thanks to the theorem of Thales
O=centre(M,N,D);


%% Deduce the point M1 and M2
M1=M;
M2=T3*[M(1) M(2) 1]';
M2=[M2(1) M2(2)];

%% Find the coordinates of the more distant vertex that belongs to T and its distance from O
[dmax,V1]=restriction(rec,P,N,Q,O,R,M1,M2);

%% Test the value and do accordingly
% 1st Case : dmax<R then the best arc is the largest arc possible
% 2nd case : we have to find an arc that do not collide
```

```
if dmax>R
[d3,M3]=distpd(V1,P,N);
[d4,M4]=distpd(V1,P,Q);
distmin=min(d3,d4);
if distmin==d3
M=M3;
else
M=M4;
end

%% Find the radius of the arc, the point M1, M2 and O
R=rayon_pt(P,N,O1,M);

%Solution using the theorem of Thales and the fact that NM and NP are
%collinear and in the same sens
nPO1=distance(P,O1);
NP=P-N;
lambda=R/nPO 1;

%we also have lambda=nNM/nNP
NM1=lambda*NP;
M1=NM1+N;
M2=T3*[M1(1)M1(2) 1]';

% M2 is the image of M1 with a rotation with angle theta
M2=[M2(1) M2(2)];
O=centre(M1,N,D);
end

%% Calculation of the angular speeds needed to follow such arc
if theta<0
w_L=V/r*(1-L/(2*R));
w_R=V/r*(1+L/(2*R));
else
w_L=V/r*(1+L/(2*R));
w_R=V/r*( 1 -L/(2*R));
end
end
```

Appendix B. Main Script

```
%% Clearance
clear all
close all

%% Parameters
angled=30; %tolerance angle in degree
V=0.25; %velocity of the robot
r=0.05; %radius of the wheel
L=0.15; %length between the two wheel

%% Set of points studied
%% Main

map=@maprec 1; % map choice

figure(1)
subplot(1,2,1); hold on;
rec=map();

[u,v]=ginput; %function that allows the user to draw the points on the map
u=u'; v=v';
u1=u;v1=v;
plot(u,v,'r+'); %the first set of points is drawn in red
N=length(u);

%Create a new set of points from the latest using the tolerance condition
for i=1:N-2
theta=angle([u(i) v(i)],[u(i+1) v(i+1)],[u(i+2) v(i+2)]);
theta=abs(theta)*180/pi;
if (180-angled<theta && theta<180+angled)
u(i+1)=u(i);
v(i+1)=v(i);
end
end

%We erase from that new set the duplications
P=[u' v'];
P=union(P,P,'rows','stable');
P=P';
u=P(1,:);
v=P(2,:);

plot(u,v,'g+') %the set obtained is drawn in green

%Initialisation of the different variable
N=length(u);
M1=[u(1) v(1)];
d1=0;d2=0;
t1=zeros( 1,N); t2=zeros( 1,N) ;t3=zeros( 1,N-1);
w_L=zeros( 1 ,N-2) ;w_R=zeros( 1,N -2);
```

```
%Drawing of the best arcs possible for every two consecutive segments
for k=1:N-2
[M2,M3,O,w_L(k),w_R(k)]=virage([u(k) v(k)],[u(k+1) v(k+1)],[u(k+2) v(k+2)],rec,V,r,L);
plot([M1(1) M2(1)],[M1(2) M2(2)]);
darc=dessiner_arc(M2,M3,O);
d1=d1 +distance (M1,M2)+darc;
t1(k+1)=t3(k);
t2(k+1)=distance(M 1,M2)/V+t1(k+1);
t3(k+1)=darc/V+t2(k+1); d
2=d2+distance([u(k) v(k)],[u(k+1) v(k+1)]);
M1=M3;
end
plot([M3(1) u(N)],[M3(2) v(N)]);
d1=d1+distance(M3,[u(N) v(N)]);
d2=d2+distance([u(N-1) v(N-1)],[u(N) v(N)]);
t1(N)=t3(N-1);
t2(N)=distance(M1,M2)/V+t 1(N);

str = sprintf('Path with curve optimization (longueur=%f)',d1);
title(str);

%Drawing of the second graph with the path unoptimized
subplot(1,2,2);
str2 = sprintf('Path without curve optimization (longueur=%f)',d2);
title(str2);hold on;
map();
plot(u1,v1,'r+')
plot(u,v,'g+');
for k=1:N-1
plot([u(k) u(k+1)],[v(k) v(k+1)],'k');
end

%Drawing of the variation of the angular velocities
figure(2); hold on;
fplot(velocity(x,w_L,V,r,t2,t3),[0,t2(N)] ,'b');
fplot(velocity(x,w_R,V,r,t2,t3),[0,t2(N)],'r','LineStyle','--');
```