

# Design of a GUI:

Definition of an arbitrary path for a cobot  
and  
corresponding collision detection

Alexander Klein

Advisor: Edward Colgate

Northwestern University

## Table of contents

1. INTRODUCTION .....	1
2. GRAPHICS.....	3
2.1. DISPLAY .....	3
2.2. POINTS AND LINES .....	3
2.3. CURVES.....	4
3. COMPUTATION .....	6
3.1. CLOTHOIDS .....	6
3.1.1 <i>General form and approximation</i> .....	6
3.1.2 <i>Implementation</i> .....	9
3.2. COLLISION DETECTION .....	12
3.2.1 <i>Distance to the corner points</i> .....	13
3.2.2 <i>Separation of the plane</i> .....	13
3.2.3 <i>Distance to the clothoid</i> .....	15
3.2.4 <i>Distance to a line</i> .....	18
3.3. COMPUTATION TIME.....	19
4. PROGRAM.....	21
5. PROBLEMS TO BE SOLVED AND FUTURE WORK.....	23
5.1. POLYGON WITH INNER ANGLES GREATER THAN $180^\circ$ .....	23
5.2. A PATH WITH INTERSECTING ELEMENTS .....	24
6. CONCLUSION.....	26
REFERENCE .....	27

## Table of figures

<b>FIGURE 1: SNAPSHOT OF SCREEN AFTER THE USER HAS DEFINED THE PATH POINTS</b>	4
<b>FIGURE 2: FINAL SHAPE OF PATH</b> .....	5
<b>FIGURE 3: CLOTHOID</b> .....	8
<b>FIGURE 4: CLOTHOID IMPLEMENTATION</b> .....	9
<b>FIGURE 5: INTERSECTION OF TWO ADJACENT CLOTHOIDS</b> .....	12
<b>FIGURE 6: SEPARATION OF THE PLANE</b> .....	14
<b>FIGURE 7: DISTANCE TO A LINE</b> .....	18
<b>FIGURE 8</b> .....	23
<b>FIGURE 9</b> .....	24

# 1. Introduction

The task of a cobot - “collaborative robot” [4] - is to support a human operator/worker. It was developed since usual robots are active elements and can work independently next to workers. In contrast, a cobot is a passive device. It needs a human operator to provide motive forces. The user defines the moving direction. Nonholonomic joints such as wheels, instead of servos, are steered in order to allow this movement with the least effort for the operator [5]. Constraints are software defined and lead to two different modes in which the cobot can move: The virtual caster and the virtual wall mode. In the former one, the wheel moves like a caster and its motion is unconstrained. Forces perpendicular to the actual moving direction of the cobot are minimized by steering the wheel in the new direction. Thus, the human collaborator can move the cobot in any direction wanted.

In the latter, the wheel is no longer steering to minimize perpendicular forces. The cobot is now in virtual wall mode. The wheel is steered tangent to the virtual wall and forces that tend to penetrate the constraint are ignored. Thus, the user has two possibilities: Moving the cobot along the wall or pushing it back to the unconstrained area.

Further there could be a path defined on which the cobot is allowed to move. Thus, the cobot is in virtual wall mode all the time and all forces perpendicular to the path are ignored. A detailed analysis of the two modes and their application has already been reported [6].

The aim of this project is to build a graphical user interface with which an arbitrary path in the xy-plane can be defined. Since a path is a constraint for the cobot’s movement, we further need to detect collision with the path.

Supported by the mouse the user defines points on the screen. These points are considered to be the corner points of the path and are connected by straight lines. Two adjacent lines are blended together with a curve in order to avoid the tangent and curvature discontinuities at the corner point joining them. To be able to choose an adequate smoothness for each blend, the user can also define its minimum curvature. To insure that the path is closed (polygon), the first point needs to be defined as the last one.

The second part of the project covers the collision detection. This is needed for the control system that is applied to keep the cobot on the defined path. The controller requires the following inputs:

- 1) the actual cobot position,
- 2) derived from 1) the minimum distance to the path, and
- 3) the path curvature and tangent at the point of minimum distance

With this, the controller can compare the actual position and moving direction to the desired values, that the cobot should have, and react accordingly.

## 2. Graphics

In this chapter, it is described how the user will define arbitrary points in the xy-plane. Straight lines will connect the points and the corners are substituted by curves. This already involves parts of the computation that is covered below.

### 2.1. Display

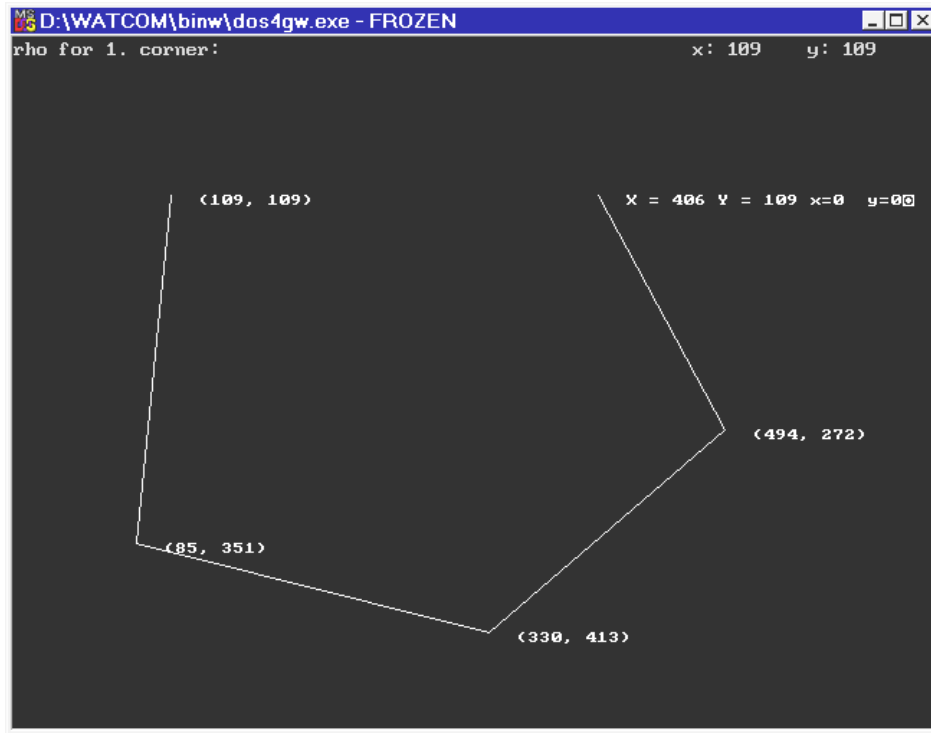
The monitor shows the mouse pointer and its actual global position in pixels in the upper right corner of the screen. Additionally, after the input of the first point the distance between the last defined point and the actual position of the mouse pointer is displayed. This setup of the screen facilitates the orientation for the user and helps to define lines with a desired length. For the chosen *videomode* (`_VRES16COLOR`) the size of the screen is 640x480 pixels and its coordinate center is located at the upper left corner.

### 2.2 Points and lines

Points are defined by moving the mouse and clicking on the left mouse button. At every chosen point the global position in pixels is displayed on the screen, providing the user with better sense of orientation within the screen. Further, after defining each point, a straight line is generated to connect it to the previous defined point. A typical picture obtained by this procedure is shown in Figure 1.

Since the objective is to get a closed polygon as the path, the program automatically defines the first two points as the last ones. Both of them are needed in order to get a corner at point 1. Thus, if  $n$  points were defined, we get  $n$  corners, where the first corner corresponds to point 2 and the last one to point 1.

Clicking on the right mouse button terminates the input of points. Next the user is offered the possibility to define the minimum C-space radius of the curvature  $\rho$  for the corners. These concepts are addressed in greater detail in the next section.



**Figure 1: Snapshot of screen after the user has defined the path points**

## 2.3 Curves

Since the corner of two lines is not a smooth transition, we have to find a way to avoid this. Thus, we define a curve that blends the two lines. One constraint for this curve is necessary in order to obtain a smooth transition: Both the curve and the line have to have the same curvature at the transition point. Since we blend two straight lines, this curvature has to be zero. As we will see this is accomplished by a clothoid. The curvature of a clothoid is a linear function of its own arc length  $s$

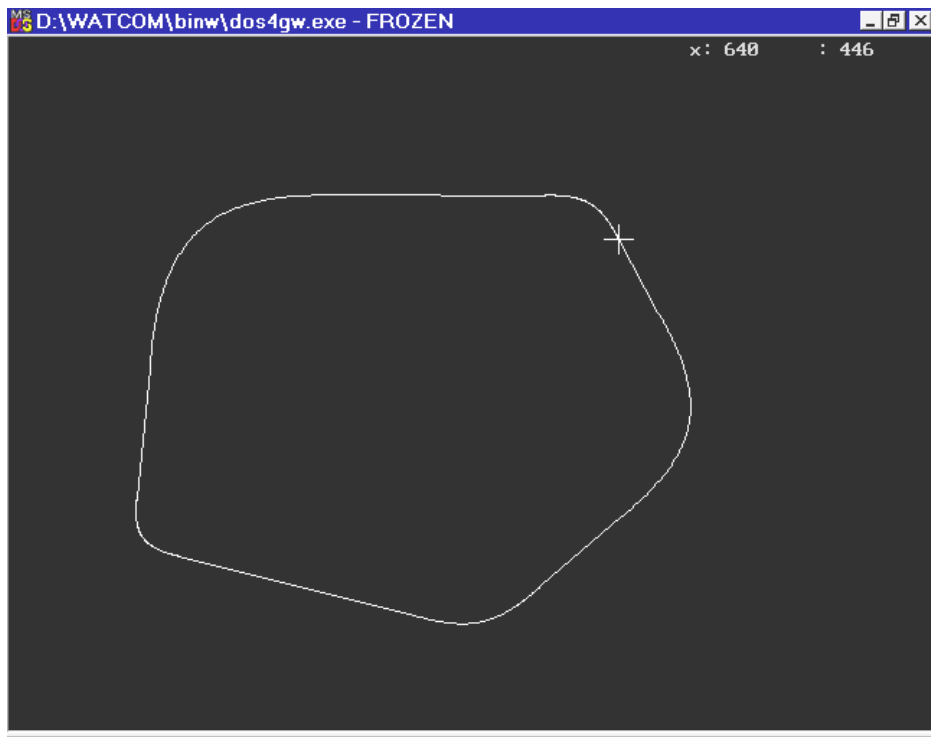
$$\kappa = k_c s + \kappa_0 \quad (2.3. 1)$$

If we choose  $\kappa_0 = 0$ , it can be seen that the curvature is zero at the beginning of the clothoid ( $s=0$ ).

The disadvantage of the usage of the clothoid is the relatively complicated mathematical description. As a result, computations are tedious whenever the clothoid is involved.

From equation (2.3. 1) can be seen that changing the coefficient  $k_c$  can vary the shape/smoothness of the blend. Thus, after defining the input, the user can vary the shape by choosing  $\rho$  (minimum C-space radius of curvature) for each corner. By increasing  $\rho$ , the blend becomes smoother (see chapter 3.1.2 Implementation).

Eventually, the polygon with all clothoids is plotted on the screen in its final shape (Figure 2). The cross marks the starting position of the cobot.



**Figure 2: Final shape of path**

### 3. Computation

As the computational procedure is the most significant part of the program, it is beneficial that we here outline its fundamental steps. The computation contains the simple procedures involving points and lines (e.g. direction of a line). In addition, it includes complex algorithms such as the Newton-Raphson iteration, needed to determine the closest point on a clothoid. Further, we have to find a coordinate frame for each corner's proximal and distal element of the clothoid.

The lines, points and clothoids will be referred to as *elements* in the subsequent paragraphs.

Most of the computation is necessary for the collision detection, i.e. to determine the shortest distance from a point in the xy-plane to the defined path. We need to divide the whole plane in two half-planes in order to be able to determine whether the cobot is closest to a clothoid.

The speed of the computation is a concern. The sequence in which we determine the closest elements is crucial for the program efficiency. If the sequence is properly chosen, we reduce the calculations to those that uniquely identify the closest element. Chapter 3.2. follows the sequence used in the program.

#### 3.1. Clothoids

This part includes the computation of the straight lines, the general form of a clothoid, its approximation and implementation for our purposes. The latter refers to the set up of the coordinate frames for each corner of the defined polygon.

##### 3.1.1 General form and approximation

A clothoid is a curve whose curvature is a linear function of its own arc length  $s$

$$\kappa = k_c s + \kappa_0 \tag{2.3. 1}$$

Since we blend together two straight lines, which have zero curvature, we choose  $\kappa_0=0$ .

A parametric description can be found as follows

$$\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} aC(p) \\ aS(p) \end{pmatrix} = \begin{pmatrix} a \int_0^p \cos\left(\frac{\pi u^2}{2}\right) du \\ a \int_0^p \sin\left(\frac{\pi u^2}{2}\right) du \end{pmatrix}, \quad a > 0 \quad (3.1.1)$$

where  $C(p)$  and  $S(p)$  are known as Fresnel integrals [1].

Since there exists no rational form of these integrals we have to find a proper approximation for the computation of the position-vector  $\mathbf{x}$ .

Heald [3] provides a number of approximations that include only rational and trigonometric functions. His lowest order approximation is the following

$$C(p) = \frac{1}{2} - R_{12}(p) \sin\left(\frac{1}{2} \pi (A_{03}(p) - p^2)\right) \quad (3.1.2)$$

$$S(p) = \frac{1}{2} - R_{12}(p) \cos\left(\frac{1}{2} \pi (A_{03}(p) - p^2)\right) \quad (3.1.3)$$

with

$$R_{12}(p) = \frac{0.56p + 1}{1.79p^2 + 2.054p + \sqrt{2}} \quad (3.1.4)$$

$$A_{03}(p) = \frac{1}{0.803p^3 + 1.886p^2 + 2.524p + 2} \quad (3.1.5)$$

These functions can be computed rather easily and are for our application sufficiently

$$\mathbf{T} = \frac{d\mathbf{x}}{dp} = \begin{pmatrix} \cos\left(\frac{\pi p^2}{2}\right) \\ \sin\left(\frac{\pi p^2}{2}\right) \end{pmatrix} \quad (3.1.6)$$

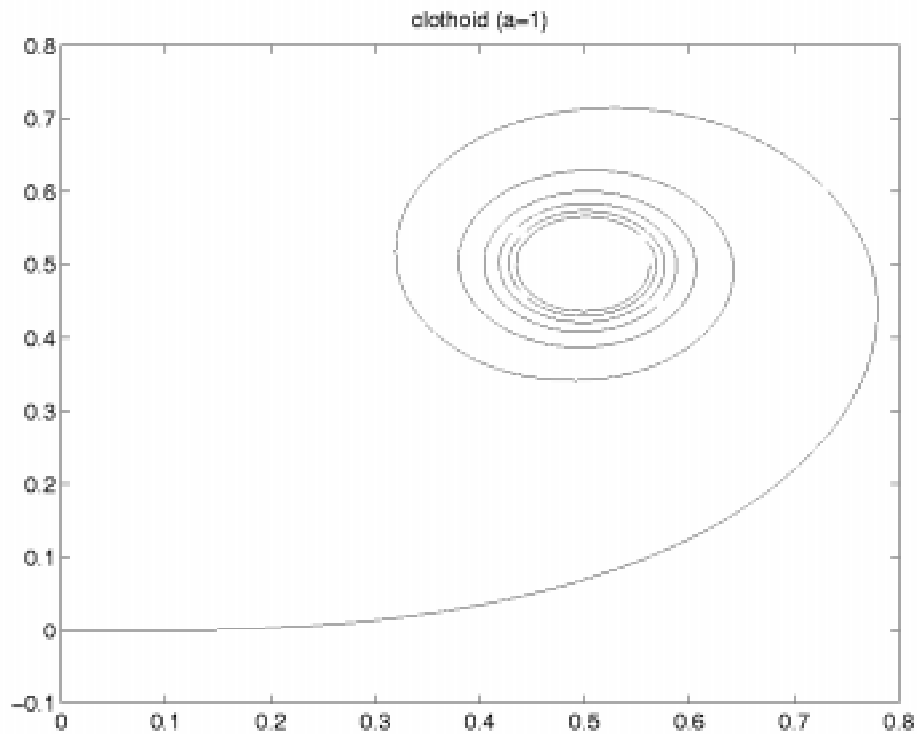
Using the definition of the curvature for a function in parametric form

$$\kappa = \frac{\dot{x}\ddot{y} - \ddot{x}y}{(\dot{x}^2 + \dot{y}^2)^{\frac{3}{2}}} \quad (3.1.7)$$

yield

$$\kappa = \pi p = \frac{\pi}{a} s \quad (3.1.8)$$

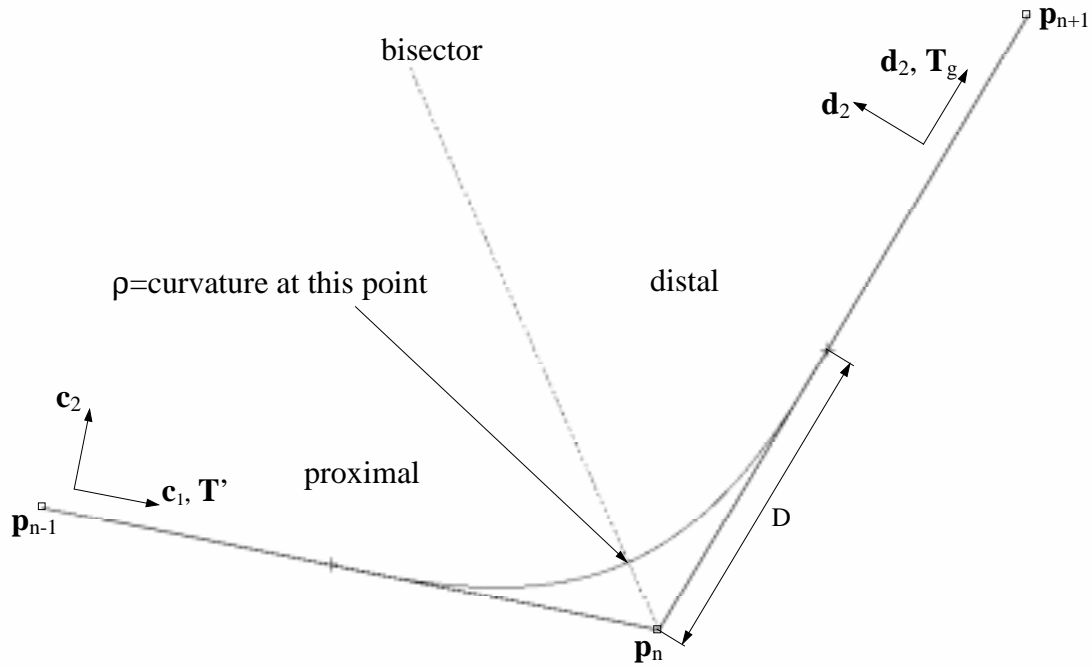
From (3.1.8) can be seen very easily, that the curvature is indeed a linear function of the arc length  $s$  since  $a$  and  $\pi$  are constants.



**Figure 3: Clothoid**

### 3.1.2 Implementation

We need to compute a clothoid that blends together two given lines with a user defined minimum C-space radius of curvature ( $\rho$ ). The clothoid is separated in two parts that are symmetric to the bisector: one is the proximal element, the other is named distal. In our case, where  $\mathbf{p}_n$  is the considered corner, the element before  $\mathbf{p}_n$  is the proximal element and the one after  $\mathbf{p}_n$  is the distal element (as shown in Figure 4):



**Figure 4: Clothoid implementation**

In order to set up the clothoid, we have to define separate coordinate frames for its two parts.

$\mathbf{T}'$  and  $\mathbf{T}_g$  represent the direction of the lines and thus define  $\mathbf{c}_1$  and  $\mathbf{d}_1$ , respectively.

$$\mathbf{c}_1 = \mathbf{T}' = \frac{\mathbf{p}_n - \mathbf{p}_{n-1}}{|\mathbf{p}_n - \mathbf{p}_{n-1}|} \quad (3.1.9)$$

$$\mathbf{d}_1 = \mathbf{T}_g = \frac{\mathbf{p}_{n+1} - \mathbf{p}_n}{|\mathbf{p}_{n+1} - \mathbf{p}_n|} \quad (3.1.10)$$

For  $\mathbf{c}_2$  and  $\mathbf{d}_2$  we have to make sure that they have the right orientation (as shown in Figure 4). This is guaranteed by the following computation:

$$\mathbf{c}_2 = \frac{(\mathbf{I} - \mathbf{T}'\mathbf{T}'^T)(\mathbf{T}'_g - \mathbf{T}')}{\|(\mathbf{I} - \mathbf{T}'\mathbf{T}'^T)(\mathbf{T}'_g - \mathbf{T}')\|} \quad (3.1. 11)$$

$$\mathbf{d}_2 = \frac{(\mathbf{I} - \mathbf{T}'_g\mathbf{T}'_g^T)(\mathbf{T}'_g - \mathbf{T}')}{\|(\mathbf{I} - \mathbf{T}'_g\mathbf{T}'_g^T)(\mathbf{T}'_g - \mathbf{T}')\|} \quad (3.1. 12)$$

where  $\mathbf{I}$  is the 2x2 identity matrix.

Further we need the parameters that describe the clothoid:

$$\begin{aligned} a &= \pi\rho p_0 \\ D &= a \left( C(p_0) + S(p_0) \cot\left(\frac{\alpha}{2}\right) \right) \\ p_0 &= \sqrt{1 - \frac{\alpha}{\pi}} \end{aligned} \quad (3.1. 13)$$

$p_0$  is half the length of the entire clothoid (the length of each element).  $D$  is the distance of the starting/ending point of the clothoid to the corner  $\mathbf{p}_n$ .  $a$  defines the smoothness of the blend.

Now, both clothoid elements can be computed

$$\Sigma_{prox}(s) = \mathbf{p}_n + (aC(p) - D)\mathbf{c}_1 + aS(p)\mathbf{c}_2 \quad , p \leq p_0 \quad (3.1. 14)$$

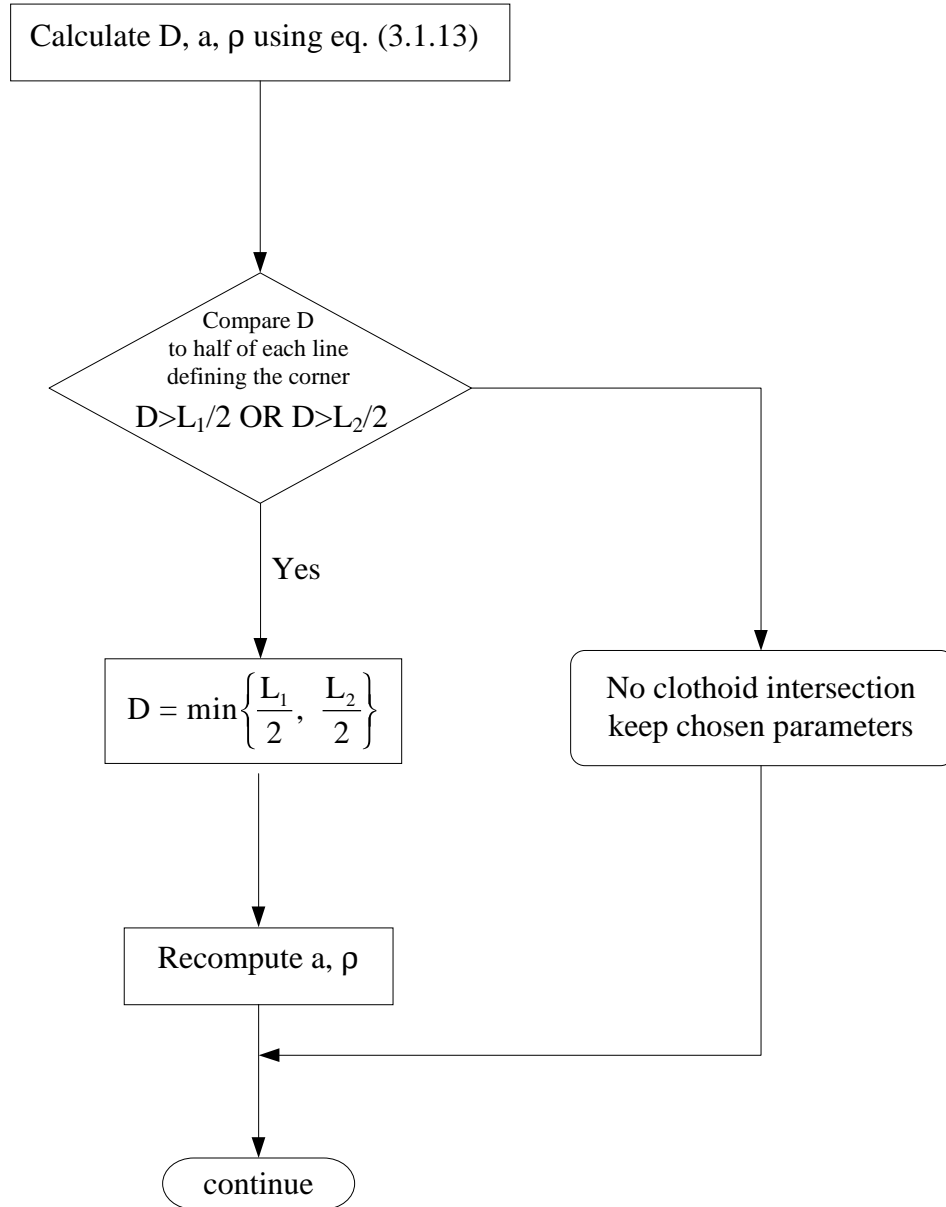
$$\Sigma_{dist}(s) = \mathbf{p}_n + (D - aC(2p_0 - p))\mathbf{d}_1 + aS(2p_0 - p)\mathbf{d}_2 \quad , p_0 < p \leq 2p_0 \quad (3.1. 15)$$

Choosing  $p$  between 0 and  $2p_0$  we can compute every point on the clothoid.

From (3.1.13) can be seen that  $a$  is proportional to  $\rho$ . Increasing the value of  $\rho$  will increase the minimum radius of curvature and thus, the blend becomes smoother.

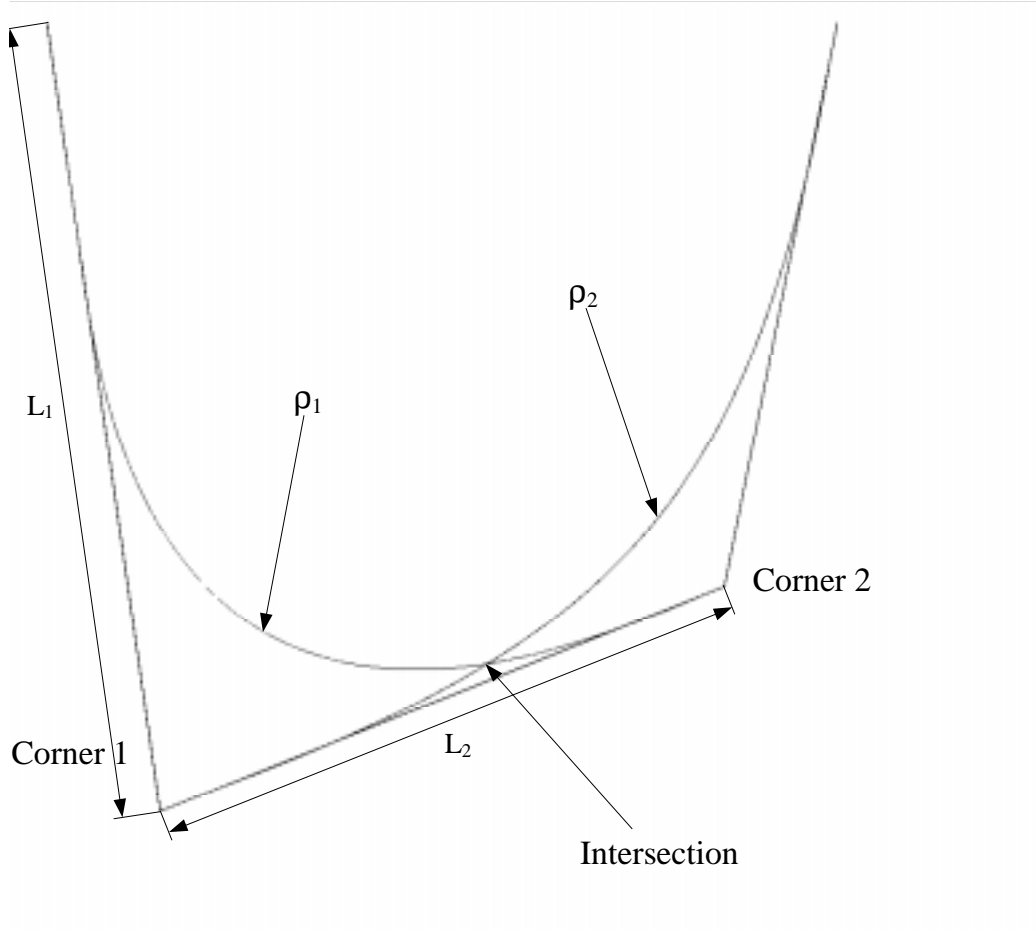
Each corner has its own turn angle  $\alpha$  and minimum C-space radius of curvature  $\rho$  and also the directions of the lines differ. Thus, the proximal and the distal coordinate frame have to be computed for each corner.

A potential problem was that two clothoids could intersect due to the choice of  $\rho$  of two adjacent corners. To avoid this problem, we follow the logic shown by the following flowchart supported by Figure 5:



First the initial parameters of the clothoid are computed. To avoid a potential intersection we decide to limit  $D$  to the minimum of  $L_1/2$  and  $L_2/2$ . Is  $D$  larger than  $\min\left\{\frac{L_1}{2}, \frac{L_2}{2}\right\}$ ,  $D$  is set to be exactly this minimum. In this case we have to recompute the corresponding

parameters and  $\rho$ . Accordingly,  $\rho$  is limited by a threshold beyond which the blend does not become any smoother.



**Figure 5: Intersection of two adjacent clothoids**

### 3.2. Collision detection

Collision detection requires the determination of the closest element and on this element the closest point, the element's curvature and tangent. The latter three values are needed as input for the controller. This chapter follows the "algorithm" shown in Appendix A.

### 3.2.1 Distance to the corner points

The computation of the distance from the actual position to a corner point is rather easy but it can be considered the key for the decision making whether the cobot is close to a corner – to a clothoid – or to a straight line. Once we know the corner in which the cobot actually is, the subsequent computation can be started.

The distance to a point is the length of the vector from the actual position to the point:

$$\text{distance\_to\_point} = |\mathbf{p}_n - \mathbf{position}| \quad (3.2. 1)$$

Applying this for all n corners the closest one can be determined. This reduces the number of potential collision elements to two lines and one clothoid, including its two parts.

### 3.2.2 Separation of the plane

After the closest point is determined, the plane is separated in two half planes. This is necessary for the decision making whether the position is so close to the corner that it is closest to the according clothoid or it is still “far” away, and thus closest to a straight line. The plane is separated in an area  $A_1$ , including the clothoid, and the area  $A_2$  (Figure 6). Is the cobot within the area  $A_1$  it might be closest to the clothoid, otherwise it will collide with a straight wall.

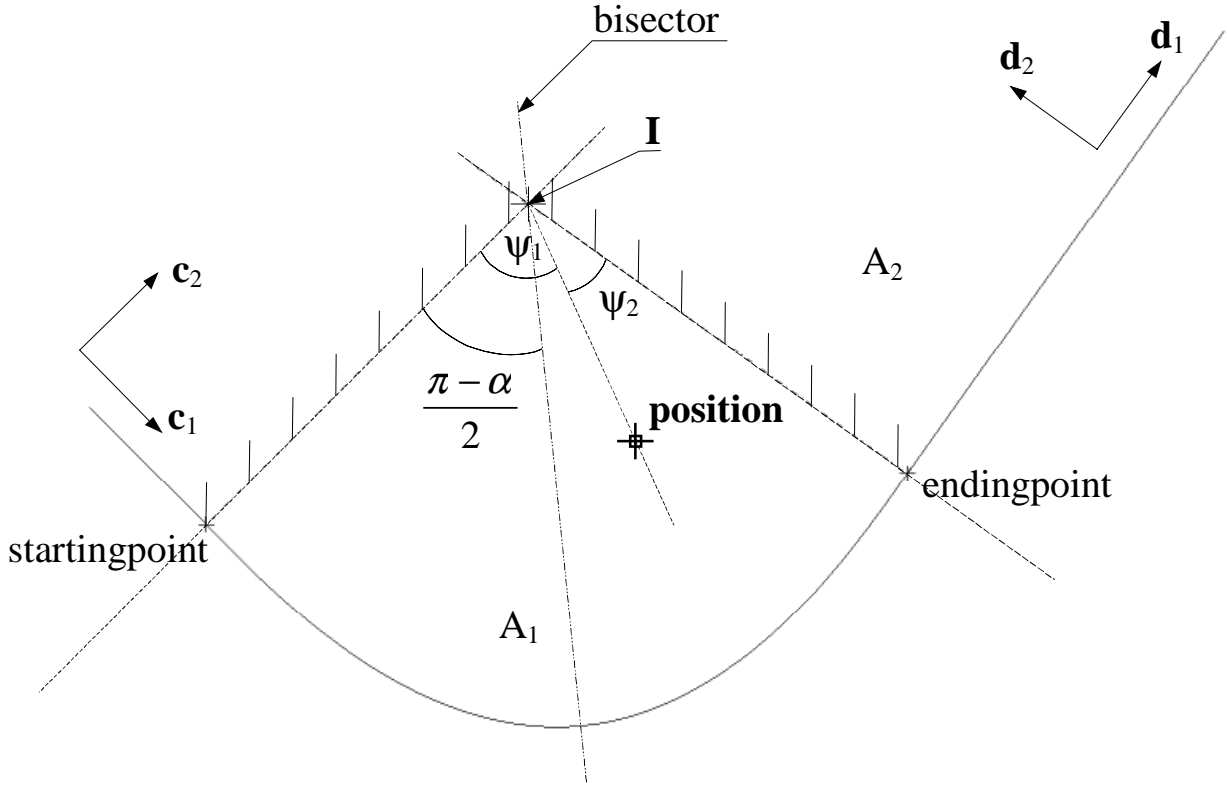
To show how the actual decision is made we assume that the cobot is at a position in  $A_1$  (closest to a clothoid). First we have to determine point **I** which is the point of intersection of the two perpendicular lines through the clothoid’s starting and ending point. To do this we write the equations for the two lines in parametric description

$$\mathbf{x}_1 = \mathbf{p}_1 + r\mathbf{c}_2 = \begin{pmatrix} p_{1x} \\ p_{1y} \end{pmatrix} + r \begin{pmatrix} c_{2x} \\ c_{2y} \end{pmatrix} \quad (3.2. 2)$$

$$\mathbf{x}_2 = \mathbf{p}_2 + t\mathbf{d}_2 = \begin{pmatrix} p_{2x} \\ p_{2y} \end{pmatrix} + s \begin{pmatrix} d_{2x} \\ d_{2y} \end{pmatrix} \quad (3.2. 3)$$

with

$$\begin{aligned} \mathbf{p}_1 &= \mathbf{p}_n - D\mathbf{c}_1 \\ \mathbf{p}_2 &= \mathbf{p}_n + D\mathbf{d}_1 \end{aligned} \quad (3.2.4)$$



**Figure 6: Separation of the plane**

Knowing that  $\mathbf{I}$  has to be on the bisector we set  $r=t$  and get the coordinates of  $\mathbf{I}$ :

$$\mathbf{I} = \begin{pmatrix} s_x \\ s_y \end{pmatrix} = \begin{pmatrix} p_{1x} \\ p_{1y} \end{pmatrix} + \frac{p_{2x} - p_{1x}}{c_{2x} - d_{2x}} \begin{pmatrix} c_{2x} \\ c_{2y} \end{pmatrix} \quad (3.2.5)$$

From this we can derive the two angles that are necessary to determine non-ambiguously that the cobot is in  $A_1$ .

Generally the angle between two vectors  $\mathbf{a}$  and  $\mathbf{b}$  is given by the scalar product

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos \alpha \quad (\text{when } \alpha \text{ is the angle between these two vectors})$$

Thus, we get the following expressions for  $\psi_1$ , and  $\psi_2$ :

$$\cos(\psi_1) = \frac{(-\mathbf{c}_2) \cdot (\mathbf{position} - \mathbf{s})}{|\mathbf{position} - \mathbf{s}|}, \text{ since } |c_2|=1 \quad (3.2. 6)$$

$$\cos(\psi_2) = \frac{(-\mathbf{d}_2) \cdot (\mathbf{position} - \mathbf{s})}{|\mathbf{position} - \mathbf{s}|}, \text{ since } |d_2|=1 \quad (3.2. 7)$$

The conditions for the position to be inside  $A_1$  and closest to the proximal element of the clothoid are

$$\begin{aligned} 0 \leq \psi_1 \leq \frac{\pi - \alpha}{2} \text{ AND} \\ \frac{\pi - \alpha}{2} < \psi_2 < \pi - \alpha \end{aligned} \quad (3.2. 8)$$

To check whether the distal element is closest,  $\psi_1$  and  $\psi_2$  have to be exchanged. We need both angles in order to define that the cobot is in  $A_1$ , because the scalar product does not determine the direction of the angle. If either of these conditions does not hold, the cobot is in area  $A_2$  and thus closest to a straight line. After this final decision making, the program computes the distance either to the according clothoid element and/or to the straight lines. This is shown in the following paragraphs.

Here, we have chosen to use the Newton-Raphson iteration. This method converges quadratically and is thus the most efficient for our purposes. However, the following concern needs to be addressed: the initial point of the iteration has to be chosen properly, so that the iteration determines only the desired null (within the considered interval). This is achieved by taking the clothoid point that lies on the bisector as starting point. This middle point of the clothoid is a proper choice since we know that the null has to be between the starting and the ending point of the clothoid.

Since the clothoid consists of two parts (proximal and distal) and each of them has its own mathematical description, we find two different functions<sup>1</sup>. The parameter  $p$  distinguishes them: The proximal element refers to  $p \leq p_0$  and the distal one refers to  $p_0 < p \leq 2p_0$ . Additionally, the indices *prox* and *dist* are used.

The vector from the actual position (**pos**) to the clothoid, using the equations (3.1.14) and (3.1.15), can be described as follows

$$\mathbf{x}_{prox} = \mathbf{pos} - \left( \mathbf{p}_n + (aC(p) - D)\mathbf{c}_1 + aS(p)\mathbf{c}_2 \right) \quad , s \leq p_0 \quad (3.2. 10)$$

$$\mathbf{x}_{dist} = \mathbf{pos} - \left( \mathbf{p}_n + (D - aC(2p_0 - p))\mathbf{d}_1 + aS(2p_0 - p)\mathbf{d}_2 \right) \quad , p_0 < s \leq 2p_0 \quad (3.2. 11)$$

With the tangent  $\mathbf{T}$  of the clothoid

$$\mathbf{T}_\Sigma(s) = \begin{cases} \cos\left(\frac{\pi p^2}{2}\right)\mathbf{c}_1 + \sin\left(\frac{\pi p^2}{2}\right)\mathbf{c}_2 & 0 \leq p < p_0 \\ \cos\left(\frac{\pi(2p_0 - p)^2}{2}\right)\mathbf{d}_1 - \sin\left(\frac{\pi(2p_0 - p)^2}{2}\right)\mathbf{d}_2 & p_0 \leq p < 2p_0 \end{cases} \quad (3.2. 12)$$

we get the scalar product of equation (3.2.10) as a function of the parameter  $p$

$$f = \begin{cases} \left[ \mathbf{pos} - \left( \mathbf{p}_n + (aC(p) - D)\mathbf{c}_1 + aS(p)\mathbf{c}_2 \right) - \mathbf{q} \right] \left[ \mathbf{T}_{\Sigma_{prox}} \right] & , p \leq p_0 \\ \left[ \mathbf{pos} - \left( \mathbf{p}_n + (D - aC(2p_0 - p))\mathbf{d}_1 + aS(2p_0 - p)\mathbf{d}_2 \right) \right] \left[ \mathbf{T}_{\Sigma_{dist}} \right] & , p_0 < s \leq 2p_0 \end{cases} \quad (3.2. 13)$$

---

<sup>1</sup> To avoid confusion: The actual program has to compute only one of the distances, because by separating the plane we already made a decision whether the position is closest to the proximal or the distal element.

For this function we want to determine the null. The resulting  $p$  will give us the point on the clothoid that has minimal distance to the position. To be able to apply Newton-Raphson we need the first derivative of  $f$  with respect to  $p$

$$\begin{aligned} \mathbf{f}'_{\text{prox}} = & a + \pi p (\mathbf{p}_n - \mathbf{q}) \cdot \left( \cos\left(\frac{\pi p^2}{2}\right) \mathbf{c}_2 - \sin\left(\frac{\pi p^2}{2}\right) \mathbf{c}_1 \right) \\ & - \pi p (aC - D) \sin\left(\frac{\pi p^2}{2}\right) + \pi p aS \cos\left(\frac{\pi p^2}{2}\right) \end{aligned} \quad (3.2. 14)$$

$$\begin{aligned} \mathbf{f}'_{\text{dist}} = & a + \pi(2p_0 - p) (\mathbf{p}_g - \mathbf{q}) \cdot \left( \sin\left(\frac{\pi(2p_0 - p)^2}{2}\right) \mathbf{d}_1 + \cos\left(\frac{\pi(2p_0 - p)^2}{2}\right) \mathbf{d}_2 \right) \\ & + \pi(2p_0 - p) \left( (D - aC) \sin\left(\frac{\pi(2p_0 - p)^2}{2}\right) + aS \cos\left(\frac{\pi(2p_0 - p)^2}{2}\right) \right) \end{aligned} \quad (3.2. 15)$$

where  $f'$  means  $df/dp$

With these equations we find the Newton-Raphson formula for this problem

$$\mathbf{p}_{n+1} = \mathbf{p}_n - \frac{f(\mathbf{p})}{f'(\mathbf{p})} \quad (3.2. 16)$$

As mentioned above this iteration is started with  $p_0$  (point of the clothoid on the bisector of the corner). The search is terminated when the change of the “corrector term”  $f/f'$  is lower than  $10^{-3}$ . This guarantees sufficient accuracy.

Now we have determined  $p$  for that the distance of the cobot to the clothoid is minimal. Using this  $p$ , the point on the clothoid can be computed with equation (3.1.14) or (3.1.15). For the curvature we need the following equation

$$\kappa_{\Sigma}(p) = \begin{cases} \frac{p}{\rho p_0} & 0 \leq p < p_0 \\ \frac{2p_0 - p}{\rho p_0} & p_0 \leq p < 2p_0 \end{cases} \quad (3.2. 17)$$

The tangent can be determined with equation (3.2.12).

### 3.2.4 Distance to a line

For the computation of the distance to the lines we need to describe two lines in parametric form, as shown in the subsequent sketch, and determine their point of intersection  $\mathbf{q}$ .

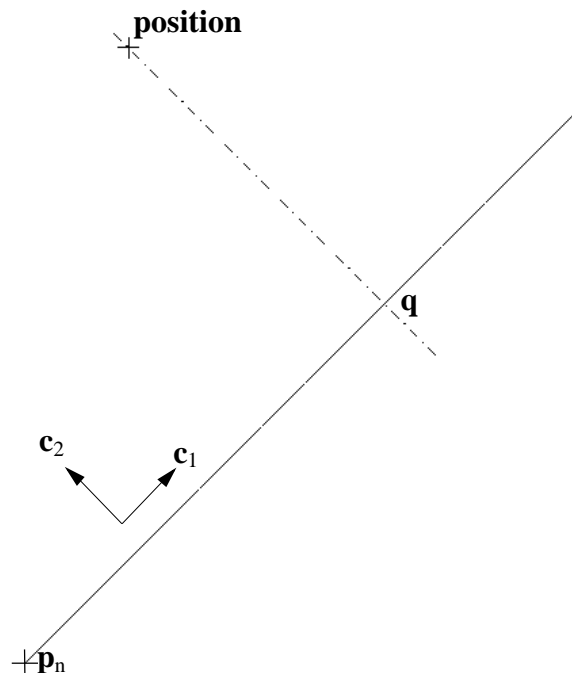


Figure 7: Distance to a line

$$\mathbf{x}_1 = \mathbf{p}_n + r\mathbf{c}_1 \quad (3.2.18)$$

$$\mathbf{x}_2 = \mathbf{position} + h\mathbf{c}_2 \quad (3.2.19)$$

where (3.2.19) describes the straight line and (3.2.20) the perpendicular through the position of the cobot.

By setting  $\mathbf{x}_1 = \mathbf{x}_2$ , we get two conditions - one for each parameter - for the point of intersection

$$r = \frac{(position_x - p_{nx})c_{2y} - (position_{ny} - p_{ny})c_{2x}}{c_{1x}c_{2y} - c_{1y}c_{2x}} \quad (3.2. 20)$$

$$h = \frac{(position_x - p_{nx})c_{1y} - (position_{ny} - p_{ny})c_{1x}}{c_{1x}c_{2y} - c_{1y}c_{2x}} \quad (3.2. 21)$$

$|r|$  is the distance of the closest point  $\mathbf{q}$  to  $\mathbf{p}_n$  and  $|h|$  is the distance from the cobot to the line.  $|r|$  has to be greater than  $D$  of the clothoid and shorter than the length of the line minus  $D$  of the following corner.

The point of intersection is found to be

$$\mathbf{q} = \mathbf{p}_n + r\mathbf{c}_1 \quad (3.2. 22)$$

This computation is done for every line. Since the last line is the same as the first one, we do not need to compute the distance to it. The line with the smallest  $|h|$  is the closest one. Finally, we have to compare the smallest  $|h|$  with the distance to the clothoid (if it was computed). The smaller value has to be taken and the corresponding element is the closest one for which we have to compute the tangent and curvature. For the case that a line is closest, the tangent is simply the direction of the line and the curvature is zero.

$$\begin{aligned} \mathbf{T} &= \mathbf{c}_1 \\ \kappa &= 0 \end{aligned} \quad (3.2. 23)$$

The computation for the clothoid is shown in the previous chapter.

### 3.3 Computation time

The collision detection should be in real time. In order to have an idea how efficient the program works, the time needed to run the function `closest_element` is measured. This function includes all the computation mentioned in the previous chapter about collision detection.

For all computations a Pentium II processor (266 MHz) was used. In order to be possible to measure the time and to get sufficiently correct results, the function is called several times. Since the computation to a straight line is much less time consuming than

the Newton-Raphson iteration, we chose to run the function 10000 times to determine the time needed to find the closest line and 1000 times to obtain results for the Newton-Raphson iteration. Thus, the following data is obtained:

	<b>Lines</b>	<b>Clothoid (N.-R. iteration)</b>
<b>Time [milliseconds]</b>	0.024	1.632

These times are sufficiently small for our real time purposes. The little time necessary for the collision detection supports the use of the Newton-Raphson iteration and the chosen “algorithm” in general.

This chapter covered the necessary computation included in the program. The clothoids are being implemented for the purpose of blending two straight lines. Further, the collision detection showed that it is necessary to determine the closest corner to the actual position and from this we separated the xy-plane in an area close to the clothoid ( $A_1$ ) and one including the straight walls ( $A_2$ ). For a cobot being in  $A_1$  it was showed how the closest point on the clothoid is determined. We need the scalar product  $\mathbf{a} \cdot \mathbf{T}$  of the vector from the actual position to the curve  $\mathbf{a}$  and the tangent  $\mathbf{T}$  of the curve. The distance is minimal when these two vectors are perpendicular. Since we have no rational form of the clothoid we determine the null of the scalar product by applying Newton-Raphson iteration.

Further, we have to determine the distance to all lines. At the end of this calculation we found the closest element for which, eventually, the tangent and the curvature are computed.

Then we considered the computation time, which is very crucial for our purposes (real time). We found that it is sufficiently small and that it is reasonable to apply Newton-Raphson iteration, although it needs the most computation time.

## 4. Program

The program code (Appendix B) is written in C for a Watcom compiler<sup>2</sup>.

It consists of 8 parts: The main program, the six used functions, which are stored in separate files, and one h-file. The separated structure should support the user who wants to change or add parts. In addition the programs should be sufficiently documented for better understanding. But the variable names used in the program are not all the same as the ones used in the previous chapters.

The main program contains all the necessary inputs: definition of points and  $\rho$  for each corner. After that, it computes the lines and the clothoids for these inputs and plots the path. In addition, it calls the function `closest_element`, which is the key for the collision detection.

Necessary for the collision detection are following functions defined in the program: `closest_element`, `closest_point`, `closest_line`, `derivativezero` (using the fresnel function). We here describe them briefly.

⇒ `closest_element`:

This function determines the closest element. It follows the steps of the collision detection “algorithm” described in 3.2. and shown in Appendix A. It calls the following functions and returns the distance, the tangent, and the curvature to the main program.

⇒ `closest_point`:

Computes the distance to all corner points. Since their coordinates are already known and stored in an array, only the number of the closest point and the corresponding distance is returned to `closest_element`.

⇒ `derivativezero`:

The Newton-Raphson iteration is implemented in this subroutine. It is called if the cobot is in area  $A_1$  (chapter 3.2.2: Figure 6). We introduce the parameter  $si$  to make the distinction between computing the distance to a proximal or to a distal element. According to the value of  $si$  (+1 or -1) the correct parameter  $p$  and coordinate frame

---

<sup>2</sup> The mouse support has to be modified when a different compiler is used.

are used. The returned values are:  $p$  for the closest point, and the distance from the position of the cobot to this point.

⇒ `closest_line`:

This function is called for each line. It is important that we reference the correct coordinate frame. For all lines of the path we use the proximal coordinate frame. The function returns the two parameters  $r$  and  $s$  (as in chapter 3.2.4). With these parameters, the coordinates of the closest point and the tangent are computed in `closest_element` (the curvature is zero).

The *fresnel* function, which is used in the main program as well as in *derivativezero*, simply computes points on a clothoid using Heald's lowest order approximation as described in chapter 3.1.1.

The last function - *mousesupport* - is responsible for the mouse support. It allows the user the input of points with the help of the mouse: defining points with the left mouse button and terminating the input with the right mouse button. It further displays the actual mouse position in the upper right corner of the screen.

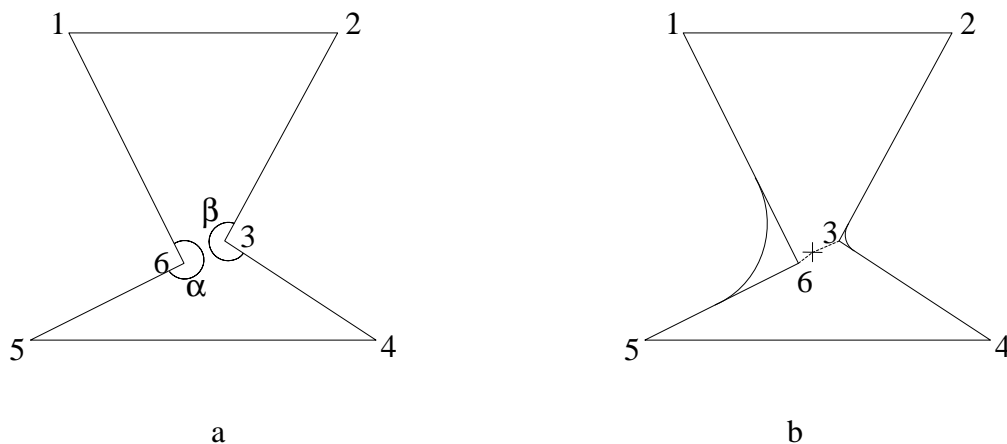
## 5. Problems to be solved and future work

The methodology and the program described in the previous sections are constrained to certain polygon shapes. The user must be aware of these limitations prior to using the program.

Here we discuss two problems: A polygon with at least one inner angle greater than  $180^\circ$ , and a path with intersecting elements. Both of them cause ambiguity in the collision detection part of the program.

### 5.1. Polygon with inner angles greater than $180^\circ$

Consider the polygon shown in Figure 8



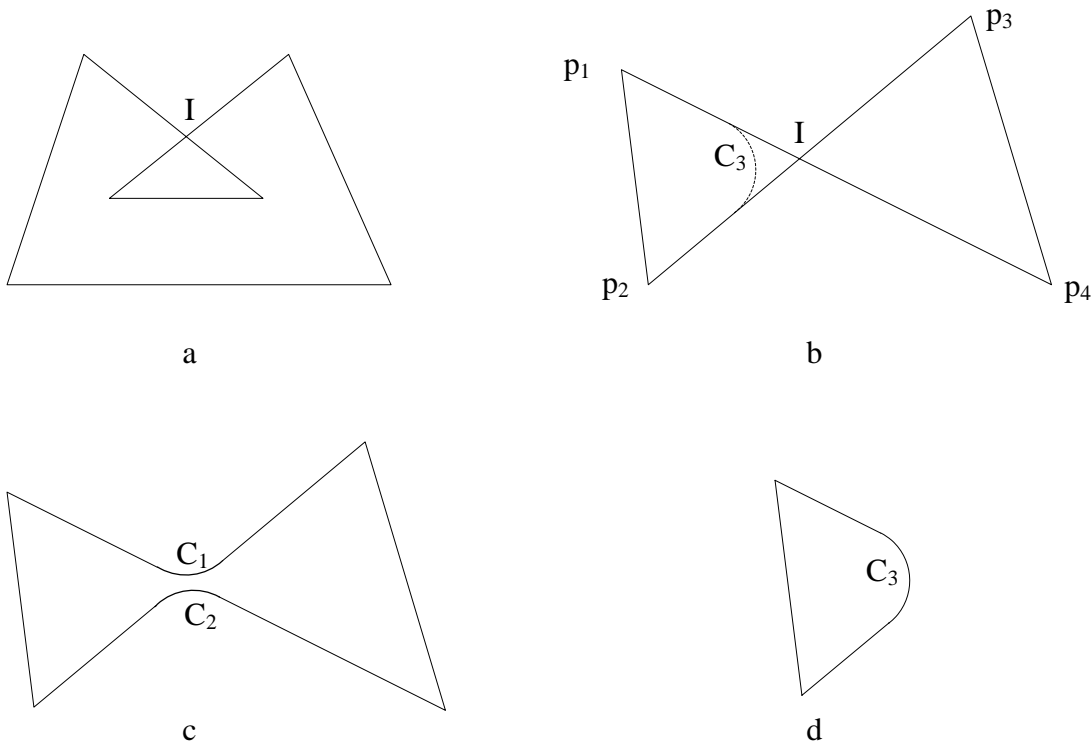
**Figure 8**

As can be seen in Figure 8, the inner angles at the corners 3 and 6 ( $\alpha$  and  $\beta$ , respectively) are greater than  $180^\circ$ . The program fails to determine the correct collision point for the following reason, as illustrated in Figure 8.b. The determination of the closest point results in the wrong decision criteria. Point 6 is computed to be the closest point. But due to the choice of  $\rho$ , it can be seen, that the clothoid at corner point 3 would be closest to the actual position of the cobot (shown as cross in Figure 8.b ). Since the program

computes only the distance to the clothoid at the “closest” point, it will compute the wrong distance.

## 5.2. A path with intersecting elements

The paths shown in Figure 9.a and b have intersecting lines (intersections are marked as “I” in the figure).



**Figure 9**

Thus, in addition to the corners of the polygon, these intersections represent new, generated, singularity points. These are, in a way, of a different “nature” than the corners. These intersection can occur if the user ends the input of points for example at point p<sub>4</sub> in Figure 9.b. Then the program would add p<sub>1</sub> as last point in order to close the polygon. As a result the line between p<sub>2</sub> and p<sub>3</sub> would intersect the line between p<sub>4</sub> and p<sub>1</sub>. Assuming that the user wanted to reach all the points chosen, Figure 9.c shows a path that avoids the

singularity I (in Figure 9.b), but still keeps the shape approximate to the initially defined one. This requires that intersection I is substituted by clothoids  $C_1$  and  $C_2$ . The goal is to eliminate the possibility of choosing, for example, clothoid  $C_3$  in the attempt to avoid I. This would, obviously, eliminate a big portion of the desired path and result in a shape shown approximately in Figure 9.d.

## 6. Conclusion

In this paper we developed a graphical user interface for defining a path in the xy-plane. This path is supposed to be the constraint for a cobot movement. The mouse supports the choice of arbitrary points. These points are considered to be the corner points of the path and are connected by straight lines. Further, we introduced a curve, namely clothoid that blends two adjacent lines in order to avoid a sharp transition at the corner.

After defining the path it became necessary to find a way to detect the closest point on the path to the actual position of the cobot. This collision detection requires an algorithm that avoids unneeded computation to increase efficiency and thus, minimize computation time. This is necessary, since the computation should be in real time. As we have seen the time consumed by the algorithm is sufficiently small. Only the distance to a clothoid requires more time since it involves Newton-Raphson iteration.

However, since there are many degrees of freedom the user must be aware of certain problems. The inner angles of the polygon have to be lower than  $180^\circ$  otherwise the collision detection can fail. On the other hand an intersection of elements should be avoided, since a point of intersection would cause a new singularity, that cannot be eluded easily.

## Reference

- [1] Colgate, J. E.: Cobot Control Notes: Clothoid Primer<sup>3</sup>
- [2] Colgate, J. E.: Cobot Control Notes: Fine Approach<sup>3</sup>
- [3] Heald, Mark A.: “Rational Approximations for Fresnel Integrals”, *Mathematics of Computation*, Vol. 44, No. 170, April 1985, pp 459-461
- [4] “Inside R&D”, Vol 25, No. 50, Dec 11 1996
- [5] Colgate J. E., Peshkin M. A., Wanasuphprasit W.:“ Cobots: Robots for collaboration with human operators”, *Proceedings of the International Mechanical Engineering Congress and Exhibition, Atlanta, GA, DSC-Vol. 58*, pp. 433-39
- [6] Wanasuphprasit W., Gillespie B. R., Colgate J. E., Peshkin M. A.: “Cobot Control”, *Proceedings of the 1997 IEEE International Conference on Robotics and Automation, Albuquerque, NM*, pp. 3571-3577

---

<sup>3</sup> Obtained through personal communication with Professor Colgate, Northwestern University